MySQL Reference Manual

MySQL Reference Manual

Table of Contents

Preface xxii
1. General Information 1
1.1. About This Manual 1
1.1.1. Conventions Used in This Manual 2
1.2. Overview of the MySQL Database Management System 4
1.2.1. History of MySQL 5
1.2.2. The Main Features of MySQL 5
1.2.3. MySQL Stability 8
1.2.4. How Big MySQL Tables Can Be 9
1.2.5. Year 2000 Compliance 10
1.3. MySQL Development Roadmap 11
1.3.1. MySQL 4.0 in a Nutshell 12
1.3.2. MySQL 4.1 in a Nutshell 14
1.3.3. MySQL 5.0: The Next Development Release 15
1.4. MySQL Information Sources 16
1.4.1. MySQL Mailing Lists 16
1.4.2. MySQL Community Support on IRC (Internet Relay Chat) 23
1.4.3. MySQL Community Support at the MySQL Forums 23
1.5. MySQL Standards Compliance 23
1.5.1. What Standards MySQL Follows 24
1.5.2. Selecting SQL Modes 24
1.5.3. Running MySQL in ANSI Mode 24
1.5.4. MySQL Extensions to Standard SQL 25
1.5.5. MySQL Differences from Standard SQL 28
1.5.6. How MySQL Deals with Constraints 34
1.5.7. Known Errors and Design Deficiencies in MySQL 36
2. Installing MySQL 43
2.1. General Installation Issues 43
2.1.1. Operating Systems Supported by MySQL 44
2.1.2. Choosing Which MySQL Distribution to Install 46
2.1.3. How to Get MySQL 57
2.1.4. Verifying Package Integrity Using MD5 Checksums or `GnuPG` 57
2.1.5. Installation Layouts 60
2.2. Standard MySQL Installation Using a Binary Distribution 61
2.3. Installing MySQL on Windows 61
2.3.1. Windows System Requirements 62
2.3.2. Choosing An Installation Package 63
2.3.3. Installing MySQL with the Automated Installer 63
2.3.4. Using the MySQL Installation Wizard 64
2.3.5. Using the Configuration Wizard 67
2.3.6. Installing MySQL from a noinstall Zip Archive 71
2.3.7. Extracting the Install Archive 72
2.3.8. Creating an Option File 72
2.3.9. Selecting a MySQL Server type 73
2.3.10. Starting the Server for the First Time 74
2.3.11. Starting MySQL from the Windows Command Line 76
2.3.12. Starting MySQL as a Windows Service 76
2.3.13. Testing The MySQL Installation 79
2.3.14. Troubleshooting a MySQL Installation Under Windows 79
2.3.15. Upgrading MySQL on Windows 81
2.3.16. MySQL on Windows Compared to MySQL on Unix 82
2.4. Installing MySQL on Linux 85
2.5. Installing MySQL on Mac OS X 87

2.6. Installing MySQL on NetWare ... 89
2.7. Installing MySQL on Other Unix-Like Systems ... 91
2.8. MySQL Installation Using a Source Distribution ... 94
2.8.1. Source Installation Overview ... 95
2.8.2. Typical **configure** Options ... 98
2.8.3. Installing from the Development Source Tree ... 101
2.8.4. Dealing with Problems Compiling MySQL ... 104
2.8.5. MIT-pthreads Notes ... 107
2.8.6. Installing MySQL from Source on Windows ... 109
2.8.7. Compiling MySQL Clients on Windows ... 113
2.9. Post-Installation Setup and Testing ... 113
2.9.1. Windows Post-Installation Procedures ... 113
2.9.2. Unix Post-Installation Procedures ... 114
2.9.3. Securing the Initial MySQL Accounts ... 125
2.10. Upgrading MySQL ... 128
2.10.1. Upgrading from Version 4.1 to 5.0 ... 129
2.10.2. Upgrading from Version 4.0 to 4.1 ... 131
2.10.3. Upgrading from Version 3.23 to 4.0 ... 137
2.10.4. Upgrading from Version 3.22 to 3.23 ... 141
2.10.5. Upgrading from Version 3.21 to 3.22 ... 142
2.10.6. Upgrading from Version 3.20 to 3.21 ... 143
2.10.7. Upgrading the Grant Tables ... 144
2.10.8. Copying MySQL Databases to Another Machine ... 145
2.11. Downgrading MySQL ... 146
2.11.1. Downgrading to 4.0 ... 147
2.12. Operating System-Specific Notes ... 147
2.12.1. Linux Notes ... 147
2.12.2. Mac OS X Notes ... 154
2.12.3. Solaris Notes ... 155
2.12.4. BSD Notes ... 159
2.12.5. Other Unix Notes ... 162
2.12.6. OS/2 Notes ... 175
2.12.7. BeOS Notes ... 176
2.13. Perl Installation Notes ... 176
2.13.1. Installing Perl on Unix ... 176
2.13.2. Installing ActiveState Perl on Windows ... 177
2.13.3. Problems Using the Perl `DBI`/`DBD` Interface ... 178
3. MySQL Tutorial ... 181
3.1. Connecting to and Disconnecting from the Server ... 181
3.2. Entering Queries ... 182
3.3. Creating and Using a Database ... 185
3.3.1. Creating and Selecting a Database ... 186
3.3.2. Creating a Table ... 187
3.3.3. Loading Data into a Table ... 188
3.3.4. Retrieving Information from a Table ... 189
3.4. Getting Information About Databases and Tables ... 203
3.5. Using **mysql** in Batch Mode ... 204
3.6. Examples of Common Queries ... 205
3.6.1. The Maximum Value for a Column ... 206
3.6.2. The Row Holding the Maximum of a Certain Column ... 206
3.6.3. Maximum of Column per Group ... 207
3.6.4. The Rows Holding the Group-wise Maximum of a Certain Field ... 207
3.6.5. Using User Variables ... 208
3.6.6. Using Foreign Keys ... 209
3.6.7. Searching on Two Keys ... 210
3.6.8. Calculating Visits Per Day ... 211
3.6.9. Using `AUTO_INCREMENT` ... 211
3.7. Queries from the Twin Project ... 212

3.7.1. Find All Non-distributed Twins ... 213
3.7.2. Show a Table of Twin Pair Status ... 215
3.8. Using MySQL with Apache ... 215
4. Using MySQL Programs ... 217
4.1. Overview of MySQL Programs ... 217
4.2. Invoking MySQL Programs ... 218
4.3. Specifying Program Options ... 218
4.3.1. Using Options on the Command Line ... 219
4.3.2. Using Option Files ... 220
4.3.3. Using Environment Variables to Specify Options ... 224
4.3.4. Using Options to Set Program Variables ... 225
5. Database Administration ... 227
5.1. The MySQL Server and Server Startup Scripts ... 227
5.1.1. Overview of the Server-Side Scripts and Utilities ... 227
5.1.2. The **mysqld-max** Extended MySQL Server ... 228
5.1.3. The **mysqld_safe** Server Startup Script ... 231
5.1.4. The **mysql.server** Server Startup Script ... 234
5.1.5. The **mysqld_multi** Program for Managing Multiple MySQL Servers ... 235
5.2. Configuring the MySQL Server ... 238
5.2.1. **mysqld** Command-Line Options ... 239
5.2.2. The Server SQL Mode ... 249
5.2.3. Server System Variables ... 255
5.2.4. Server Status Variables ... 283
5.3. The MySQL Server Shutdown Process ... 293
5.4. General Security Issues ... 295
5.4.1. General Security Guidelines ... 295
5.4.2. Making MySQL Secure Against Attackers ... 297
5.4.3. Startup Options for **mysqld** Concerning Security ... 299
5.4.4. Security Issues with `LOAD DATA LOCAL` ... 300
5.5. The MySQL Access Privilege System ... 301
5.5.1. What the Privilege System Does ... 301
5.5.2. How the Privilege System Works ... 301
5.5.3. Privileges Provided by MySQL ... 306
5.5.4. Connecting to the MySQL Server ... 309
5.5.5. Access Control, Stage 1: Connection Verification ... 310
5.5.6. Access Control, Stage 2: Request Verification ... 314
5.5.7. When Privilege Changes Take Effect ... 316
5.5.8. Causes of `Access denied` Errors ... 317
5.5.9. Password Hashing in MySQL 4.1 ... 321
5.6. MySQL User Account Management ... 326
5.6.1. MySQL Usernames and Passwords ... 327
5.6.2. Adding New User Accounts to MySQL ... 328
5.6.3. Removing User Accounts from MySQL ... 331
5.6.4. Limiting Account Resources ... 331
5.6.5. Assigning Account Passwords ... 333
5.6.6. Keeping Your Password Secure ... 334
5.6.7. Using Secure Connections ... 335
5.7. Disaster Prevention and Recovery ... 342
5.7.1. Database Backups ... 343
5.7.2. Example Backup and Recovery Strategy ... 345
5.7.3. Table Maintenance and Crash Recovery ... 348
5.7.4. Setting Up a Table Maintenance Schedule ... 360
5.7.5. Getting Information About a Table ... 361
5.8. MySQL Localization and International Usage ... 367
5.8.1. The Character Set Used for Data and Sorting ... 367
5.8.2. Setting the Error Message Language ... 369
5.8.3. Adding a New Character Set ... 369
5.8.4. The Character Definition Arrays ... 371

5.8.5. String Collating Support ... 371
5.8.6. Multi-Byte Character Support ... 372
5.8.7. Problems With Character Sets ... 372
5.8.8. MySQL Server Time Zone Support ... 372
5.9. The MySQL Log Files ... 374
5.9.1. The Error Log ... 374
5.9.2. The General Query Log ... 375
5.9.3. The Update Log ... 375
5.9.4. The Binary Log ... 376
5.9.5. The Slow Query Log ... 379
5.9.6. Log File Maintenance ... 380
5.10. Running Multiple MySQL Servers on the Same Machine ... 381
5.10.1. Running Multiple Servers on Windows ... 382
5.10.2. Running Multiple Servers on Unix ... 386
5.10.3. Using Client Programs in a Multiple-Server Environment ... 387
5.11. The MySQL Query Cache ... 388
5.11.1. How the Query Cache Operates ... 388
5.11.2. Query Cache `SELECT` Options ... 390
5.11.3. Query Cache Configuration ... 390
5.11.4. Query Cache Status and Maintenance ... 391
6. Replication in MySQL ... 393
6.1. Introduction to Replication ... 393
6.2. Replication Implementation Overview ... 393
6.3. Replication Implementation Details ... 394
6.3.1. Replication Master Thread States ... 395
6.3.2. Replication Slave I/O Thread States ... 396
6.3.3. Replication Slave SQL Thread States ... 397
6.3.4. Replication Relay and Status Files ... 397
6.4. How to Set Up Replication ... 399
6.5. Replication Compatibility Between MySQL Versions ... 404
6.6. Upgrading a Replication Setup ... 404
6.6.1. Upgrading Replication to 4.0 or 4.1 ... 404
6.6.2. Upgrading Replication to 5.0 ... 405
6.7. Replication Features and Known Problems ... 405
6.8. Replication Startup Options ... 409
6.9. Replication FAQ ... 418
6.10. Troubleshooting Replication ... 423
6.11. Reporting Replication Bugs ... 424
7. MySQL Optimization ... 426
7.1. Optimization Overview ... 426
7.1.1. MySQL Design Limitations and Tradeoffs ... 426
7.1.2. Designing Applications for Portability ... 427
7.1.3. What We Have Used MySQL For ... 428
7.1.4. The MySQL Benchmark Suite ... 428
7.1.5. Using Your Own Benchmarks ... 429
7.2. Optimizing `SELECT` Statements and Other Queries ... 430
7.2.1. `EXPLAIN` Syntax (Get Information About a `SELECT`) ... 430
7.2.2. Estimating Query Performance ... 439
7.2.3. Speed of `SELECT` Queries ... 440
7.2.4. How MySQL Optimizes `WHERE` Clauses ... 440
7.2.5. Range Optimization ... 442
7.2.6. Index Merge Optimization ... 446
7.2.7. How MySQL Optimizes `IS NULL` ... 448
7.2.8. How MySQL Optimizes `DISTINCT` ... 449
7.2.9. How MySQL Optimizes `LEFT JOIN` and `RIGHT JOIN` ... 450
7.2.10. How MySQL Optimizes `ORDER BY` ... 451
7.2.11. How MySQL Optimizes `GROUP BY` ... 453
7.2.12. How MySQL Optimizes `LIMIT` ... 455

7.2.13. How to Avoid Table Scans ... 455
7.2.14. Speed of `INSERT` Statements ... 456
7.2.15. Speed of `UPDATE` Statements ... 458
7.2.16. Speed of `DELETE` Statements ... 458
7.2.17. Other Optimization Tips ... 458
7.3. Locking Issues ... 461
7.3.1. Locking Methods ... 461
7.3.2. Table Locking Issues ... 463
7.4. Optimizing Database Structure ... 465
7.4.1. Design Choices ... 465
7.4.2. Make Your Data as Small as Possible ... 465
7.4.3. Column Indexes ... 466
7.4.4. Multiple-Column Indexes ... 467
7.4.5. How MySQL Uses Indexes ... 468
7.4.6. The `MyISAM` Key Cache ... 470
7.4.7. How MySQL Counts Open Tables ... 475
7.4.8. How MySQL Opens and Closes Tables ... 475
7.4.9. Drawbacks to Creating Many Tables in the Same Database ... 477
7.5. Optimizing the MySQL Server ... 477
7.5.1. System Factors and Startup Parameter Tuning ... 477
7.5.2. Tuning Server Parameters ... 478
7.5.3. Controlling Query Optimizer Performance ... 480
7.5.4. How Compiling and Linking Affects the Speed of MySQL ... 481
7.5.5. How MySQL Uses Memory ... 482
7.5.6. How MySQL Uses DNS ... 483
7.6. Disk Issues ... 484
7.6.1. Using Symbolic Links ... 485
8. MySQL Client and Utility Programs ... 489
8.1. Overview of the Client-Side Scripts and Utilities ... 489
8.2. **myisampack**, the MySQL Compressed Read-only Table Generator ... 491
8.3. **mysql**, the Command-Line Tool ... 497
8.3.1. **mysql** Commands ... 503
8.3.2. Executing SQL Statements from a Text File ... 506
8.3.3. **mysql** Tips ... 507
8.4. **mysqladmin**, Administering a MySQL Server ... 508
8.5. The **mysqlbinlog** Binary Log Utility ... 513
8.6. **mysqlcc**, the MySQL Control Center ... 517
8.7. The **mysqlcheck** Table Maintenance and Repair Program ... 520
8.8. The **mysqldump** Database Backup Program ... 523
8.9. The **mysqlhotcopy** Database Backup Program ... 530
8.10. The **mysqlimport** Data Import Program ... 532
8.11. **mysqlshow**, Showing Databases, Tables, and Columns ... 534
8.12. **perror**, Explaining Error Codes ... 536
8.13. The **replace** String-Replacement Utility ... 537
9. Language Structure ... 538
9.1. Literal Values ... 538
9.1.1. Strings ... 538
9.1.2. Numbers ... 540
9.1.3. Hexadecimal Values ... 540
9.1.4. Boolean Values ... 541
9.1.5. Bit-Field Values ... 541
9.1.6. `NULL` Values ... 541
9.2. Database, Table, Index, Column, and Alias Names ... 542
9.2.1. Identifier Qualifiers ... 543
9.2.2. Identifier Case Sensitivity ... 544
9.3. User Variables ... 545
9.4. System Variables ... 546
9.4.1. Structured System Variables ... 547

9.5. Comment Syntax ... 549
9.6. Treatment of Reserved Words in MySQL ... 550
10. Character Set Support ... 554
10.1. Character Sets and Collations in General ... 554
10.2. Character Sets and Collations in MySQL ... 555
10.3. Determining the Default Character Set and Collation ... 556
10.3.1. Server Character Set and Collation ... 556
10.3.2. Database Character Set and Collation ... 557
10.3.3. Table Character Set and Collation ... 558
10.3.4. Column Character Set and Collation ... 558
10.3.5. Examples of Character Set and Collation Assignment ... 559
10.3.6. Connection Character Sets and Collations ... 560
10.3.7. Character String Literal Character Set and Collation ... 561
10.3.8. Using `COLLATE` in SQL Statements ... 563
10.3.9. `COLLATE` Clause Precedence ... 563
10.3.10. `BINARY` Operator ... 564
10.3.11. Some Special Cases Where the Collation Determination Is Tricky ... 564
10.3.12. Collations Must Be for the Right Character Set ... 565
10.3.13. An Example of the Effect of Collation ... 566
10.4. Operations Affected by Character Set Support ... 567
10.4.1. Result Strings ... 567
10.4.2. `CONVERT()` ... 567
10.4.3. `CAST()` ... 568
10.4.4. `SHOW` Statements ... 568
10.5. Unicode Support ... 569
10.6. UTF8 for Metadata ... 570
10.7. Compatibility with Other DBMSs ... 571
10.8. New Character Set Configuration File Format ... 571
10.9. National Character Set ... 572
10.10. Upgrading Character Sets from MySQL 4.0 ... 572
10.10.1. 4.0 Character Sets and Corresponding 4.1 Character Set/Collation Pairs . 573
10.10.2. Converting 4.0 Character Columns to 4.1 Format ... 574
10.11. Character Sets and Collations That MySQL Supports ... 575
10.11.1. Unicode Character Sets ... 575
10.11.2. West European Character Sets ... 578
10.11.3. Central European Character Sets ... 579
10.11.4. South European and Middle East Character Sets ... 580
10.11.5. Baltic Character Sets ... 581
10.11.6. Cyrillic Character Sets ... 581
10.11.7. Asian Character Sets ... 582
11. Column Types ... 584
11.1. Column Type Overview ... 584
11.1.1. Overview of Numeric Types ... 584
11.1.2. Overview of Date and Time Types ... 587
11.1.3. Overview of String Types ... 588
11.2. Numeric Types ... 592
11.3. Date and Time Types ... 594
11.3.1. The `DATETIME`, `DATE`, and `TIMESTAMP` Types ... 595
11.3.2. The `TIME` Type ... 602
11.3.3. The `YEAR` Type ... 603
11.3.4. Y2K Issues and Date Types ... 603
11.4. String Types ... 603
11.4.1. The `CHAR` and `VARCHAR` Types ... 604
11.4.2. The `BINARY` and `VARBINARY` Types ... 605
11.4.3. The `BLOB` and `TEXT` Types ... 605
11.4.4. The `ENUM` Type ... 607
11.4.5. The `SET` Type ... 608
11.5. Column Type Storage Requirements ... 609

11.6. Choosing the Right Type for a Column ... 611
11.7. Using Column Types from Other Database Engines ... 611
12. Functions and Operators ... 613
12.1. Operators ... 613
12.1.1. Operator Precedence ... 613
12.1.2. Parentheses ... 614
12.1.3. Comparison Functions and Operators ... 614
12.1.4. Logical Operators ... 620
12.2. Control Flow Functions ... 622
12.3. String Functions ... 624
12.3.1. String Comparison Functions ... 637
12.4. Numeric Functions ... 639
12.4.1. Arithmetic Operators ... 639
12.4.2. Mathematical Functions ... 641
12.5. Date and Time Functions ... 649
12.6. Full-Text Search Functions ... 670
12.6.1. Boolean Full-Text Searches ... 672
12.6.2. Full-Text Searches with Query Expansion ... 675
12.6.3. Full-Text Restrictions ... 675
12.6.4. Fine-Tuning MySQL Full-Text Search ... 676
12.6.5. Full-Text Search TODO ... 677
12.7. Cast Functions and Operators ... 678
12.8. Other Functions ... 681
12.8.1. Bit Functions ... 681
12.8.2. Encryption Functions ... 683
12.8.3. Information Functions ... 686
12.8.4. Miscellaneous Functions ... 693
12.9. Functions and Modifiers for Use with `GROUP BY` Clauses ... 696
12.9.1. `GROUP BY` (Aggregate) Functions ... 696
12.9.2. `GROUP BY` Modifiers ... 700
12.9.3. `GROUP BY` with Hidden Fields ... 703
13. SQL Statement Syntax ... 704
13.1. Data Manipulation Statements ... 704
13.1.1. `DELETE` Syntax ... 704
13.1.2. `DO` Syntax ... 707
13.1.3. `HANDLER` Syntax ... 707
13.1.4. `INSERT` Syntax ... 708
13.1.5. `LOAD DATA INFILE` Syntax ... 714
13.1.6. `REPLACE` Syntax ... 722
13.1.7. `SELECT` Syntax ... 723
13.1.8. Subquery Syntax ... 732
13.1.9. `TRUNCATE` Syntax ... 743
13.1.10. `UPDATE` Syntax ... 744
13.2. Data Definition Statements ... 745
13.2.1. `ALTER DATABASE` Syntax ... 746
13.2.2. `ALTER TABLE` Syntax ... 746
13.2.3. `ALTER VIEW` Syntax ... 751
13.2.4. `CREATE DATABASE` Syntax ... 751
13.2.5. `CREATE INDEX` Syntax ... 752
13.2.6. `CREATE TABLE` Syntax ... 753
13.2.7. `CREATE VIEW` Syntax ... 765
13.2.8. `DROP DATABASE` Syntax ... 770
13.2.9. `DROP INDEX` Syntax ... 771
13.2.10. `DROP TABLE` Syntax ... 771
13.2.11. `DROP VIEW` Syntax ... 771
13.2.12. `RENAME TABLE` Syntax ... 772
13.3. MySQL Utility Statements ... 772
13.3.1. `DESCRIBE` Syntax (Get Information About Columns) ... 772

13.3.2. USE Syntax ... 773
13.4. MySQL Transactional and Locking Statements ... 774
13.4.1. START TRANSACTION, COMMIT, and ROLLBACK Syntax ... 774
13.4.2. Statements That Cannot Be Rolled Back ... 775
13.4.3. Statements That Cause an Implicit Commit ... 775
13.4.4. SAVEPOINT and ROLLBACK TO SAVEPOINT Syntax ... 776
13.4.5. LOCK TABLES and UNLOCK TABLES Syntax ... 776
13.4.6. SET TRANSACTION Syntax ... 779
13.5. Database Administration Statements ... 779
13.5.1. Account Management Statements ... 779
13.5.2. Table Maintenance Statements ... 789
13.5.3. SET Syntax ... 794
13.5.4. SHOW Syntax ... 799
13.5.5. Other Administrative Statements ... 816
13.6. Replication Statements ... 821
13.6.1. SQL Statements for Controlling Master Servers ... 821
13.6.2. SQL Statements for Controlling Slave Servers ... 823
13.7. SQL Syntax for Prepared Statements ... 831
14. MySQL Storage Engines and Table Types ... 834
14.1. The MyISAM Storage Engine ... 836
14.1.1. MyISAM Startup Options ... 838
14.1.2. Space Needed for Keys ... 839
14.1.3. MyISAM Table Storage Formats ... 839
14.1.4. MyISAM Table Problems ... 842
14.2. The MERGE Storage Engine ... 843
14.2.1. MERGE Table Problems ... 845
14.3. The MEMORY (HEAP) Storage Engine ... 846
14.4. The BDB (BerkeleyDB) Storage Engine ... 848
14.4.1. Operating Systems Supported by BDB ... 849
14.4.2. Installing BDB ... 849
14.4.3. BDB Startup Options ... 850
14.4.4. Characteristics of BDB Tables ... 851
14.4.5. Things We Need to Fix for BDB ... 853
14.4.6. Restrictions on BDB Tables ... 853
14.4.7. Errors That May Occur When Using BDB Tables ... 853
14.5. The EXAMPLE Storage Engine ... 854
14.6. The FEDERATED Storage Engine ... 854
14.6.1. Installing the FEDERATED Storage Engine ... 854
14.6.2. Description of the FEDERATED Storage Engine ... 854
14.6.3. How to use FEDERATED Tables ... 855
14.6.4. Limitations of the FEDERATED Storage Engine ... 856
14.7. The ARCHIVE Storage Engine ... 857
14.8. The CSV Storage Engine ... 857
14.9. The ISAM Storage Engine ... 858
15. The InnoDB Storage Engine ... 860
15.1. InnoDB Overview ... 860
15.2. InnoDB Contact Information ... 860
15.3. InnoDB in MySQL 3.23 ... 860
15.4. InnoDB Configuration ... 861
15.5. InnoDB Startup Options ... 866
15.6. Creating the InnoDB Tablespace ... 871
15.6.1. Dealing with InnoDB Initialization Problems ... 872
15.7. Creating InnoDB Tables ... 873
15.7.1. How to Use Transactions in InnoDB with Different APIs ... 873
15.7.2. Converting MyISAM Tables to InnoDB ... 874
15.7.3. How an AUTO_INCREMENT Column Works in InnoDB ... 875
15.7.4. FOREIGN KEY Constraints ... 875
15.7.5. InnoDB and MySQL Replication ... 879

15.7.6. Using Per-Table Tablespaces ... 880
15.8. Adding and Removing InnoDB Data and Log Files ... 882
15.9. Backing Up and Recovering an InnoDB Database ... 883
15.9.1. Forcing Recovery ... 884
15.9.2. Checkpoints ... 885
15.10. Moving an InnoDB Database to Another Machine ... 886
15.11. InnoDB Transaction Model and Locking ... 886
15.11.1. InnoDB and AUTOCOMMIT ... 886
15.11.2. InnoDB and TRANSACTION ISOLATION LEVEL ... 887
15.11.3. Consistent Non-Locking Read ... 888
15.11.4. Locking Reads SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE ... 888
15.11.5. Next-Key Locking: Avoiding the Phantom Problem ... 889
15.11.6. An Example of How the Consistent Read Works in InnoDB ... 890
15.11.7. Locks Set by Different SQL Statements in InnoDB ... 891
15.11.8. When Does MySQL Implicitly Commit or Roll Back a Transaction? ... 892
15.11.9. Deadlock Detection and Rollback ... 892
15.11.10. How to Cope with Deadlocks ... 893
15.12. InnoDB Performance Tuning Tips ... 894
15.12.1. SHOW INNODB STATUS and the InnoDB Monitors ... 895
15.13. Implementation of Multi-Versioning ... 900
15.14. Table and Index Structures ... 901
15.14.1. Physical Structure of an Index ... 901
15.14.2. Insert Buffering ... 901
15.14.3. Adaptive Hash Indexes ... 902
15.14.4. Physical Record Structure ... 902
15.15. File Space Management and Disk I/O ... 903
15.15.1. Disk I/O ... 903
15.15.2. Using Raw Devices for the Tablespace ... 903
15.15.3. File Space Management ... 904
15.15.4. Defragmenting a Table ... 904
15.16. Error Handling ... 905
15.16.1. InnoDB Error Codes ... 905
15.16.2. Operating System Error Codes ... 906
15.17. Restrictions on InnoDB Tables ... 911
15.18. InnoDB Troubleshooting ... 913
15.18.1. Troubleshooting InnoDB Data Dictionary Operations ... 914
16. MySQL Cluster ... 916
16.1. MySQL Cluster Glossary ... 916
16.2. MySQL Cluster Overview ... 919
16.3. Basic MySQL Cluster Concepts ... 921
16.4. Simple Multi-Computer How-To ... 922
16.4.1. Hardware, Software, and Networking ... 924
16.4.2. Installation ... 924
16.4.3. Configuration ... 926
16.4.4. Initial Startup ... 928
16.4.5. Loading Sample Data and Performing Queries ... 929
16.4.6. Safe Shutdown and Restart ... 932
16.5. MySQL Cluster Configuration ... 933
16.5.1. Building from Source Code ... 933
16.5.2. Installing the Software ... 933
16.5.3. Quick Test Setup of MySQL Cluster ... 933
16.5.4. Configuration File ... 936
16.6. Process Management in MySQL Cluster ... 960
16.6.1. MySQL Server Process Usage for MySQL Cluster ... 960
16.6.2. **ndbd**, the Storage Engine Node Process ... 961
16.6.3. **ndb_mgmd**, the Management Server Process ... 963
16.6.4. **ndb_mgm**, the Management Client Process ... 963

16.6.5. Command Options for MySQL Cluster Processes ... 963
16.7. Management of MySQL Cluster ... 966
16.7.1. Commands in the Management Client ... 966
16.7.2. Event Reports Generated in MySQL Cluster ... 967
16.7.3. Single User Mode ... 972
16.7.4. On-line Backup of MySQL Cluster ... 973
16.8. Using High-Speed Interconnects with MySQL Cluster ... 975
16.8.1. Configuring MySQL Cluster to use SCI Sockets ... 976
16.8.2. Understanding the Impact of Cluster Interconnects ... 979
16.9. Cluster Limitations in MySQL 4.1 ... 981
16.10. MySQL Cluster FAQ ... 983
17. Introduction to MaxDB ... 990
17.1. History of MaxDB ... 990
17.2. Licensing and Support ... 990
17.3. MaxDB-Related Links ... 990
17.4. Basic Concepts of MaxDB ... 990
17.5. Feature Differences Between MaxDB and MySQL ... 990
17.6. Interoperability Features Between MaxDB and MySQL ... 991
17.7. Reserved Words in MaxDB ... 991
18. Spatial Extensions in MySQL ... 995
18.1. Introduction ... 995
18.2. The OpenGIS Geometry Model ... 996
18.2.1. The Geometry Class Hierarchy ... 996
18.2.2. Class `Geometry` ... 997
18.2.3. Class `Point` ... 998
18.2.4. Class `Curve` ... 998
18.2.5. Class `LineString` ... 999
18.2.6. Class `Surface` ... 999
18.2.7. Class `Polygon` ... 1000
18.2.8. Class `GeometryCollection` ... 1000
18.2.9. Class `MultiPoint` ... 1000
18.2.10. Class `MultiCurve` ... 1001
18.2.11. Class `MultiLineString` ... 1001
18.2.12. Class `MultiSurface` ... 1001
18.2.13. Class `MultiPolygon` ... 1001
18.3. Supported Spatial Data Formats ... 1002
18.3.1. Well-Known Text (WKT) Format ... 1002
18.3.2. Well-Known Binary (WKB) Format ... 1003
18.4. Creating a Spatially Enabled MySQL Database ... 1004
18.4.1. MySQL Spatial Data Types ... 1004
18.4.2. Creating Spatial Values ... 1005
18.4.3. Creating Spatial Columns ... 1008
18.4.4. Populating Spatial Columns ... 1009
18.4.5. Fetching Spatial Data ... 1010
18.5. Analyzing Spatial Information ... 1010
18.5.1. Geometry Format Conversion Functions ... 1011
18.5.2. `Geometry` Functions ... 1012
18.5.3. Functions That Create New Geometries from Existing Ones ... 1019
18.5.4. Functions for Testing Spatial Relations Between Geometric Objects ... 1020
18.5.5. Relations on Geometry Minimal Bounding Rectangles (MBRs) ... 1020
18.5.6. Functions That Test Spatial Relationships Between Geometries ... 1022
18.6. Optimizing Spatial Analysis ... 1023
18.6.1. Creating Spatial Indexes ... 1023
18.6.2. Using a Spatial Index ... 1025
18.7. MySQL Conformance and Compatibility ... 1026
18.7.1. GIS Features That Are Not Yet Implemented ... 1026
19. Stored Procedures and Functions ... 1028
19.1. Stored Procedures and the Grant Tables ... 1028

19.2. Stored Procedure Syntax 1029
19.2.1. CREATE PROCEDURE and CREATE FUNCTION 1029
19.2.2. ALTER PROCEDURE and ALTER FUNCTION 1031
19.2.3. DROP PROCEDURE and DROP FUNCTION 1031
19.2.4. SHOW CREATE PROCEDURE and SHOW CREATE FUNCTION 1032
19.2.5. SHOW PROCEDURE STATUS and SHOW FUNCTION STATUS 1032
19.2.6. CALL Statement 1032
19.2.7. BEGIN ... END Compound Statement 1033
19.2.8. DECLARE Statement 1033
19.2.9. Variables in Stored Procedures 1033
19.2.10. Conditions and Handlers 1034
19.2.11. Cursors 1035
19.2.12. Flow Control Constructs 1037
20. Triggers 1040
20.1. CREATE TRIGGER Syntax 1040
20.2. DROP TRIGGER Syntax 1041
20.3. Using Triggers 1041
21. The INFORMATION_SCHEMA Information Database 1044
21.1. INFORMATION_SCHEMA Tables 1045
21.1.1. The INFORMATION_SCHEMA SCHEMATA Table 1046
21.1.2. The INFORMATION_SCHEMA TABLES Table 1046
21.1.3. The INFORMATION_SCHEMA COLUMNS Table 1048
21.1.4. The INFORMATION_SCHEMA STATISTICS Table 1049
21.1.5. The INFORMATION_SCHEMA USER_PRIVILEGES Table 1050
21.1.6. The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table 1051
21.1.7. The INFORMATION_SCHEMA TABLE_PRIVILEGES Table 1051
21.1.8. The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table 1051
21.1.9. The INFORMATION_SCHEMA CHARACTER_SETS Table 1052
21.1.10. The INFORMATION_SCHEMA COLLATIONS Table 1053
21.1.11. The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table 1053
21.1.12. The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table 1054
21.1.13. The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table 1054
21.1.14. The INFORMATION_SCHEMA ROUTINES Table 1055
21.1.15. The INFORMATION_SCHEMA VIEWS Table 1057
21.1.16. Other INFORMATION_SCHEMA Tables 1058
21.2. Extensions to SHOW Statements 1058
22. MySQL APIs 1060
22.1. MySQL Program Development Utilities 1060
22.1.1. **msql2mysql**, Convert mSQL Programs for Use with MySQL 1060
22.1.2. **mysql_config**, Get compile options for compiling clients 1060
22.2. MySQL C API 1062
22.2.1. C API Data types 1062
22.2.2. C API Function Overview 1065
22.2.3. C API Function Descriptions 1069
22.2.4. C API Prepared Statements 1111
22.2.5. C API Prepared Statement Data types 1112
22.2.6. C API Prepared Statement Function Overview 1115
22.2.7. C API Prepared Statement Function Descriptions 1117
22.2.8. C API Prepared statement problems 1140
22.2.9. C API Handling of Multiple Query Execution 1141
22.2.10. C API Handling of Date and Time Values 1141
22.2.11. C API Threaded Function Descriptions 1142
22.2.12. C API Embedded Server Function Descriptions 1144
22.2.13. Common questions and problems when using the C API 1145
22.2.14. Building Client Programs 1147
22.2.15. How to Make a Threaded Client 1147
22.2.16. libmysqld, the Embedded MySQL Server Library 1149

22.3. MySQL PHP API 1153
22.3.1. Common Problems with MySQL and PHP 1154
22.4. MySQL Perl API 1154
22.5. MySQL C++ API 1154
22.5.1. Borland C++ 1155
22.6. MySQL Python API 1155
22.7. MySQL Tcl API 1155
22.8. MySQL Eiffel Wrapper 1155
23. MySQL Connectors 1156
23.1. MySQL ODBC Support 1156
23.1.1. Introduction to MyODBC 1156
23.1.2. General Information About ODBC and MyODBC 1158
23.1.3. How to Install MyODBC 1161
23.1.4. Installing MyODBC from a Binary Distribution on Windows 1162
23.1.5. Installing MyODBC from a Binary Distribution on Unix 1162
23.1.6. Installing MyODBC from a Source Distribution on Windows 1163
23.1.7. Installing MyODBC from a Source Distribution on Unix 1165
23.1.8. Installing MyODBC from the BitKeeper Development Source Tree 1171
23.1.9. MyODBC Configuration 1172
23.1.10. MyODBC Connection-Related Issues 1190
23.1.11. MyODBC and Microsoft Access 1191
23.1.12. MyODBC and Microsoft VBA and ASP 1196
23.1.13. MyODBC and Third-Party ODBC Tools 1197
23.1.14. MyODBC General Functionality 1198
23.1.15. Basic MyODBC Application Steps 1203
23.1.16. MyODBC API Reference 1204
23.1.17. MyODBC Data Types 1209
23.1.18. MyODBC Error Codes 1210
23.1.19. MyODBC With VB: ADO, DAO and RDO 1211
23.1.20. MyODBC with Microsoft .NET 1215
23.1.21. Credits 1219
23.2. MySQL Java Connectivity (JDBC) 1219
24. Error Handling in MySQL 1221
25. Extending MySQL 1253
25.1. MySQL Internals 1253
25.1.1. MySQL Threads 1253
25.1.2. MySQL Test Suite 1253
25.2. Adding New Functions to MySQL 1256
25.2.1. Features of the User-Defined Function Interface 1257
25.2.2. `CREATE FUNCTION/DROP FUNCTION` Syntax 1257
25.2.3. Adding a New User-defined Function 1258
25.2.4. Adding a New Native Function 1267
25.3. Adding New Procedures to MySQL 1268
25.3.1. Procedure Analyse 1268
25.3.2. Writing a Procedure 1268
A. Problems and Common Errors 1270
A.1. How to Determine What Is Causing a Problem 1270
A.2. Common Errors When Using MySQL Programs 1271
A.2.1. `Access denied` 1271
A.2.2. `Can't connect to [local] MySQL server` 1271
A.2.3. `Client does not support authentication protocol` 1273
A.2.4. Password Fails When Entered Interactively 1274
A.2.5. `Host '`*`host_name`*`' is blocked` 1274
A.2.6. `Too many connections` 1275
A.2.7. `Out of memory` 1275
A.2.8. `MySQL server has gone away` 1276
A.2.9. `Packet too large` 1277
A.2.10. Communication Errors and Aborted Connections 1278

A.2.11. The table is full 1279
A.2.12. Can't create/write to file 1280
A.2.13. Commands out of sync 1280
A.2.14. Ignoring user 1281
A.2.15. Table '*tbl_name*' doesn't exist 1281
A.2.16. Can't initialize character set 1282
A.2.17. File Not Found 1282
A.3. Installation-Related Issues 1283
A.3.1. Problems Linking to the MySQL Client Library 1283
A.3.2. How to Run MySQL as a Normal User 1284
A.3.3. Problems with File Permissions 1285
A.4. Administration-Related Issues 1285
A.4.1. How to Reset the Root Password 1285
A.4.2. What to Do If MySQL Keeps Crashing 1287
A.4.3. How MySQL Handles a Full Disk 1289
A.4.4. Where MySQL Stores Temporary Files 1290
A.4.5. How to Protect or Change the MySQL Socket File /tmp/mysql.sock 1291
A.4.6. Time Zone Problems 1291
A.5. Query-Related Issues 1292
A.5.1. Case Sensitivity in Searches 1292
A.5.2. Problems Using DATE Columns 1292
A.5.3. Problems with NULL Values 1293
A.5.4. Problems with Column Aliases 1294
A.5.5. Rollback Failure for Non-Transactional Tables 1295
A.5.6. Deleting Rows from Related Tables 1295
A.5.7. Solving Problems with No Matching Rows 1296
A.5.8. Problems with Floating-Point Comparisons 1296
A.6. Optimizer-Related Issues 1298
A.7. Table Definition-Related Issues 1299
A.7.1. Problems with ALTER TABLE 1299
A.7.2. How to Change the Order of Columns in a Table 1300
A.7.3. TEMPORARY TABLE Problems 1300
B. Credits 1302
B.1. Developers at MySQL AB 1302
B.2. Contributors to MySQL 1307
B.3. Documenters and translators 1311
B.4. Libraries used by and included with MySQL 1313
B.5. Packages that support MySQL 1314
B.6. Tools that were used to create MySQL 1314
B.7. Supporters of MySQL 1315
C. MySQL and the Future (the TODO) 1316
C.1. New Features Planned for 5.0 1316
C.2. New Features Planned for 5.1 1317
C.3. New Features Planned for the Near Future 1318
C.4. New Features Planned for the Mid-Term Future 1320
C.5. New Features We Don't Plan to Implement 1322
D. MySQL Change History 1323
D.1. Changes in release 5.0.x (Development) 1323
D.1.1. Changes in release 5.0.4 (not released yet) 1324
D.1.2. Changes in release 5.0.3 (release imminent) 1324
D.1.3. Changes in release 5.0.2 (01 Dec 2004) 1334
D.1.4. Changes in release 5.0.1 (27 Jul 2004) 1337
D.1.5. Changes in release 5.0.0 (22 Dec 2003: Alpha) 1341
D.2. Changes in release 4.1.x (Production) 1342
D.2.1. Changes in release 4.1.11 (not released yet) 1342
D.2.2. Changes in release 4.1.10 (12 Feb 2005) 1347
D.2.3. Changes in release 4.1.9 (11 Jan 2005) 1352
D.2.4. Changes in release 4.1.8 (14 Dec 2004) 1354

D.2.5. Changes in release 4.1.7 (23 Oct 2004: Production) 1358
D.2.6. Changes in release 4.1.6 (10 Oct 2004) 1360
D.2.7. Changes in release 4.1.5 (16 Sep 2004) 1362
D.2.8. Changes in release 4.1.4 (26 Aug 2004: Gamma) 1363
D.2.9. Changes in release 4.1.3 (28 Jun 2004: Beta) 1365
D.2.10. Changes in release 4.1.2 (28 May 2004) 1368
D.2.11. Changes in release 4.1.1 (01 Dec 2003) 1378
D.2.12. Changes in release 4.1.0 (03 Apr 2003: Alpha) 1383
D.3. Changes in release 4.0.x (Production) 1386
D.3.1. Changes in release 4.0.25 (not released yet) 1387
D.3.2. Changes in release 4.0.24 (04 Mar 2005) 1387
D.3.3. Changes in release 4.0.23 (18 Dec 2004) 1390
D.3.4. Changes in release 4.0.22 (27 Oct 2004) 1391
D.3.5. Changes in release 4.0.21 (06 Sep 2004) 1393
D.3.6. Changes in release 4.0.20 (17 May 2004) 1396
D.3.7. Changes in release 4.0.19 (04 May 2004) 1396
D.3.8. Changes in release 4.0.18 (12 Feb 2004) 1400
D.3.9. Changes in release 4.0.17 (14 Dec 2003) 1403
D.3.10. Changes in release 4.0.16 (17 Oct 2003) 1406
D.3.11. Changes in release 4.0.15 (03 Sep 2003) 1408
D.3.12. Changes in release 4.0.14 (18 Jul 2003) 1412
D.3.13. Changes in release 4.0.13 (16 May 2003) 1416
D.3.14. Changes in release 4.0.12 (15 Mar 2003: Production) 1420
D.3.15. Changes in release 4.0.11 (20 Feb 2003) 1422
D.3.16. Changes in release 4.0.10 (29 Jan 2003) 1423
D.3.17. Changes in release 4.0.9 (09 Jan 2003) 1424
D.3.18. Changes in release 4.0.8 (07 Jan 2003) 1425
D.3.19. Changes in release 4.0.7 (20 Dec 2002) 1426
D.3.20. Changes in release 4.0.6 (14 Dec 2002: Gamma) 1426
D.3.21. Changes in release 4.0.5 (13 Nov 2002) 1428
D.3.22. Changes in release 4.0.4 (29 Sep 2002) 1430
D.3.23. Changes in release 4.0.3 (26 Aug 2002: Beta) 1432
D.3.24. Changes in release 4.0.2 (01 Jul 2002) 1434
D.3.25. Changes in release 4.0.1 (23 Dec 2001) 1438
D.3.26. Changes in release 4.0.0 (Oct 2001: Alpha) 1439
D.4. Changes in release 3.23.x (Recent; still supported) 1441
D.4.1. Changes in release 3.23.59 (not released yet) 1441
D.4.2. Changes in release 3.23.58 (11 Sep 2003) 1442
D.4.3. Changes in release 3.23.57 (06 Jun 2003) 1443
D.4.4. Changes in release 3.23.56 (13 Mar 2003) 1444
D.4.5. Changes in release 3.23.55 (23 Jan 2003) 1445
D.4.6. Changes in release 3.23.54 (05 Dec 2002) 1445
D.4.7. Changes in release 3.23.53 (09 Oct 2002) 1446
D.4.8. Changes in release 3.23.52 (14 Aug 2002) 1447
D.4.9. Changes in release 3.23.51 (31 May 2002) 1448
D.4.10. Changes in release 3.23.50 (21 Apr 2002) 1449
D.4.11. Changes in release 3.23.49 (14 Feb 2002) 1450
D.4.12. Changes in release 3.23.48 (07 Feb 2002) 1450
D.4.13. Changes in release 3.23.47 (27 Dec 2001) 1451
D.4.14. Changes in release 3.23.46 (29 Nov 2001) 1452
D.4.15. Changes in release 3.23.45 (22 Nov 2001) 1452
D.4.16. Changes in release 3.23.44 (31 Oct 2001) 1453
D.4.17. Changes in release 3.23.43 (04 Oct 2001) 1454
D.4.18. Changes in release 3.23.42 (08 Sep 2001) 1454
D.4.19. Changes in release 3.23.41 (11 Aug 2001) 1455
D.4.20. Changes in release 3.23.40 (18 Jul 2001) 1456
D.4.21. Changes in release 3.23.39 (12 Jun 2001) 1457
D.4.22. Changes in release 3.23.38 (09 May 2001) 1457

D.4.23. Changes in release 3.23.37 (17 Apr 2001) 1458
D.4.24. Changes in release 3.23.36 (27 Mar 2001) 1459
D.4.25. Changes in release 3.23.35 (15 Mar 2001) 1460
D.4.26. Changes in release 3.23.34a (11 Mar 2001) 1460
D.4.27. Changes in release 3.23.34 (10 Mar 2001) 1460
D.4.28. Changes in release 3.23.33 (09 Feb 2001) 1461
D.4.29. Changes in release 3.23.32 (22 Jan 2001) 1462
D.4.30. Changes in release 3.23.31 (17 Jan 2001: Production) 1463
D.4.31. Changes in release 3.23.30 (04 Jan 2001) 1464
D.4.32. Changes in release 3.23.29 (16 Dec 2000) 1465
D.4.33. Changes in release 3.23.28 (22 Nov 2000: Gamma) 1467
D.4.34. Changes in release 3.23.27 (24 Oct 2000) 1468
D.4.35. Changes in release 3.23.26 (18 Oct 2000) 1469
D.4.36. Changes in release 3.23.25 (29 Sep 2000) 1470
D.4.37. Changes in release 3.23.24 (08 Sep 2000) 1471
D.4.38. Changes in release 3.23.23 (01 Sep 2000) 1471
D.4.39. Changes in release 3.23.22 (31 Jul 2000) 1473
D.4.40. Changes in release 3.23.21 (04 Jul 2000) 1473
D.4.41. Changes in release 3.23.20 (28 Jun 2000: Beta) 1474
D.4.42. Changes in release 3.23.19 1474
D.4.43. Changes in release 3.23.18 (11 Jun 2000) 1475
D.4.44. Changes in release 3.23.17 (07 Jun 2000) 1475
D.4.45. Changes in release 3.23.16 (16 May 2000) 1476
D.4.46. Changes in release 3.23.15 (08 May 2000) 1476
D.4.47. Changes in release 3.23.14 (09 Apr 2000) 1478
D.4.48. Changes in release 3.23.13 (14 Mar 2000) 1478
D.4.49. Changes in release 3.23.12 (07 Mar 2000) 1479
D.4.50. Changes in release 3.23.11 (16 Feb 2000) 1479
D.4.51. Changes in release 3.23.10 (30 Jan 2000) 1480
D.4.52. Changes in release 3.23.9 (29 Jan 2000) 1480
D.4.53. Changes in release 3.23.8 (02 Jan 2000) 1481
D.4.54. Changes in release 3.23.7 (10 Dec 1999) 1482
D.4.55. Changes in release 3.23.6 (15 Nov 1999) 1483
D.4.56. Changes in release 3.23.5 (20 Oct 1999) 1483
D.4.57. Changes in release 3.23.4 (28 Sep 1999) 1484
D.4.58. Changes in release 3.23.3 (13 Sep 1999) 1485
D.4.59. Changes in release 3.23.2 (09 Aug 1999) 1486
D.4.60. Changes in release 3.23.1 (08 Jul 1999) 1487
D.4.61. Changes in release 3.23.0 (05 Jul 1999: Alpha) 1487
D.5. Changes in release 3.22.x (Old; discontinued) 1489
D.5.1. Changes in release 3.22.35 1489
D.5.2. Changes in release 3.22.34 1489
D.5.3. Changes in release 3.22.33 1489
D.5.4. Changes in release 3.22.32 (14 Feb 2000) 1490
D.5.5. Changes in release 3.22.31 1490
D.5.6. Changes in release 3.22.30 (11 Jan 2000) 1490
D.5.7. Changes in release 3.22.29 (02 Jan 2000) 1490
D.5.8. Changes in release 3.22.28 (20 Oct 1999) 1491
D.5.9. Changes in release 3.22.27 (05 Oct 1999) 1491
D.5.10. Changes in release 3.22.26 (16 Sep 1999) 1491
D.5.11. Changes in release 3.22.25 (07 Jun 1999) 1491
D.5.12. Changes in release 3.22.24 (05 Jul 1999) 1492
D.5.13. Changes in release 3.22.23 (08 Jun 1999) 1492
D.5.14. Changes in release 3.22.22 (30 Apr 1999) 1492
D.5.15. Changes in release 3.22.21 (04 Apr 1999) 1493
D.5.16. Changes in release 3.22.20 (18 Mar 1999) 1493
D.5.17. Changes in release 3.22.19 (01 Mar 1999) 1493
D.5.18. Changes in release 3.22.18 (26 Feb 1999) 1493

D.5.19. Changes in release 3.22.17 (22 Feb 1999: Production) 1493
D.5.20. Changes in release 3.22.16 (05 Feb 1999) 1494
D.5.21. Changes in release 3.22.15 (27 Jan 1999) 1494
D.5.22. Changes in release 3.22.14 (01 Jan 1999: Gamma) 1494
D.5.23. Changes in release 3.22.13 (16 Dec 1998) 1495
D.5.24. Changes in release 3.22.12 (09 Dec 1998) 1495
D.5.25. Changes in release 3.22.11 (24 Nov 1998) 1496
D.5.26. Changes in release 3.22.10 (04 Nov 1998) 1496
D.5.27. Changes in release 3.22.9 (19 Oct 1998) 1497
D.5.28. Changes in release 3.22.8 (06 Oct 1998) 1498
D.5.29. Changes in release 3.22.7 (21 Sep 1998: Beta) 1498
D.5.30. Changes in release 3.22.6 (31 Aug 1998) 1499
D.5.31. Changes in release 3.22.5 (20 Aug 1998: Alpha) 1499
D.5.32. Changes in release 3.22.4 (06 Jul 1998: Beta) 1500
D.5.33. Changes in release 3.22.3 (30 Jun 1998) 1501
D.5.34. Changes in release 3.22.2 1501
D.5.35. Changes in release 3.22.1 (Jun 1998) 1502
D.5.36. Changes in release 3.22.0 (18 May 1998: Alpha) 1503
D.6. Changes in release 3.21.x 1504
D.6.1. Changes in release 3.21.33 (08 Jul 1998) 1504
D.6.2. Changes in release 3.21.32 (30 Jun 1998) 1504
D.6.3. Changes in release 3.21.31 (10 Jun 1998) 1505
D.6.4. Changes in release 3.21.30 1505
D.6.5. Changes in release 3.21.29 1506
D.6.6. Changes in release 3.21.28 1506
D.6.7. Changes in release 3.21.27 1506
D.6.8. Changes in release 3.21.26 1507
D.6.9. Changes in release 3.21.25 1507
D.6.10. Changes in release 3.21.24 1507
D.6.11. Changes in release 3.21.23 1508
D.6.12. Changes in release 3.21.22 1508
D.6.13. Changes in release 3.21.21a 1509
D.6.14. Changes in release 3.21.21 1509
D.6.15. Changes in release 3.21.20 1509
D.6.16. Changes in release 3.21.19 1510
D.6.17. Changes in release 3.21.18 1510
D.6.18. Changes in release 3.21.17 1510
D.6.19. Changes in release 3.21.16 1511
D.6.20. Changes in release 3.21.15 1511
D.6.21. Changes in release 3.21.14b 1512
D.6.22. Changes in release 3.21.14a 1512
D.6.23. Changes in release 3.21.13 1513
D.6.24. Changes in release 3.21.12 1513
D.6.25. Changes in release 3.21.11 1514
D.6.26. Changes in release 3.21.10 1515
D.6.27. Changes in release 3.21.9 1515
D.6.28. Changes in release 3.21.8 1515
D.6.29. Changes in release 3.21.7 1516
D.6.30. Changes in release 3.21.6 1516
D.6.31. Changes in release 3.21.5 1516
D.6.32. Changes in release 3.21.4 1517
D.6.33. Changes in release 3.21.3 1517
D.6.34. Changes in release 3.21.2 1518
D.6.35. Changes in release 3.21.0 1519
D.7. Changes in release 3.20.x 1520
D.7.1. Changes in release 3.20.18 1520
D.7.2. Changes in release 3.20.17 1520
D.7.3. Changes in release 3.20.16 1521

D.7.4. Changes in release 3.20.15 1522
D.7.5. Changes in release 3.20.14 1522
D.7.6. Changes in release 3.20.13 1523
D.7.7. Changes in release 3.20.11 1523
D.7.8. Changes in release 3.20.10 1524
D.7.9. Changes in release 3.20.9 1524
D.7.10. Changes in release 3.20.8 1524
D.7.11. Changes in release 3.20.7 1524
D.7.12. Changes in release 3.20.6 1525
D.7.13. Changes in release 3.20.3 1526
D.7.14. Changes in release 3.20.0 1527
D.8. Changes in release 3.19.x 1527
D.8.1. Changes in release 3.19.5 1528
D.8.2. Changes in release 3.19.4 1528
D.8.3. Changes in release 3.19.3 1528
D.9. `InnoDB` Change History 1529
D.9.1. MySQL/InnoDB-4.0.21, September 10, 2004 1529
D.9.2. MySQL/InnoDB-4.1.4, August 31, 2004 1530
D.9.3. MySQL/InnoDB-4.1.3, June 28, 2004 1531
D.9.4. MySQL/InnoDB-4.1.2, May 30, 2004 1532
D.9.5. MySQL/InnoDB-4.0.20, May 18, 2004 1533
D.9.6. MySQL/InnoDB-4.0.19, May 4, 2004 1533
D.9.7. MySQL/InnoDB-4.0.18, February 13, 2004 1534
D.9.8. MySQL/InnoDB-5.0.0, December 24, 2003 1535
D.9.9. MySQL/InnoDB-4.0.17, December 17, 2003 1535
D.9.10. MySQL/InnoDB-4.1.1, December 4, 2003 1536
D.9.11. MySQL/InnoDB-4.0.16, October 22, 2003 1536
D.9.12. MySQL/InnoDB-3.23.58, September 15, 2003 1536
D.9.13. MySQL/InnoDB-4.0.15, September 10, 2003 1537
D.9.14. MySQL/InnoDB-4.0.14, July 22, 2003 1537
D.9.15. MySQL/InnoDB-3.23.57, June 20, 2003 1538
D.9.16. MySQL/InnoDB-4.0.13, May 20, 2003 1539
D.9.17. MySQL/InnoDB-4.1.0, April 3, 2003 1540
D.9.18. MySQL/InnoDB-3.23.56, March 17, 2003 1540
D.9.19. MySQL/InnoDB-4.0.12, March 18, 2003 1540
D.9.20. MySQL/InnoDB-4.0.11, February 25, 2003 1540
D.9.21. MySQL/InnoDB-4.0.10, February 4, 2003 1541
D.9.22. MySQL/InnoDB-3.23.55, January 24, 2003 1541
D.9.23. MySQL/InnoDB-4.0.9, January 14, 2003 1542
D.9.24. MySQL/InnoDB-4.0.8, January 7, 2003 1542
D.9.25. MySQL/InnoDB-4.0.7, December 26, 2002 1542
D.9.26. MySQL/InnoDB-4.0.6, December 19, 2002 1543
D.9.27. MySQL/InnoDB-3.23.54, December 12, 2002 1543
D.9.28. MySQL/InnoDB-4.0.5, November 18, 2002 1543
D.9.29. MySQL/InnoDB-3.23.53, October 9, 2002 1544
D.9.30. MySQL/InnoDB-4.0.4, October 2, 2002 1545
D.9.31. MySQL/InnoDB-4.0.3, August 28, 2002 1546
D.9.32. MySQL/InnoDB-3.23.52, August 16, 2002 1546
D.9.33. MySQL/InnoDB-4.0.2, July 10, 2002 1548
D.9.34. MySQL/InnoDB-3.23.51, June 12, 2002 1548
D.9.35. MySQL/InnoDB-3.23.50, April 23, 2002 1548
D.9.36. MySQL/InnoDB-3.23.49, February 17, 2002 1549
D.9.37. MySQL/InnoDB-3.23.48, February 9, 2002 1549
D.9.38. MySQL/InnoDB-3.23.47, December 28, 2001 1550
D.9.39. MySQL/InnoDB-4.0.1, December 23, 2001 1550
D.9.40. MySQL/InnoDB-3.23.46, November 30, 2001 1551
D.9.41. MySQL/InnoDB-3.23.45, November 23, 2001 1551
D.9.42. MySQL/InnoDB-3.23.44, November 2, 2001 1551

D.9.43. MySQL/InnoDB-3.23.43, October 4, 2001 1552
D.9.44. MySQL/InnoDB-3.23.42, September 9, 2001 1552
D.9.45. MySQL/InnoDB-3.23.41, August 13, 2001 1552
D.9.46. MySQL/InnoDB-3.23.40, July 16, 2001 1553
D.9.47. MySQL/InnoDB-3.23.39, June 13, 2001 1553
D.9.48. MySQL/InnoDB-3.23.38, May 12, 2001 1553
D.10. `MySQL Cluster` Change History 1553
D.10.1. MySQL Cluster-4.1.10 (12 Feb 2005) 1553
D.10.2. MySQL Cluster-4.1.9 (13 Jan 2005) 1554
D.10.3. MySQL Cluster-4.1.8 (14 Dec 2004) 1555
D.10.4. MySQL Cluster-4.1.7 (23 Oct 2004) 1557
D.10.5. MySQL Cluster-4.1.6 (10 Oct 2004) 1557
D.10.6. MySQL Cluster-4.1.5 (16 Sep 2004) 1558
D.10.7. MySQL Cluster-4.1.4 (31 Aug 2004) 1560
D.10.8. MySQL Cluster-5.0.1 (27 Jul 2004) 1560
D.10.9. MySQL Cluster-4.1.3 (28 Jun 2004) 1561
D.11. Changes in MyODBC 1561
D.11.1. Changes in MyODBC 3.51.11 1561
E. Porting to Other Systems 1562
E.1. Debugging a MySQL Server 1563
E.1.1. Compiling MySQL for Debugging 1563
E.1.2. Creating Trace Files 1564
E.1.3. Debugging **mysqld** under **gdb** 1564
E.1.4. Using a Stack Trace 1566
E.1.5. Using Log Files to Find Cause of Errors in **mysqld** 1567
E.1.6. Making a Test Case If You Experience Table Corruption 1567
E.2. Debugging a MySQL Client 1568
E.3. The DBUG Package 1569
E.4. Comments about RTS Threads 1570
E.5. Differences Between Thread Packages 1571
F. Environment Variables 1573
G. MySQL Regular Expressions 1574
H. GNU General Public License 1578
I. MySQL FLOSS License Exception 1583
Index 1585

Preface

This is the Reference Manual for the MySQL Database System. It documents MySQL up to Version 5.0.3-alpha, but is also applicable for older versions of the MySQL software (such as 3.23 or 4.0-production) because functional changes are indicated with reference to a version number.

Chapter 1. General Information

The MySQL (R) software delivers a very fast, multi-threaded, multi-user, and robust SQL (Structured Query Language) database server. MySQL Server is intended for mission-critical, heavy-load production systems as well as for embedding into mass-deployed software. MySQL is a registered trademark of MySQL AB.

The MySQL software is Dual Licensed. Users can choose to use the MySQL software as an Open Source product under the terms of the GNU General Public License (http://www.fsf.org/licenses/) or can purchase a standard commercial license from MySQL AB. See http://www.mysql.com/company/legal/licensing/ for more information on our licensing policies.

The following list describes some sections of particular interest in this manual:

- For a discussion about the capabilities of the MySQL Database Server, see Section 1.2.2, “The Main Features of MySQL”.
- For installation instructions, see Chapter 2, *Installing MySQL*.
- For tips on porting the MySQL Database Software to new architectures or operating systems, see Appendix E, *Porting to Other Systems*.
- For information about upgrading from a Version 4.0 release, see Section 2.10.2, “Upgrading from Version 4.0 to 4.1”.
- For information about upgrading from a Version 3.23 release, see Section 2.10.3, “Upgrading from Version 3.23 to 4.0”.
- For information about upgrading from a Version 3.22 release, see Section 2.10.4, “Upgrading from Version 3.22 to 3.23”.
- For a tutorial introduction to the MySQL Database Server, see Chapter 3, *MySQL Tutorial*.
- For examples of SQL and benchmarking information, see the benchmarking directory (`sql-bench` in the distribution).
- For a history of new features and bugfixes, see Appendix D, *MySQL Change History*.
- For a list of currently known bugs and misfeatures, see Section 1.5.7, “Known Errors and Design Deficiencies in MySQL”.
- For future plans, see Appendix C, *MySQL and the Future (the TODO)*.
- For a list of all the contributors to this project, see Appendix B, *Credits*.

Important:

Reports of errors (often called ``bugs''), as well as questions and comments, should be sent to http://bugs.mysql.com. See Section 1.4.1.3, “How to Report Bugs or Problems”.

If you have found a sensitive security bug in MySQL Server, please let us know immediately by sending an email message to `<security@mysql.com>`.

1.1. About This Manual

This is the Reference Manual for the MySQL Database System. It documents MySQL up to Version 5.0.3-alpha, but is also applicable for older versions of the MySQL software (such as 3.23 or 4.0-production) because functional changes are indicated with reference to a version number.

Because this manual serves as a reference, it does not provide general instruction on SQL or relational database concepts. It also does not teach you how to use your operating system or command-line interpreter.

The MySQL Database Software is under constant development, and the Reference Manual is updated frequently as well. The most recent version of the manual is available online in searchable form at http://dev.mysql.com/doc/. Other formats also are available, including HTML, PDF, and Windows CHM versions.

The primary document is the Texinfo file. The HTML version is produced automatically using a modified version of **texi2html**. The plain text and Info versions are produced with **makeinfo**. The PostScript version is produced using **texi2dvi** and **dvips**. The PDF version is produced with **pdftex**.

If you have any suggestions concerning additions or corrections to this manual, please send them to the documentation team at `<docs@mysql.com>`.

This manual was initially written by David Axmark and Michael ``Monty'' Widenius. It is maintained by the MySQL Documentation Team, consisting of Paul DuBois, Stefan Hinz, Mike Hillyer, Jon Stephens, and Russell Dyer. For the many other contributors, see Appendix B, *Credits*.

The copyright (2004) to this manual is owned by the Swedish company MySQL AB. MySQL and the MySQL logo are (registered) trademarks of MySQL AB. Other trademarks and registered trademarks referred to in this manual are the property of their respective owners, and are used for identification purposes only.

1.1.1. Conventions Used in This Manual

This manual uses certain typographical conventions:

- `constant`

  Constant-width font is used for command names and options; SQL statements; database, table, and column names; C and Perl code; and environment variables. Example: ``To see how **mysqladmin** works, invoke it with the `--help` option.''

- *`constant italic`*

  Italic constant-width font is used to indicate variable input for which you should substitute a value of your own choosing.

- `filename`

  Constant-width font with surrounding quotes is used for filenames and pathnames. Example: ``The distribution is installed under the `/usr/local/` directory.''

- '`c`'

  Constant-width font with surrounding quotes is also used to indicate character sequences. Example: ``To specify a wildcard, use the '`%`' character.''

- *italic*

Italic font is used for emphasis, *like this*.

- **boldface**

  Boldface font is used in table headings and to convey **especially strong emphasis**.

When commands are shown that are meant to be executed from within a particular program, the program is indicated by a prompt shown before the command. For example, `shell>` indicates a command that you execute from your login shell, and `mysql>` indicates a statement that you execute from the **mysql** client program:

```
shell> type a shell command here
mysql> type a mysql statement here
```

The ``shell'' is your command interpreter. On Unix, this is typically a program such as **sh** or **csh**. On Windows, the equivalent program is **command.com** or **cmd.exe**, typically run in a console window.

When you enter a command or statement shown in an example, do not type the prompt shown in the example.

Database, table, and column names must often be substituted into statements. To indicate that such substitution is necessary, this manual uses *`db_name`*, *`tbl_name`*, and *`col_name`*. For example, you might see a statement like this:

```
mysql> SELECT col_name FROM db_name.tbl_name;
```

This means that if you were to enter a similar statement, you would supply your own database, table, and column names, perhaps like this:

```
mysql> SELECT author_name FROM biblio_db.author_list;
```

SQL keywords are not case sensitive and may be written in uppercase or lowercase. This manual uses uppercase.

In syntax descriptions, square brackets ('`[`' and '`]`') are used to indicate optional words or clauses. For example, in the following statement, `IF EXISTS` is optional:

```
DROP TABLE [IF EXISTS] tbl_name
```

When a syntax element consists of a number of alternatives, the alternatives are separated by vertical bars ('`|`'). When one member from a set of choices *may* be chosen, the alternatives are listed within square brackets ('`[`' and '`]`'):

```
TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str)
```

When one member from a set of choices *must* be chosen, the alternatives are listed within braces ('`{`' and '`}`'):

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

An ellipsis (`...`) indicates the omission of a section of a statement, typically to provide a shorter version of more complex syntax. For example, `INSERT ... SELECT` is shorthand for the form of `IN-`

SERT statement that is followed by a SELECT statement.

An ellipsis can also indicate that the preceding syntax element of a statement may be repeated. In the following example, multiple *reset_option* values may be given, with each of those after the first preceded by commas:

```
RESET reset_option [,reset_option] ...
```

Commands for setting shell variables are shown using Bourne shell syntax. For example, the sequence to set an environment variable and run a command looks like this in Bourne shell syntax:

```
shell> VARNAME=value some_command
```

If you are using **csh** or **tcsh**, you must issue commands somewhat differently. You would execute the sequence just shown like this:

```
shell> setenv VARNAME value
shell> some_command
```

1.2. Overview of the MySQL Database Management System

MySQL, the most popular Open Source SQL database management system, is developed, distributed, and supported by MySQL AB. MySQL AB is a commercial company, founded by the MySQL developers. It is a second generation Open Source company that unites Open Source values and methodology with a successful business model.

The MySQL Web site (http://www.mysql.com/) provides the latest information about MySQL software and MySQL AB.

- MySQL is a database management system.

  A database is a structured collection of data. It may be anything from a simple shopping list to a picture gallery or the vast amounts of information in a corporate network. To add, access, and process data stored in a computer database, you need a database management system such as MySQL Server. Since computers are very good at handling large amounts of data, database management systems play a central role in computing, as standalone utilities or as parts of other applications.

- MySQL is a relational database management system.

  A relational database stores data in separate tables rather than putting all the data in one big storeroom. This adds speed and flexibility. The SQL part of ``MySQL'' stands for ``Structured Query Language.'' SQL is the most common standardized language used to access databases and is defined by the ANSI/ISO SQL Standard. The SQL standard has been evolving since 1986 and several versions exist. In this manual, ``SQL-92'' refers to the standard released in 1992, ``SQL:1999'' refers to the standard released in 1999, and ``SQL:2003'' refers to the current version of the standard. We use the phrase ``the SQL standard'' to mean the current version of the SQL Standard at any time.

- MySQL software is Open Source.

  Open Source means that it is possible for anyone to use and modify the software. Anybody can download the MySQL software from the Internet and use it without paying anything. If you wish, you may study the source code and change it to suit your needs. The MySQL software uses the GPL (GNU General Public License), http://www.fsf.org/licenses/, to define what you may and may not do

with the software in different situations. If you feel uncomfortable with the GPL or need to embed MySQL code into a commercial application, you can buy a commercially licensed version from us. See the MySQL Licensing Overview for more information (http://www.mysql.com/company/legal/licensing/).

- The MySQL Database Server is very fast, reliable, and easy to use.

  If that is what you are looking for, you should give it a try. MySQL Server also has a practical set of features developed in close cooperation with our users. You can find a performance comparison of MySQL Server with other database managers on our benchmark page. See Section 7.1.4, “The MySQL Benchmark Suite”.

  MySQL Server was originally developed to handle large databases much faster than existing solutions and has been successfully used in highly demanding production environments for several years. Although under constant development, MySQL Server today offers a rich and useful set of functions. Its connectivity, speed, and security make MySQL Server highly suited for accessing databases on the Internet.

- MySQL Server works in client/server or embedded systems.

  The MySQL Database Software is a client/server system that consists of a multi-threaded SQL server that supports different backends, several different client programs and libraries, administrative tools, and a wide range of application programming interfaces (APIs).

  We also provide MySQL Server as an embedded multi-threaded library that you can link into your application to get a smaller, faster, easier-to-manage product.

- A large amount of contributed MySQL software is available.

  It is very likely that your favorite application or language supports the MySQL Database Server.

The official way to pronounce ``MySQL'' is ``My Ess Que Ell'' (not ``my sequel''), but we don't mind if you pronounce it as ``my sequel'' or in some other localized way.

1.2.1. History of MySQL

We started out with the intention of using `mSQL` to connect to our tables using our own fast low-level (ISAM) routines. However, after some testing, we came to the conclusion that `mSQL` was not fast enough or flexible enough for our needs. This resulted in a new SQL interface to our database but with almost the same API interface as `mSQL`. This API was designed to allow third-party code that was written for use with `mSQL` to be ported easily for use with MySQL.

The derivation of the name MySQL is not clear. Our base directory and a large number of our libraries and tools have had the prefix ``my'' for well over 10 years. However, co-founder Monty Widenius's daughter is also named My. Which of the two gave its name to MySQL is still a mystery, even for us.

The name of the MySQL Dolphin (our logo) is ``Sakila,'' which was chosen by the founders of MySQL AB from a huge list of names suggested by users in our ``Name the Dolphin'' contest. The winning name was submitted by Ambrose Twebaze, an Open Source software developer from Swaziland, Africa. According to Ambrose, the name Sakila has its roots in SiSwati, the local language of Swaziland. Sakila is also the name of a town in Arusha, Tanzania, near Ambrose's country of origin, Uganda.

1.2.2. The Main Features of MySQL

The following list describes some of the important characteristics of the MySQL Database Software. See also Section 1.3, “MySQL Development Roadmap” for more information about current and upcoming

features.

- Internals and Portability
  - Written in C and C++.
  - Tested with a broad range of different compilers.
  - Works on many different platforms. See Section 2.1.1, “Operating Systems Supported by MySQL”.
  - Uses GNU Automake, Autoconf, and Libtool for portability.
  - APIs for C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, and Tcl are available. See Chapter 22, *MySQL APIs*.
  - Fully multi-threaded using kernel threads. It can easily use multiple CPUs if they are available.
  - Provides transactional and non-transactional storage engines.
  - Uses very fast B-tree disk tables (`MyISAM`) with index compression.
  - Relatively easy to add another storage engine. This is useful if you want to add an SQL interface to an in-house database.
  - A very fast thread-based memory allocation system.
  - Very fast joins using an optimized one-sweep multi-join.
  - In-memory hash tables, which are used as temporary tables.
  - SQL functions are implemented using a highly optimized class library and should be as fast as possible. Usually there is no memory allocation at all after query initialization.
  - The MySQL code is tested with Purify (a commercial memory leakage detector) as well as with Valgrind, a GPL tool (http://developer.kde.org/~sewardj/).
  - The server is available as a separate program for use in a client/server networked environment. It is also available as a library that can be embedded (linked) into standalone applications. Such applications can be used in isolation or in environments where no network is available.
- Column Types
  - Many column types: signed/unsigned integers 1, 2, 3, 4, and 8 bytes long, `FLOAT`, `DOUBLE`, `CHAR`, `VARCHAR`, `TEXT`, `BLOB`, `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`, `YEAR`, `SET`, `ENUM`, and OpenGIS spatial types. See Chapter 11, *Column Types*.
  - Fixed-length and variable-length records.
- Statements and Functions
  - Full operator and function support in the `SELECT` and `WHERE` clauses of queries. For example:

    ```
    mysql> SELECT CONCAT(first_name, ' ', last_name)
        -> FROM citizen
        -> WHERE income/dependents > 10000 AND age > 30;
    ```

  - Full support for SQL `GROUP BY` and `ORDER BY` clauses. Support for group functions (`COUNT()`, `COUNT(DISTINCT ...)`, `AVG()`, `STD()`, `SUM()`, `MAX()`, `MIN()`, and

GROUP_CONCAT()).

- Support for LEFT OUTER JOIN and RIGHT OUTER JOIN with both standard SQL and ODBC syntax.
- Support for aliases on tables and columns as required by standard SQL.
- DELETE, INSERT, REPLACE, and UPDATE return the number of rows that were changed (affected). It is possible to return the number of rows matched instead by setting a flag when connecting to the server.
- The MySQL-specific SHOW command can be used to retrieve information about databases, tables, and indexes. The EXPLAIN command can be used to determine how the optimizer resolves a query.
- Function names do not clash with table or column names. For example, ABS is a valid column name. The only restriction is that for a function call, no spaces are allowed between the function name and the '(' that follows it. See Section 9.6, “Treatment of Reserved Words in MySQL”.
- You can mix tables from different databases in the same query (as of MySQL 3.22).

- Security
  - A privilege and password system that is very flexible and secure, and that allows host-based verification. Passwords are secure because all password traffic is encrypted when you connect to a server.
- Scalability and Limits
  - Handles large databases. We use MySQL Server with databases that contain 50 million records. We also know of users who use MySQL Server with 60,000 tables and about 5,000,000,000 rows.
  - Up to 64 indexes per table are allowed (32 before MySQL 4.1.2). Each index may consist of 1 to 16 columns or parts of columns. The maximum index width is 1000 bytes (500 before MySQL 4.1.2). An index may use a prefix of a column for CHAR, VARCHAR, BLOB, or TEXT column types.
- Connectivity
  - Clients can connect to the MySQL server using TCP/IP sockets on any platform. On Windows systems in the NT family (NT, 2000, XP, or 2003), clients can connect using named pipes. On Unix systems, clients can connect using Unix domain socket files.
  - In MySQL versions 4.1 and higher, Windows servers also support shared-memory connections if started with the --shared-memory option. Clients can connect through shared memory by using the --protocol=memory option.
  - The Connector/ODBC (MyODBC) interface provides MySQL support for client programs that use ODBC (Open Database Connectivity) connections. For example, you can use MS Access to connect to your MySQL server. Clients can be run on Windows or Unix. MyODBC source is available. All ODBC 2.5 functions are supported, as are many others. See Section 23.1, “MySQL ODBC Support”.
  - The Connector/J interface provides MySQL support for Java client programs that use JDBC connections. Clients can be run on Windows or Unix. Connector/J source is available. See Section 23.2, “MySQL Java Connectivity (JDBC)”.

- Localization
  - The server can provide error messages to clients in many languages. See Section 5.8.2, “Setting the Error Message Language”.
  - Full support for several different character sets, including `latin1` (ISO-8859-1), `german`, `big5`, `ujis`, and more. For example, the Scandinavian characters '`â`', '`ä`' and '`ö`' are allowed in table and column names. Unicode support is available as of MySQL 4.1.
  - All data is saved in the chosen character set. All comparisons for normal string columns are case-insensitive.
  - Sorting is done according to the chosen character set (using Swedish collation by default). It is possible to change this when the MySQL server is started. To see an example of very advanced sorting, look at the Czech sorting code. MySQL Server supports many different character sets that can be specified at compile time and runtime.
- Clients and Tools
  - The MySQL server has built-in support for SQL statements to check, optimize, and repair tables. These statements are available from the command line through the **mysqlcheck** client. MySQL also includes **myisamchk**, a very fast command-line utility for performing these operations on `MyISAM` tables. See Chapter 5, *Database Administration*.
  - All MySQL programs can be invoked with the `--help` or `-?` options to obtain online assistance.

1.2.3. MySQL Stability

This section addresses the questions, ``*How stable is MySQL Server?*'' and, ``*Can I depend on MySQL Server in this project?*'' We will try to clarify these issues and answer some important questions that concern many potential users. The information in this section is based on data gathered from the mailing lists, which are very active in identifying problems as well as reporting types of use.

The original code stems back to the early 1980s. It provides a stable code base, and the `ISAM` table format used by the original storage engine remains backward-compatible. At TcX, the predecessor of MySQL AB, MySQL code has worked in projects since mid-1996, without any problems. When the MySQL Database Software initially was released to a wider public, our new users quickly found some pieces of untested code. Each new release since then has had fewer portability problems, even though each new release has also had many new features.

Each release of the MySQL Server has been usable. Problems have occurred only when users try code from the ``gray zones.'' Naturally, new users don't know what the gray zones are; this section therefore attempts to document those areas that are currently known. The descriptions mostly deal with Version 3.23, 4.0 and 4.1 of MySQL Server. All known and reported bugs are fixed in the latest version, with the exception of those listed in the bugs section, which are design-related. See Section 1.5.7, “Known Errors and Design Deficiencies in MySQL”.

The MySQL Server design is multi-layered with independent modules. Some of the newer modules are listed here with an indication of how well-tested each of them is:

- Replication (Stable)

  Large groups of servers using replication are in production use, with good results. Work on enhanced replication features is continuing in MySQL 5.x.

- `InnoDB` tables (Stable)

  The `InnoDB` transactional storage engine has been declared stable in the MySQL 3.23 tree, starting from version 3.23.49. `InnoDB` is being used in large, heavy-load production systems.

- `BDB` tables (Stable)

  The `Berkeley DB` code is very stable, but we are still improving the `BDB` transactional storage engine interface in MySQL Server.

- Full-text searches (Stable)

  Full-text searching is widely used. Important feature enhancements were added in MySQL 4.0 and 4.1.

- `MyODBC` 3.51 (Stable)

  `MyODBC` 3.51 uses ODBC SDK 3.51 and is in wide production use. Some issues brought up appear to be application-related and independent of the ODBC driver or underlying database server.

1.2.4. How Big MySQL Tables Can Be

MySQL 3.22 had a 4GB (4 gigabyte) limit on table size. With the `MyISAM` storage engine in MySQL 3.23, the maximum table size was increased to 8 million terabytes (2 ^ 63 bytes). With this larger allowed table size, the maximum effective table size for MySQL databases is usually determined by operating system constraints on file sizes, not by MySQL internal limits.

The `InnoDB` storage engine maintains `InnoDB` tables within a tablespace that can be created from several files. This allows a table to exceed the maximum individual file size. The tablespace can include raw disk partitions, which allows extremely large tables. The maximum tablespace size is 64TB.

The following table lists some examples of operating system file-size limits. This is only a rough guide and is not intended to be definitive. For the most up-to-date information, be sure to check the documentation specific to your operating system.

Operating System	**File-size Limit**
Linux 2.2-Intel 32-bit	2GB (LFS: 4GB)
Linux 2.4	(using ext3 filesystem) 4TB
Solaris 9/10	16TB
NetWare w/NSS filesystem	8TB
win32 w/ FAT/FAT32	2GB/4GB
win32 w/ NTFS	2TB (possibly larger)
MacOS X w/ HFS+	2TB

On Linux 2.2, you can get `MyISAM` tables larger than 2GB in size by using the Large File Support (LFS) patch for the ext2 filesystem. On Linux 2.4, patches also exist for ReiserFS to get support for big files (up to 2TB). Most current Linux distributions are based on kernel 2.4 and include all the required LFS patches. With JFS and XFS, petabyte and larger files are possible on Linux. However, the maximum available file size still depends on several factors, one of them being the filesystem used to store MySQL tables.

For a detailed overview about LFS in Linux, have a look at Andreas Jaeger's *Large File Support in Linux* page at http://www.suse.de/~aj/linux_lfs.html.

Windows users please note: FAT and VFAT (FAT32) are **not** considered suitable for production use with MySQL. Use NTFS instead.

By default, MySQL creates `MyISAM` tables with an internal structure that allows a maximum size of about 4GB. You can check the maximum table size for a table with the `SHOW TABLE STATUS` statement or with **myisamchk -dv** ***`tbl_name`***. See Section 13.5.4, "`SHOW` Syntax".

If you need a `MyISAM` table that is larger than 4GB in size (and your operating system supports large files), the `CREATE TABLE` statement allows `AVG_ROW_LENGTH` and `MAX_ROWS` options. See Section 13.2.6, "`CREATE TABLE` Syntax". You can also change these options with `ALTER TABLE` after the table has been created, to increase the table's maximum allowable size. See Section 13.2.2, "`ALTER TABLE` Syntax".

Other ways to work around file-size limits for `MyISAM` tables are as follows:

- If your large table is read-only, you can use **myisampack** to compress it. **myisampack** usually compresses a table by at least 50%, so you can have, in effect, much bigger tables. **myisampack** also can merge multiple tables into a single table. See Section 8.2, "**myisampack**, the MySQL Compressed Read-only Table Generator".
- MySQL includes a `MERGE` library that allows you to handle a collection of `MyISAM` tables that have identical structure as a single `MERGE` table. See Section 14.2, "The `MERGE` Storage Engine".

1.2.5. Year 2000 Compliance

The MySQL Server itself has no problems with Year 2000 (Y2K) compliance:

- MySQL Server uses Unix time functions that handle dates into the year `2037` for `TIMESTAMP` values. For `DATE` and `DATETIME` values, dates through the year `9999` are accepted.
- All MySQL date functions are implemented in one source file, `sql/time.cc`, and are coded very carefully to be year 2000-safe.
- In MySQL 3.22 and later, the `YEAR` column type can store years `0` and `1901` to `2155` in one byte and display them using two or four digits. All two-digit years are considered to be in the range `1970` to `2069`, which means that if you store `01` in a `YEAR` column, MySQL Server treats it as `2001`.

The following simple demonstration illustrates that MySQL Server has no problems with `DATE` or `DATETIME` values through the year 9999, and no problems with `TIMESTAMP` values until after the year 2030:

```
mysql> DROP TABLE IF EXISTS y2k;
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE y2k (date DATE,
    ->                   date_time DATETIME,
    ->                   time_stamp TIMESTAMP);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO y2k VALUES
    -> ('1998-12-31','1998-12-31 23:59:59',19981231235959),
    -> ('1999-01-01','1999-01-01 00:00:00',19990101000000),
    -> ('1999-09-09','1999-09-09 23:59:59',19990909235959),
    -> ('2000-01-01','2000-01-01 00:00:00',20000101000000),
    -> ('2000-02-28','2000-02-28 00:00:00',20000228000000),
    -> ('2000-02-29','2000-02-29 00:00:00',20000229000000),
```

```
    -> ('2000-03-01','2000-03-01 00:00:00',20000301000000),
    -> ('2000-12-31','2000-12-31 23:59:59',20001231235959),
    -> ('2001-01-01','2001-01-01 00:00:00',20010101000000),
    -> ('2004-12-31','2004-12-31 23:59:59',20041231235959),
    -> ('2005-01-01','2005-01-01 00:00:00',20050101000000),
    -> ('2030-01-01','2030-01-01 00:00:00',20300101000000),
    -> ('2040-01-01','2040-01-01 00:00:00',20400101000000),
    -> ('9999-12-31','9999-12-31 23:59:59',99991231235959);
Query OK, 14 rows affected (0.01 sec)
Records: 14  Duplicates: 0  Warnings: 2

mysql> SELECT * FROM y2k;
+------------+---------------------+----------------+
| date       | date_time           | time_stamp     |
+------------+---------------------+----------------+
| 1998-12-31 | 1998-12-31 23:59:59 | 19981231235959 |
| 1999-01-01 | 1999-01-01 00:00:00 | 19990101000000 |
| 1999-09-09 | 1999-09-09 23:59:59 | 19990909235959 |
| 2000-01-01 | 2000-01-01 00:00:00 | 20000101000000 |
| 2000-02-28 | 2000-02-28 00:00:00 | 20000228000000 |
| 2000-02-29 | 2000-02-29 00:00:00 | 20000229000000 |
| 2000-03-01 | 2000-03-01 00:00:00 | 20000301000000 |
| 2000-12-31 | 2000-12-31 23:59:59 | 20001231235959 |
| 2001-01-01 | 2001-01-01 00:00:00 | 20010101000000 |
| 2004-12-31 | 2004-12-31 23:59:59 | 20041231235959 |
| 2005-01-01 | 2005-01-01 00:00:00 | 20050101000000 |
| 2030-01-01 | 2030-01-01 00:00:00 | 20300101000000 |
| 2040-01-01 | 2040-01-01 00:00:00 | 00000000000000 |
| 9999-12-31 | 9999-12-31 23:59:59 | 00000000000000 |
+------------+---------------------+----------------+
14 rows in set (0.00 sec)
```

The final two `TIMESTAMP` column values are zero because the year values (`2040`, `9999`) exceed the `TIMESTAMP` maximum. The `TIMESTAMP` data type, which is used to store the current time, supports values that range from `19700101000000` to `20300101000000` on 32-bit machines (signed value). On 64-bit machines, `TIMESTAMP` handles values up to `2106` (unsigned value).

Although MySQL Server itself is Y2K-safe, you may run into problems if you use it with applications that are not Y2K-safe. For example, many old applications store or manipulate years using two-digit values (which are ambiguous) rather than four-digit values. This problem may be compounded by applications that use values such as `00` or `99` as ``missing'' value indicators. Unfortunately, these problems may be difficult to fix because different applications may be written by different programmers, each of whom may use a different set of conventions and date-handling functions.

Thus, even though MySQL Server has no Y2K problems, it is the application's responsibility to provide unambiguous input. See Section 11.3.4, “Y2K Issues and Date Types” for MySQL Server's rules for dealing with ambiguous date input data that contains two-digit year values.

1.3. MySQL Development Roadmap

This section provides a snapshot of the MySQL development roadmap, including major features implemented or planned for MySQL 4.0, 4.1, 5.0, and 5.1. The following sections provide information for each release series.

The current production release series is MySQL 4.1, which was declared stable for production use as of Version 4.1.7, released in October 2004. The previous production release series is MySQL 4.0, which was declared stable for production use as of Version 4.0.12, released in March 2003. Production status means that future 4.1 and 4.0 development is limited only to bugfixes. For the older MySQL 3.23 series, only critical bugfixes are made.

Active MySQL development currently is taking place in the MySQL 5.0 release series, this means that new features are being added there. MySQL 5.0 is available in alpha status.

Before upgrading from one release series to the next, please see the notes at Section 2.10, “Upgrading MySQL”.

Plans for some of the most requested features are summarized in the following table.

Feature	**MySQL Series**
Unions	4.0
Subqueries	4.1
R-trees	4.1 (for `MyISAM` tables)
Stored procedures	5.0
Views	5.0
Cursors	5.0
Foreign keys	5.1 (implemented in 3.23 for `InnoDB`)
Triggers	5.0 and 5.1
Full outer join	5.1
Constraints	5.1

1.3.1. MySQL 4.0 in a Nutshell

MySQL Server 4.0 is available in production status.

MySQL 4.0 is available for download at http://dev.mysql.com/ and from our mirrors. MySQL 4.0 has been tested by a large number of users and is in production use at many large sites.

The major new features of MySQL Server 4.0 are geared toward our existing business and community users, enhancing the MySQL database software as the solution for mission-critical, heavy-load database systems. Other new features target the users of embedded databases.

1.3.1.1. Features Available in MySQL 4.0

- Speed enhancements
  - MySQL 4.0 has a query cache that can give a huge speed boost to applications with repetitive queries. See Section 5.11, “The MySQL Query Cache”.
  - Version 4.0 further increases the speed of MySQL Server in a number of areas, such as bulk `INSERT` statements, searching on packed indexes, full-text searching (using `FULLTEXT` indexes), and `COUNT(DISTINCT)`.
- Embedded MySQL Server introduced
  - The new Embedded Server library can easily be used to create standalone and embedded applications. The embedded server provides an alternative to using MySQL in a client/server environment. See Section 1.3.1.2, “The Embedded MySQL Server”.
- `InnoDB` storage engine as standard
  - The `InnoDB` storage engine is offered as a standard feature of the MySQL server. This means

full support for ACID transactions, foreign keys with cascading `UPDATE` and `DELETE`, and row-level locking are standard features. See Chapter 15, *The `InnoDB` Storage Engine*.

- New functionality
  - The enhanced `FULLTEXT` search properties of MySQL Server 4.0 enables `FULLTEXT` indexing of large text masses with both binary and natural-language searching logic. You can customize minimal word length and define your own stop word lists in any human language, enabling a new set of applications to be built with MySQL Server. See Section 12.6, "Full-Text Search Functions".
- Standards compliance, portability, and migration
  - MySQL Server supports the `UNION` statement, a standard SQL feature.
  - MySQL runs natively on Novell NetWare 6.0 and higher. See Section 2.6, "Installing MySQL on NetWare".
  - Features to simplify migration from other database systems to MySQL Server include `TRUNCATE TABLE` (as in Oracle).
- Internationalization
  - Our German, Austrian, and Swiss users should note that MySQL 4.0 supports a new character set, `latin1_de`, which ensures that the *German sorting order* sorts words with umlauts in the same order as do German telephone books.
- Usability enhancements

  In the process of implementing features for new users, we have not forgotten requests from our loyal community of existing users.

  - Most **mysqld** parameters (startup options) can be set without taking down the server. This is a convenient feature for database administrators (DBAs). See Section 13.5.3, "`SET` Syntax".
  - Multiple-table `DELETE` and `UPDATE` statements have been added.
  - On Windows, symbolic link handling at the database level is enabled by default. On Unix, the `MyISAM` storage engine supports symbolic linking at the table level (and not just the database level as before).
  - `SQL_CALC_FOUND_ROWS` and `FOUND_ROWS()` are new functions that make it possible to find out the number of rows a `SELECT` query that includes a `LIMIT` clause would have returned without that clause.

The news section of this manual includes a more in-depth list of features. See Section D.3, "Changes in release 4.0.x (Production)".

1.3.1.2. The Embedded MySQL Server

The `libmysqld` embedded server library makes MySQL Server suitable for a vastly expanded realm of applications. By using this library, developers can embed MySQL Server into various applications and electronics devices, where the end user has no knowledge of there actually being an underlying database. Embedded MySQL Server is ideal for use behind the scenes in Internet appliances, public kiosks, turnkey hardware/software combination units, high performance Internet servers, self-contained databases distributed on CD-ROM, and so on.

Many users of `libmysqld` benefit from the MySQL Dual Licensing. For those not wishing to be bound by the GPL, the software is also made available under a commercial license. See http://www.mysql.com/company/legal/licensing/ for more information on the licensing policy of MySQL AB. The embedded MySQL library uses the same interface as the normal client library, so it is convenient and easy to use. See Section 22.2.16, “libmysqld, the Embedded MySQL Server Library”.

On Windows there are two different libraries:

`libmysqld.lib`	Dynamic library for threaded applications.
`mysqldemb.lib`	Static library for not threaded applications.

1.3.2. MySQL 4.1 in a Nutshell

MySQL Server 4.0 laid the foundation for new features implemented in MySQL 4.1, such as subqueries and Unicode support, and for the work on stored procedures being done in version 5.0. These features come at the top of the wish list of many of our customers. Well-known for its stability, speed, and ease of use, MySQL Server is able to fulfill the requirement checklists of very demanding buyers.

MySQL Server 4.1 is currently in production status, and binaries are available for download at http://dev.mysql.com/downloads/mysql/4.1.html. All binary releases pass our extensive test suite without any errors on the platforms on which we test. See Section D.2, “Changes in release 4.1.x (Production)”.

For those wishing to use the most recent development source for MySQL 4.1, we also make our BitKeeper repositories publicly available. See Section 2.8.3, “Installing from the Development Source Tree”.

1.3.2.1. Features Available in MySQL 4.1

This section lists features implemented in MySQL 4.1. Features that are available in MySQL 5.0 are described in Section C.1, “New Features Planned for 5.0”.

- Support for subqueries and derived tables
  - A ``subquery'' is a `SELECT` statement nested within another statement. A ``derived table'' (an unnamed view) is a subquery in the `FROM` clause of another statement. See Section 13.1.8, “Subquery Syntax”.
- Speed enhancements
  - Faster binary client/server protocol with support for prepared statements and parameter binding. See Section 22.2.4, “C API Prepared Statements”.
  - `BTREE` indexing is supported for `HEAP` tables, significantly improving response time for non-exact searches.
- New functionality
  - `CREATE TABLE` *`tbl_name2`* `LIKE` *`tbl_name1`* allows you to create, with a single statement, a new table with a structure exactly like that of an existing table.
  - The `MyISAM` storage engine supports OpenGIS spatial types for storing geographical data. See Chapter 18, *Spatial Extensions in MySQL*.
  - Replication can be done over SSL connections.

- Standards compliance, portability, and migration
  - The new client/server protocol adds the ability to pass multiple warnings to the client, rather than only a single result. This makes it much easier to track problems that occur in operations such as bulk data loading.
  - `SHOW WARNINGS` shows warnings for the last command. See Section 13.5.4.20, "`SHOW WARNINGS` Syntax".
- Internationalization and Localization
  - To support applications that require the use of local languages, the MySQL software offers extensive Unicode support through the `utf8` and `ucs2` character sets.
  - Character sets can be defined per column, table, and database. This allows for a high degree of flexibility in application design, particularly for multi-language Web sites.
  - For documentation for this improved character set support, see Chapter 10, *Character Set Support*.
  - Per-connection time zones are supported, allowing individual clients to select their own time zone when necessary.
- Usability enhancements
  - In response to popular demand, we have added a server-based `HELP` command that can be used to get help information for SQL statements. The advantage of having this information on the server side is that the information is always applicable to the particular server version that you actually are using. Because this information is available by issuing an SQL statement, any client can be written to access it. For example, the `help` command of the **mysql** command-line client has been modified to have this capability.
  - In the new client/server protocol, multiple statements can be issued with a single call. See Section 22.2.9, "C API Handling of Multiple Query Execution".
  - The new client/server protocol also supports returning multiple result sets. This might occur as a result of sending multiple statements, for example.
  - A new `INSERT ... ON DUPLICATE KEY UPDATE ...` syntax has been implemented. This allows you to `UPDATE` an existing row if the `INSERT` would have caused a duplicate in a `PRIMARY` or `UNIQUE` index. See Section 13.1.4, "`INSERT` Syntax".
  - A new aggregate function, `GROUP_CONCAT()` adds the extremely useful capability of concatenating column values from grouped rows into a single result string. See Section 12.9, "Functions and Modifiers for Use with `GROUP BY` Clauses".

The news section of this manual includes a more in-depth list of features. See Section D.2, "Changes in release 4.1.x (Production)".

1.3.3. MySQL 5.0: The Next Development Release

New development for MySQL is focused on the 5.0 release, featuring stored procedures, views (including updatable views), rudimentary triggers, and other new features. See Section C.1, "New Features Planned for 5.0".

For those wishing to take a look at the bleeding edge of MySQL development, we make our BitKeeper repository for MySQL version 5.0 publicly available. See Section 2.8.3, "Installing from the Develop-

ment Source Tree". As of December 2003, binary builds of version 5.0 have also been available.

1.4. MySQL Information Sources

1.4.1. MySQL Mailing Lists

This section introduces the MySQL mailing lists and provides guidelines as to how the lists should be used. When you subscribe to a mailing list, you receive all postings to the list as email messages. You can also send your own questions and answers to the list.

1.4.1.1. The MySQL Mailing Lists

To subscribe to or unsubscribe from any of the mailing lists described in this section, visit http://lists.mysql.com/. For most of them, you can select the regular version of the list where you get individual messages, or a digest version where you get one large message per day.

Please *do not* send messages about subscribing or unsubscribing to any of the mailing lists, because such messages are distributed automatically to thousands of other users.

Your local site may have many subscribers to a MySQL mailing list. If so, the site may have a local mailing list, so that messages sent from `lists.mysql.com` to your site are propagated to the local list. In such cases, please contact your system administrator to be added to or dropped from the local MySQL list.

If you wish to have traffic for a mailing list go to a separate mailbox in your mail program, set up a filter based on the message headers. You can use either the `List-ID:` or `Delivered-To:` headers to identify list messages.

The MySQL mailing lists are as follows:

- `announce`

  This list is for announcements of new versions of MySQL and related programs. This is a low-volume list to which all MySQL users should subscribe.

- `mysql`

  This is the main list for general MySQL discussion. Please note that some topics are better discussed on the more-specialized lists. If you post to the wrong list, you may not get an answer.

- `bugs`

  This list is for people who want to stay informed about issues reported since the last release of MySQL or who want to be actively involved in the process of bug hunting and fixing. See Section 1.4.1.3, "How to Report Bugs or Problems".

- `internals`

  This list is for people who work on the MySQL code. This is also the forum for discussions on MySQL development and for posting patches.

- `mysqldoc`

  This list is for people who work on the MySQL documentation: people from MySQL AB, translators, and other community members.

- `benchmarks`

This list is for anyone interested in performance issues. Discussions concentrate on database performance (not limited to MySQL), but also include broader categories such as performance of the kernel, filesystem, disk system, and so on.

- `packagers`

  This list is for discussions on packaging and distributing MySQL. This is the forum used by distribution maintainers to exchange ideas on packaging MySQL and on ensuring that MySQL looks and feels as similar as possible on all supported platforms and operating systems.

- `java`

  This list is for discussions about the MySQL server and Java. It is mostly used to discuss JDBC drivers, including MySQL Connector/J.

- `win32`

  This list is for all topics concerning the MySQL software on Microsoft operating systems, such as Windows 9x, Me, NT, 2000, XP, and 2003.

- `myodbc`

  This list is for all topics concerning connecting to the MySQL server with ODBC.

- `gui-tools`

  This list is for all topics concerning MySQL GUI tools, including `MySQL Administrator` and the `MySQL Control Center` graphical client.

- `cluster`

  This list is for discussion of MySQL Cluster.

- `dotnet`

  This list is for discussion of the MySQL server and the .NET platform. Mostly related to the MySQL Connector/Net provider.

- `plusplus`

  This list is for all topics concerning programming with the C++ API for MySQL.

- `perl`

  This list is for all topics concerning the Perl support for MySQL with `DBD::mysql`.

If you're unable to get an answer to your questions from a MySQL mailing list, one option is to purchase support from MySQL AB. This puts you in direct contact with MySQL developers.

The following table shows some MySQL mailing lists in languages other than English. These lists are not operated by MySQL AB.

- `<mysql-france-subscribe@yahoogroups.com>`

  A French mailing list.

- `<list@tinc.net>`

A Korean mailing list. Email `subscribe mysql your@email.address` to this list.

- `<mysql-de-request@lists.4t2.com>`

  A German mailing list. Email `subscribe mysql-de your@email.address` to this list. You can find information about this mailing list at http://www.4t2.com/mysql/.

- `<mysql-br-request@listas.linkway.com.br>`

  A Portuguese mailing list. Email `subscribe mysql-br your@email.address` to this list.

- `<mysql-alta@elistas.net>`

  A Spanish mailing list. Email `subscribe mysql your@email.address` to this list.

1.4.1.2. Asking Questions or Reporting Bugs

Before posting a bug report or question, please do the following:

- Start by searching the MySQL online manual at http://dev.mysql.com/doc/. We try to keep the manual up to date by updating it frequently with solutions to newly found problems. The change history (http://dev.mysql.com/doc/mysql/en/News.html) can be particularly useful since it is quite possible that a newer version contains a solution to your problem.
- Search in the bugs database at http://bugs.mysql.com/ to see whether the bug has been reported and fixed.
- Search the MySQL mailing list archives at http://lists.mysql.com/.
- You can also use http://www.mysql.com/search/ to search all the Web pages (including the manual) that are located at the MySQL AB Web site.

If you can't find an answer in the manual or the archives, check with your local MySQL expert. If you still can't find an answer to your question, please follow the guidelines on sending mail to a MySQL mailing list, outlined in the next section, before contacting us.

1.4.1.3. How to Report Bugs or Problems

The normal place to report bugs is http://bugs.mysql.com/, which is the address for our bugs database. This database is public, and can be browsed and searched by anyone. If you log in to the system, you can enter new reports.

Writing a good bug report takes patience, but doing it right the first time saves time both for us and for yourself. A good bug report, containing a full test case for the bug, makes it very likely that we will fix the bug in the next release. This section helps you write your report correctly so that you don't waste your time doing things that may not help us much or at all.

We encourage everyone to use the **mysqlbug** script to generate a bug report (or a report about any problem). **mysqlbug** can be found in the `scripts` directory (source distribution) and in the `bin` directory under your MySQL installation directory (binary distribution). If you are unable to use **mysqlbug** (for example, if you are running on Windows), it is still vital that you include all the necessary information noted in this section (most importantly, a description of the operating system and the MySQL version).

The **mysqlbug** script helps you generate a report by determining much of the following information

automatically, but if something important is missing, please include it with your message. Please read this section carefully and make sure that all the information described here is included in your report.

Preferably, you should test the problem using the latest production or development version of MySQL Server before posting. Anyone should be able to repeat the bug by just using `mysql test < script_file` on the included test case or by running the shell or Perl script that is included in the bug report.

All bugs posted in the bugs database at http://bugs.mysql.com/ are corrected or documented in the next MySQL release. If only minor code changes are needed to correct a problem, we may also post a patch that fixes the problem.

If you have found a sensitive security bug in MySQL, you can send email to `<security@mysql.com>`.

If you have a repeatable bug report, please report it to the bugs database at http://bugs.mysql.com/. Note that even in this case it's good to run the **mysqlbug** script first to find information about your system. Any bug that we are able to repeat has a high chance of being fixed in the next MySQL release.

To report other problems, you can use one of the MySQL mailing lists.

Remember that it is possible for us to respond to a message containing too much information, but not to one containing too little. People often omit facts because they think they know the cause of a problem and assume that some details don't matter. A good principle is this: If you are in doubt about stating something, state it. It is faster and less troublesome to write a couple more lines in your report than to wait longer for the answer if we must ask you to provide information that was missing from the initial report.

The most common errors made in bug reports are (a) not including the version number of the MySQL distribution used, and (b) not fully describing the platform on which the MySQL server is installed (including the platform type and version number). This is highly relevant information, and in 99 cases out of 100, the bug report is useless without it. Very often we get questions like, ``Why doesn't this work for me?'' Then we find that the feature requested wasn't implemented in that MySQL version, or that a bug described in a report has been fixed in newer MySQL versions. Sometimes the error is platform-dependent; in such cases, it is next to impossible for us to fix anything without knowing the operating system and the version number of the platform.

If you compiled MySQL from source, remember also to provide information about your compiler, if it is related to the problem. Often people find bugs in compilers and think the problem is MySQL-related. Most compilers are under development all the time and become better version by version. To determine whether your problem depends on your compiler, we need to know what compiler you use. Note that every compiling problem should be regarded as a bug and reported accordingly.

It is most helpful when a good description of the problem is included in the bug report. That is, give a good example of everything you did that led to the problem and describe, in exact detail, the problem itself. The best reports are those that include a full example showing how to reproduce the bug or problem. See Section E.1.6, “Making a Test Case If You Experience Table Corruption”.

If a program produces an error message, it is very important to include the message in your report. If we try to search for something from the archives using programs, it is better that the error message reported exactly matches the one that the program produces. (Even the lettercase should be observed.) You should never try to reproduce from memory what the error message was; instead, copy and paste the entire message into your report.

If you have a problem with Connector/ODBC (MyODBC), please try to generate a trace file and send it with your report. See Section 23.1.1.9, “How to Report MyODBC Problems or Bugs”.

Please remember that many of the people who read your report do so using an 80-column display. When generating reports or examples using the **mysql** command-line tool, you should therefore use the -

`-vertical` option (or the `\G` statement terminator) for output that would exceed the available width for such a display (for example, with the `EXPLAIN SELECT` statement; see the example later in this section).

Please include the following information in your report:

- The version number of the MySQL distribution you are using (for example, MySQL 4.0.12). You can find out which version you are running by executing **mysqladmin version**. The **mysqladmin** program can be found in the `bin` directory under your MySQL installation directory.
- The manufacturer and model of the machine on which you experience the problem.
- The operating system name and version. If you work with Windows, you can usually get the name and version number by double-clicking your My Computer icon and pulling down the ``Help/About Windows'' menu. For most Unix-like operating systems, you can get this information by executing the command `uname -a`.
- Sometimes the amount of memory (real and virtual) is relevant. If in doubt, include these values.
- If you are using a source distribution of the MySQL software, the name and version number of the compiler used are needed. If you have a binary distribution, the distribution name is needed.
- If the problem occurs during compilation, include the exact error messages and also a few lines of context around the offending code in the file where the error occurs.
- If **mysqld** died, you should also report the query that crashed **mysqld**. You can usually find this out by running **mysqld** with query logging enabled, and then looking in the log after **mysqld** crashes See Section E.1.5, “Using Log Files to Find Cause of Errors in **mysqld**”.
- If a database table is related to the problem, include the output from **mysqldump --no-data** ***`db_name tbl_name`***. This is very easy to do and is a powerful way to get information about any table in a database. The information helps us create a situation matching the one you have experienced.
- For speed-related bugs or problems with `SELECT` statements, you should always include the output of `EXPLAIN SELECT ...`, and at least the number of rows that the `SELECT` statement produces. You should also include the output from `SHOW CREATE TABLE` *`tbl_name`* for each involved table. The more information you give about your situation, the more likely it is that someone can help you.

  The following is an example of a very good bug report. It should be posted with the **mysqlbug** script. The example uses the **mysql** command-line tool. Note the use of the `\G` statement terminator for statements whose output width would otherwise exceed that of an 80-column display device.

  ```
  mysql> SHOW VARIABLES;
  mysql> SHOW COLUMNS FROM ...\G
         <output from SHOW COLUMNS>
  mysql> EXPLAIN SELECT ...\G
         <output from EXPLAIN>
  mysql> FLUSH STATUS;
  mysql> SELECT ...;
         <A short version of the output from SELECT,
         including the time taken to run the query>
  mysql> SHOW STATUS;
         <output from SHOW STATUS>
  ```

- If a bug or problem occurs while running **mysqld**, try to provide an input script that reproduces the anomaly. This script should include any necessary source files. The more closely the script can reproduce your situation, the better. If you can make a reproducible test case, you should post it on ht-

tp://bugs.mysql.com/ for high-priority treatment.

If you can't provide a script, you should at least include the output from **mysqladmin variables extended-status processlist** in your mail to provide some information on how your system is performing.

- If you can't produce a test case with only a few rows, or if the test table is too big to be mailed to the mailing list (more than 10 rows), you should dump your tables using **mysqldump** and create a `README` file that describes your problem.

  Create a compressed archive of your files using **tar** and **gzip** or **zip**, and use FTP to transfer the archive to ftp://ftp.mysql.com/pub/mysql/upload/. Then enter the problem into our bugs database at http://bugs.mysql.com/.

- If you think that the MySQL server produces a strange result from a query, include not only the result, but also your opinion of what the result should be, and an account describing the basis for your opinion.

- When giving an example of the problem, it's better to use the variable names, table names, and so on that exist in your actual situation than to come up with new names. The problem could be related to the name of a variable or table. These cases are rare, perhaps, but it is better to be safe than sorry. After all, it should be easier for you to provide an example that uses your actual situation, and it is by all means better for us. In case you have data that you don't want to show to others, you can use FTP to transfer it to ftp://ftp.mysql.com/pub/mysql/upload/. If the information is really top secret and you don't want to show it even to us, then go ahead and provide an example using other names, but please regard this as the last choice.

- Include all the options given to the relevant programs, if possible. For example, indicate the options that you use when you start the **mysqld** server as well as the options that you use to run any MySQL client programs. The options to programs such as **mysqld** and **mysql**, and to the **configure** script, are often keys to answers and are very relevant. It is never a bad idea to include them. If you use any modules, such as Perl or PHP, please include the version numbers of those as well.

- If your question is related to the privilege system, please include the output of **mysqlaccess**, the output of **mysqladmin reload**, and all the error messages you get when trying to connect. When you test your privileges, you should first run **mysqlaccess**. After this, execute **mysqladmin reload version** and try to connect with the program that gives you trouble. **mysqlaccess** can be found in the `bin` directory under your MySQL installation directory.

- If you have a patch for a bug, do include it. But don't assume that the patch is all we need, or that we can use it, if you don't provide some necessary information such as test cases showing the bug that your patch fixes. We might find problems with your patch or we might not understand it at all; if so, we can't use it.

  If we can't verify exactly what the purpose of the patch is, we won't use it. Test cases help us here. Show that the patch handles all the situations that may occur. If we find a borderline case (even a rare one) where the patch won't work, it may be useless.

- Guesses about what the bug is, why it occurs, or what it depends on are usually wrong. Even the MySQL team can't guess such things without first using a debugger to determine the real cause of a bug.

- Indicate in your bug report that you have checked the reference manual and mail archive so that others know you have tried to solve the problem yourself.

- If you get a `parse error`, please check your syntax closely. If you can't find something wrong with it, it's extremely likely that your current version of MySQL Server doesn't support the syntax you are using. If you are using the current version and the manual at http://dev.mysql.com/doc/ doesn't cover the syntax you are using, MySQL Server doesn't support your query. In this case, your

only options are to implement the syntax yourself or email `<licensing@mysql.com>` and ask for an offer to implement it.

If the manual covers the syntax you are using, but you have an older version of MySQL Server, you should check the MySQL change history to see when the syntax was implemented. In this case, you have the option of upgrading to a newer version of MySQL Server. See Appendix D, *MySQL Change History*.

- If your problem is that your data appears corrupt or you get errors when you access a particular table, you should first check and then try to repair your tables with `CHECK TABLE` and `REPAIR TABLE` or with **myisamchk**. See Chapter 5, *Database Administration*.

  If you are running Windows, please verify that `lower_case_table_names` is 1 or 2 with `SHOW VARIABLES LIKE 'lower_case_table_names'`.

- If you often get corrupted tables, you should try to find out when and why this happens. In this case, the error log in the MySQL data directory may contain some information about what happened. (This is the file with the `.err` suffix in the name.) See Section 5.9.1, “The Error Log”. Please include any relevant information from this file in your bug report. Normally **mysqld** should *never* crash a table if nothing killed it in the middle of an update. If you can find the cause of **mysqld** dying, it's much easier for us to provide you with a fix for the problem. See Section A.1, “How to Determine What Is Causing a Problem”.

- If possible, download and install the most recent version of MySQL Server and check whether it solves your problem. All versions of the MySQL software are thoroughly tested and should work without problems. We believe in making everything as backward-compatible as possible, and you should be able to switch MySQL versions without difficulty. See Section 2.1.2, “Choosing Which MySQL Distribution to Install”.

If you are a support customer, please cross-post the bug report to `<mysql-support@mysql.com>` for higher-priority treatment, as well as to the appropriate mailing list to see whether someone else has experienced (and perhaps solved) the problem.

For information on reporting bugs in MyODBC, see Section 23.1.1.9, “How to Report MyODBC Problems or Bugs”.

For solutions to some common problems, see Appendix A, *Problems and Common Errors*.

When answers are sent to you individually and not to the mailing list, it is considered good etiquette to summarize the answers and send the summary to the mailing list so that others may have the benefit of responses you received that helped you solve your problem.

1.4.1.4. Guidelines for Answering Questions on the Mailing List

If you consider your answer to have broad interest, you may want to post it to the mailing list instead of replying directly to the individual who asked. Try to make your answer general enough that people other than the original poster may benefit from it. When you post to the list, please make sure that your answer is not a duplication of a previous answer.

Try to summarize the essential part of the question in your reply; don't feel obliged to quote the entire original message.

Please don't post mail messages from your browser with HTML mode turned on. Many users don't read mail with a browser.

1.4.2. MySQL Community Support on IRC (Internet Relay

Chat)

In addition to the various MySQL mailing lists, you can find experienced community people on `IRC` (`Internet Relay Chat`). These are the best networks/channels currently known to us:

- **freenode** (see http://www.freenode.net/ for servers)
  - `#mysql` Primarily MySQL questions, but other database and general SQL questions are welcome. Questions about PHP, Perl or C in combination with MySQL are also common.

If you are looking for IRC client software to connect to an IRC network, take a look at `X-Chat` (http://www.xchat.org/). X-Chat (GPL licensed) is available for Unix as well as for Windows platforms (a free Windows build of X-Chat is available at http://www.silverex.org/download/).

1.4.3. MySQL Community Support at the MySQL Forums

The latest community support resource are the forums at http://forums.mysql.com.

There are a variety of forums available, grouped in the following general categories:

- Migration
- MySQL Usage
- MySQL Connectors
- MySQL Technology
- Business

1.5. MySQL Standards Compliance

This section describes how MySQL relates to the ANSI/ISO SQL standards. MySQL Server has many extensions to the SQL standard, and here you can find out what they are and how to use them. You can also find information about functionality missing from MySQL Server, and how to work around some differences.

The SQL standard has been evolving since 1986 and several versions exist. In this manual, ``SQL-92'' refers to the standard released in 1992, ``SQL:1999'' refers to the standard released in 1999, and ``SQL:2003'' refers to the current version of the standard. We use the phrase ``the SQL standard'' to mean the current version of the SQL Standard at any time.

Our goal is to not restrict MySQL Server usability for any usage without a very good reason for doing so. Even if we don't have the resources to perform development for every possible use, we are always willing to help and offer suggestions to people who are trying to use MySQL Server in new territories.

One of our main goals with the product is to continue to work toward compliance with the SQL standard, but without sacrificing speed or reliability. We are not afraid to add extensions to SQL or support for non-SQL features if this greatly increases the usability of MySQL Server for a large segment of our user base. The `HANDLER` interface in MySQL Server 4.0 is an example of this strategy. See Section 13.1.3, “`HANDLER` Syntax”.

We continue to support transactional and non-transactional databases to satisfy both mission-critical 24/7 usage and heavy Web or logging usage.

MySQL Server was originally designed to work with medium size databases (10-100 million rows, or about 100MB per table) on small computer systems. Today MySQL Server handles terabyte-size databases, but the code can also be compiled in a reduced version suitable for hand-held and embedded devices. The compact design of the MySQL server makes development in both directions possible without any conflicts in the source tree.

Currently, we are not targeting realtime support, although MySQL replication capabilities offer significant functionality.

Database cluster support exists through third-party clustering solutions as well as the integration of our acquired NDB Cluster technology, available from version 4.1.2. See Chapter 16, *MySQL Cluster*.

We are also looking at providing XML support in the database server.

1.5.1. What Standards MySQL Follows

We are aiming toward supporting the full ANSI/ISO SQL standard, but without making concessions to speed and quality of the code.

ODBC levels 0#3.51.

1.5.2. Selecting SQL Modes

The MySQL server can operate in different SQL modes, and can apply these modes differentially for different clients. This allows an application to tailor server operation to its own requirements.

Modes define what SQL syntax MySQL should support and what kind of validation checks it should perform on the data. This makes it easier to use MySQL in a lot of different environments and to use MySQL together with other database servers.

You can set the default SQL mode by starting **mysqld** with the `--sql-mode="modes"` option. Beginning with MySQL 4.1, you can also change the mode after startup time by setting the `sql_mode` variable with a `SET [SESSION|GLOBAL] sql_mode='modes'` statement.

For more information on setting the server mode, see Section 5.2.2, “The Server SQL Mode”.

1.5.3. Running MySQL in ANSI Mode

You can tell **mysqld** to use the ANSI mode with the `--ansi` startup option. See Section 5.2.1, “**mysqld** Command-Line Options”.

Running the server in ANSI mode is the same as starting it with these options (specify the `--sql_mode` value on a single line):

```
--transaction-isolation=SERIALIZABLE
--sql-mode=REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,
IGNORE_SPACE
```

In MySQL 4.1, you can achieve the same effect with these two statements (specify the `sql_mode` value on a single line):

```
SET GLOBAL TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET GLOBAL sql_mode = 'REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,
IGNORE_SPACE';
```

See Section 1.5.2, “Selecting SQL Modes”.

In MySQL 4.1.1, the `sql_mode` options shown can be also be set with this statement:

```
SET GLOBAL sql_mode='ansi';
```

In this case, the value of the `sql_mode` variable is set to all options that are relevant for ANSI mode. You can check the result like this:

```
mysql> SET GLOBAL sql_mode='ansi';
mysql> SELECT @@global.sql_mode;
        -> 'REAL_AS_FLOAT,PIPES_AS_CONCAT,ANSI_QUOTES,
            IGNORE_SPACE,ANSI';
```

1.5.4. MySQL Extensions to Standard SQL

MySQL Server includes some extensions that you probably won't find in other SQL databases. Be warned that if you use them, your code won't be portable to other SQL servers. In some cases, you can write code that includes MySQL extensions, but is still portable, by using comments of the form `/*! ... */`. In this case, MySQL Server parses and execute the code within the comment as it would any other MySQL statement, but other SQL servers will ignore the extensions. For example:

```
SELECT /*! STRAIGHT_JOIN */ col_name FROM table1,table2 WHERE ...
```

If you add a version number after the `'!'` character, the syntax within the comment is executed only if the MySQL version is equal to or newer than the specified version number:

```
CREATE /*!32302 TEMPORARY */ TABLE t (a INT);
```

This means that if you have Version 3.23.02 or newer, MySQL Server uses the `TEMPORARY` keyword.

The following descriptions list MySQL extensions, organized by category.

- Organization of data on disk

  MySQL Server maps each database to a directory under the MySQL data directory, and tables within a database to filenames in the database directory. This has a few implications:

  - Database names and table names are case sensitive in MySQL Server on operating systems that have case-sensitive filenames (such as most Unix systems). See Section 9.2.2, “Identifier Case Sensitivity”.
  - You can use standard system commands to back up, rename, move, delete, and copy tables that are managed by the `MyISAM` or `ISAM` storage engines. For example, to rename a `MyISAM` table, rename the `.MYD`, `.MYI`, and `.frm` files to which the table corresponds.

  Database, table, index, column, or alias names may begin with a digit (but may not consist solely of digits).

- General language syntax

  - Strings may be enclosed by either `'"'` or `''`, not just by `''`.
  - Use of `'\'` as an escape character in strings.

  - In SQL statements, you can access tables from different databases with the *db_name.tbl_name* syntax. Some SQL servers provide the same functionality but call this User space. MySQL Server doesn't support tablespaces such as used in statements like this: CREATE TABLE ralph.my_table...IN my_tablespace.

- SQL statement syntax

  - The ANALYZE TABLE, CHECK TABLE, OPTIMIZE TABLE, and REPAIR TABLE statements.
  - The CREATE DATABASE and DROP DATABASE statements. See Section 13.2.4, “CREATE DATABASE Syntax”.
  - The DO statement.
  - EXPLAIN SELECT to get a description of how tables are joined.
  - The FLUSH and RESET statements.
  - The SET statement. See Section 13.5.3, “SET Syntax”.
  - The SHOW statement. See Section 13.5.4, “SHOW Syntax”.
  - Use of LOAD DATA INFILE. In many cases, this syntax is compatible with Oracle's LOAD DATA INFILE. See Section 13.1.5, “LOAD DATA INFILE Syntax”.
  - Use of RENAME TABLE. See Section 13.2.12, “RENAME TABLE Syntax”.
  - Use of REPLACE instead of DELETE + INSERT. See Section 13.1.6, “REPLACE Syntax”.
  - Use of CHANGE col_name, DROP col_name, or DROP INDEX, IGNORE or RENAME in an ALTER TABLE statement. Use of multiple ADD, ALTER, DROP, or CHANGE clauses in an ALTER TABLE statement. See Section 13.2.2, “ALTER TABLE Syntax”.
  - Use of index names, indexes on a prefix of a field, and use of INDEX or KEY in a CREATE TABLE statement. See Section 13.2.6, “CREATE TABLE Syntax”.
  - Use of TEMPORARY or IF NOT EXISTS with CREATE TABLE.
  - Use of IF EXISTS with DROP TABLE.
  - You can drop multiple tables with a single DROP TABLE statement.
  - The ORDER BY and LIMIT clauses of the UPDATE and DELETE statements.
  - INSERT INTO ... SET col_name = ... syntax.
  - The DELAYED clause of the INSERT and REPLACE statements.
  - The LOW_PRIORITY clause of the INSERT, REPLACE, DELETE, and UPDATE statements.
  - Use of INTO OUTFILE and STRAIGHT_JOIN in a SELECT statement. See Section 13.1.7, “SELECT Syntax”.
  - The SQL_SMALL_RESULT option in a SELECT statement.
  - You don't need to name all selected columns in the GROUP BY part. This gives better performance for some very specific, but quite normal queries. See Section 12.9, “Functions and Modifi-

ers for Use with GROUP BY Clauses".

- You can specify ASC and DESC with GROUP BY.
- The ability to set variables in a statement with the := assignment operator:

```
mysql> SELECT @a:=SUM(total),@b=COUNT(*),@a/@b AS avg
    -> FROM test_table;
mysql> SELECT @t1:=(@t2:=1)+@t3:=4,@t1,@t2,@t3;
```

- Column types
  - The column types MEDIUMINT, SET, ENUM, and the different BLOB and TEXT types.
  - The column attributes AUTO_INCREMENT, BINARY, NULL, UNSIGNED, and ZEROFILL.
- Functions and operators
  - To make it easier for users who come from other SQL environments, MySQL Server supports aliases for many functions. For example, all string functions support both standard SQL syntax and ODBC syntax.
  - MySQL Server understands the || and && operators to mean logical OR and AND, as in the C programming language. In MySQL Server, || and OR are synonyms, as are && and AND. Because of this nice syntax, MySQL Server doesn't support the standard SQL || operator for string concatenation; use CONCAT() instead. Because CONCAT() takes any number of arguments, it's easy to convert use of the || operator to MySQL Server.
  - Use of COUNT(DISTINCT list) where list has more than one element.
  - All string comparisons are case-insensitive by default, with sort ordering determined by the current character set (ISO-8859-1 Latin1 by default). If you don't like this, you should declare your columns with the BINARY attribute or use the BINARY cast, which causes comparisons to be done using the underlying character code values rather then a lexical ordering.
  - The % operator is a synonym for MOD(). That is, N % M is equivalent to MOD(N,M). % is supported for C programmers and for compatibility with PostgreSQL.
  - The =, <>, <= ,<, >=,>, <<, >>, <=>, AND, OR, or LIKE operators may be used in column comparisons to the left of the FROM in SELECT statements. For example:

    ```
    mysql> SELECT col1=1 AND col2=2 FROM tbl_name;
    ```

  - The LAST_INSERT_ID() function that returns the most recent AUTO_INCREMENT value. See Section 12.8.3, "Information Functions".
  - LIKE is allowed on numeric columns.
  - The REGEXP and NOT REGEXP extended regular expression operators.
  - CONCAT() or CHAR() with one argument or more than two arguments. (In MySQL Server, these functions can take any number of arguments.)
  - The BIT_COUNT(), CASE, ELT(), FROM_DAYS(), FORMAT(), IF(), PASSWORD(), ENCRYPT(), MD5(), ENCODE(), DECODE(), PERIOD_ADD(), PERIOD_DIFF(),

TO_DAYS(), and WEEKDAY() functions.

- Use of TRIM() to trim substrings. Standard SQL supports removal of single characters only.
- The GROUP BY functions STD(), BIT_OR(), BIT_AND(), BIT_XOR(), and GROUP_CONCAT(). See Section 12.9, “Functions and Modifiers for Use with GROUP BY Clauses”.

For a prioritized list indicating when new extensions are added to MySQL Server, you should consult the online MySQL TODO list at http://dev.mysql.com/doc/mysql/en/TODO.html. That is the latest version of the TODO list in this manual. See Appendix C, *MySQL and the Future (the TODO)*.

1.5.5. MySQL Differences from Standard SQL

We try to make MySQL Server follow the ANSI SQL standard and the ODBC SQL standard, but MySQL Server performs operations differently in some cases:

- For VARCHAR columns, trailing spaces are removed when the value is stored. (Fixed in MySQL 5.0.3). See Section 1.5.7, “Known Errors and Design Deficiencies in MySQL”.
- In some cases, CHAR columns are silently converted to VARCHAR columns when you define a table or alter its structure. (Fixed in MySQL 5.0.3). See Section 13.2.6.1, “Silent Column Specification Changes”.
- Privileges for a table are not automatically revoked when you delete a table. You must explicitly issue a REVOKE statement to revoke privileges for a table. See Section 13.5.1.3, “GRANT and REVOKE Syntax”.
- The CAST() function does not support cast to REAL or BIGINT. See Section 12.7, “Cast Functions and Operators”.
- Standard SQL requires that a HAVING clause in a SELECT statement be able to refer to columns in the GROUP BY clause. This cannot be done before MySQL 5.0.2.

1.5.5.1. Subqueries

MySQL 4.1 supports subqueries and derived tables. A ``subquery'' is a SELECT statement nested within another statement. A ``derived table'' (an unnamed view) is a subquery in the FROM clause of another statement. See Section 13.1.8, “Subquery Syntax”.

For MySQL versions older than 4.1, most subqueries can be rewritten using joins or other methods. See Section 13.1.8.11, “Rewriting Subqueries as Joins for Earlier MySQL Versions” for examples that show how to do this.

1.5.5.2. SELECT INTO TABLE

MySQL Server doesn't support the Sybase SQL extension: SELECT ... INTO TABLE .... Instead, MySQL Server supports the standard SQL syntax INSERT INTO ... SELECT ..., which is basically the same thing. See Section 13.1.4.1, “INSERT ... SELECT Syntax”.

```
INSERT INTO tbl_temp2 (fld_id)
    SELECT tbl_temp1.fld_order_id
    FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

Alternatively, you can use `SELECT INTO OUTFILE ...` or `CREATE TABLE ... SELECT`.

From version 5.0, MySQL supports `SELECT ... INTO` with user variables. The same syntax may also be used inside stored procedures using cursors and local variables. See Section 19.2.9.3, "`SELECT ... INTO` Statement".

1.5.5.3. Transactions and Atomic Operations

MySQL Server (version 3.23-max and all versions 4.0 and above) supports transactions with the `InnoDB` and `BDB` transactional storage engines. `InnoDB` provides *full* `ACID` compliance. See Chapter 14, *MySQL Storage Engines and Table Types*.

The other non-transactional storage engines in MySQL Server (such as `MyISAM`) follow a different paradigm for data integrity called ``atomic operations.'' In transactional terms, `MyISAM` tables effectively always operate in `AUTOCOMMIT=1` mode. Atomic operations often offer comparable integrity with higher performance.

With MySQL Server supporting both paradigms, you can decide whether your applications are best served by the speed of atomic operations or the use of transactional features. This choice can be made on a per-table basis.

As noted, the trade-off for transactional versus non-transactional table types lies mostly in performance. Transactional tables have significantly higher memory and diskspace requirements, and more CPU overhead. On the other hand, transactional table types such as `InnoDB` also offer many significant features. MySQL Server's modular design allows the concurrent use of different storage engines to suit different requirements and deliver optimum performance in all situations.

But how do you use the features of MySQL Server to maintain rigorous integrity even with the non-transactional `MyISAM` tables, and how do these features compare with the transactional table types?

1. If your applications are written in a way that is dependent on being able to call `ROLLBACK` rather than `COMMIT` in critical situations, transactions are more convenient. Transactions also ensure that unfinished updates or corrupting activities are not committed to the database; the server is given the opportunity to do an automatic rollback and your database is saved.

   If you use non-transactional tables, MySQL Server in almost all cases allows you to resolve potential problems by including simple checks before updates and by running simple scripts that check the databases for inconsistencies and automatically repair or warn if such an inconsistency occurs. Note that just by using the MySQL log or even adding one extra log, you can normally fix tables perfectly with no data integrity loss.

2. More often than not, critical transactional updates can be rewritten to be atomic. Generally speaking, all integrity problems that transactions solve can be done with `LOCK TABLES` or atomic updates, ensuring that there are no automatic aborts from the server, which is a common problem with transactional database systems.

3. Even a transactional system can lose data if the server goes down. The difference between different systems lies in just how small the time-lag is where they could lose data. No system is 100% secure, only ``secure enough.'' Even Oracle, reputed to be the safest of transactional database systems, is reported to sometimes lose data in such situations.

   To be safe with MySQL Server, whether or not using transactional tables, you only need to have backups and have binary logging turned on. With this you can recover from any situation that you could with any other transactional database system. It is always good to have backups, regardless of which database system you use.

The transactional paradigm has its benefits and its drawbacks. Many users and application developers depend on the ease with which they can code around problems where an abort appears to be, or is necessary. However, even if you are new to the atomic operations paradigm, or more familiar with transactions, do consider the speed benefit that non-transactional tables can offer on the order of three to five times the speed of the fastest and most optimally tuned transactional tables.

In situations where integrity is of highest importance, MySQL Server offers transaction-level reliability and integrity even for non-transactional tables. If you lock tables with `LOCK TABLES`, all updates stall until integrity checks are made. If you obtain a `READ LOCAL` lock (as opposed to a write lock) for a table that allows concurrent inserts at the end of the table, reads are allowed, as are inserts by other clients. The newly inserted records are not be seen by the client that has the read lock until it releases the lock. With `INSERT DELAYED`, you can queue inserts into a local queue, until the locks are released, without having the client wait for the insert to complete. See Section 13.1.4.2, “`INSERT DELAYED` Syntax”.

``Atomic,'' in the sense that we mean it, is nothing magical. It only means that you can be sure that while each specific update is running, no other user can interfere with it, and there can never be an automatic rollback (which can happen with transactional tables if you are not very careful). MySQL Server also guarantees that there are no dirty reads.

Following are some techniques for working with non-transactional tables:

- Loops that need transactions normally can be coded with the help of `LOCK TABLES`, and you don't need cursors to update records on the fly.
- To avoid using `ROLLBACK`, you can use the following strategy:
  1. Use `LOCK TABLES` to lock all the tables you want to access.
  2. Test the conditions that must be true before performing the update.
  3. Update if everything is okay.
  4. Use `UNLOCK TABLES` to release your locks.

  This is usually a much faster method than using transactions with possible rollbacks, although not always. The only situation this solution doesn't handle is when someone kills the threads in the middle of an update. In this case, all locks are released but some of the updates may not have been executed.
- You can also use functions to update records in a single operation. You can get a very efficient application by using the following techniques:
  - Modify columns relative to their current value.
  - Update only those columns that actually have changed.

  For example, when we are doing updates to some customer information, we update only the customer data that has changed and test only that none of the changed data, or data that depends on the changed data, has changed compared to the original row. The test for changed data is done with the `WHERE` clause in the `UPDATE` statement. If the record wasn't updated, we give the client a message: ``Some of the data you have changed has been changed by another user.'' Then we show the old row versus the new row in a window so that the user can decide which version of the customer record to use.

  This gives us something that is similar to column locking but is actually even better because we only update some of the columns, using values that are relative to their current values. This means that typical `UPDATE` statements look something like these:

  ```
  UPDATE tablename SET pay_back=pay_back+125;
  ```

```
UPDATE customer
  SET
    customer_date='current_date',
    address='new address',
    phone='new phone',
    money_owed_to_us=money_owed_to_us-125
  WHERE
    customer_id=id AND address='old address' AND phone='old phone';
```

This is very efficient and works even if another client has changed the values in the `pay_back` or `money_owed_to_us` columns.

- In many cases, users have wanted `LOCK TABLES` and/or `ROLLBACK` for the purpose of managing unique identifiers. This can be handled much more efficiently without locking or rolling back by using an `AUTO_INCREMENT` column and either the `LAST_INSERT_ID()` SQL function or the `mysql_insert_id()` C API function. See Section 12.8.3, "Information Functions". See Section 22.2.3.33, "`mysql_insert_id()`".

  You can generally code around the need for row-level locking. Some situations really do need it, and `InnoDB` tables support row-level locking. With `MyISAM` tables, you can use a flag column in the table and do something like the following:

  ```
  UPDATE tbl_name SET row_flag=1 WHERE id=ID;
  ```

  MySQL returns `1` for the number of affected rows if the row was found and `row_flag` wasn't `1` in the original row.

  You can think of it as though MySQL Server changed the preceding query to:

  ```
  UPDATE tbl_name SET row_flag=1 WHERE id=ID AND row_flag <> 1;
  ```

1.5.5.4. Stored Procedures and Triggers

Stored procedures are implemented in MySQL version 5.0. See Chapter 19, *Stored Procedures and Functions*.

Triggers are currently being implemented, with basic functionality in MySQL 5.0, with further development planned for MySQL 5.1.

1.5.5.5. Foreign Keys

In MySQL Server 3.23.44 and up, the `InnoDB` storage engine supports checking of foreign key constraints, including `CASCADE`, `ON DELETE`, and `ON UPDATE`. See Section 15.7.4, "`FOREIGN KEY` Constraints".

For storage engines other than `InnoDB`, MySQL Server parses the `FOREIGN KEY` syntax in `CREATE TABLE` statements, but does not use or store it. In the future, the implementation will be extended to store this information in the table specification file so that it may be retrieved by **mysqldump** and ODBC. At a later stage, foreign key constraints will be implemented for `MyISAM` tables as well.

Foreign key enforcement offers several benefits to database developers:

- Assuming proper design of the relationships, foreign key constraints make it more difficult for a programmer to introduce an inconsistency into the database.
- Centralized checking of constraints by the database server makes it unnecessary to perform these checks on the application side. This eliminates the possibility that different applications may not all check the constraints in the same way.
- Using cascading updates and deletes can simplify the application code.
- Properly designed foreign key rules aid in documenting relationships between tables.

Do keep in mind that these benefits come at the cost of additional overhead for the database server to perform the necessary checks. Additional checking by the server affects performance, which for some applications may be sufficiently undesirable as to be avoided if possible. (Some major commercial applications have coded the foreign-key logic at the application level for this reason.)

MySQL gives database developers the choice of which approach to use. If you don't need foreign keys and want to avoid the overhead associated with enforcing referential integrity, you can choose another table type instead, such as `MyISAM`. (For example, the `MyISAM` storage engine offers very fast performance for applications that perform only `INSERT` and `SELECT` operations, because the inserts can be performed concurrently with retrievals. See Section 7.3.2, “Table Locking Issues”.)

If you choose not to take advantage of referential integrity checks, keep the following considerations in mind:

- In the absence of server-side foreign key relationship checking, the application itself must handle relationship issues. For example, it must take care to insert rows into tables in the proper order, and to avoid creating orphaned child records. It must also be able to recover from errors that occur in the middle of multiple-record insert operations.
- If `ON DELETE` is the only referential integrity capability an application needs, note that as of MySQL Server 4.0, you can use multiple-table `DELETE` statements to delete rows from many tables with a single statement. See Section 13.1.1, “`DELETE` Syntax”.
- A workaround for the lack of `ON DELETE` is to add the appropriate `DELETE` statement to your application when you delete records from a table that has a foreign key. In practice, this is often as quick as using foreign keys, and is more portable.

Be aware that the use of foreign keys can in some instances lead to problems:

- Foreign key support addresses many referential integrity issues, but it is still necessary to design key relationships carefully to avoid circular rules or incorrect combinations of cascading deletes.
- It is not uncommon for a DBA to create a topology of relationships that makes it difficult to restore individual tables from a backup. (MySQL alleviates this difficulty by allowing you to temporarily disable foreign key checks when reloading a table that depends on other tables. See Section 15.7.4, “`FOREIGN KEY` Constraints”. As of MySQL 4.1.1, **mysqldump** generates dump files that take advantage of this capability automatically when reloaded.)

Note that foreign keys in SQL are used to check and enforce referential integrity, not to join tables. If you want to get results from multiple tables from a `SELECT` statement, you do this by performing a join between them:

```
SELECT * FROM t1, t2 WHERE t1.id = t2.id;
```

See Section 13.1.7.1, “`JOIN` Syntax”. See Section 3.6.6, “Using Foreign Keys”.

The `FOREIGN KEY` syntax without `ON DELETE ...` is often used by ODBC applications to produce automatic `WHERE` clauses.

1.5.5.6. Views

Views (including updatable views) are implemented in the 5.0 version of MySQL Server. Views are available in binary releases from 5.0.1 and up. See Section 13.2.7, “`CREATE VIEW` Syntax”.

Views are useful for allowing users to access a set of relations (tables) as if it were a single table, and limiting their access to just that. Views can also be used to restrict access to rows (a subset of a particular table). For access control to columns, you can also use the sophisticated privilege system in MySQL Server. See Section 5.5, “The MySQL Access Privilege System”.

In designing an implementation of views, our ambitious goal, as much as is possible within the confines of SQL, has been full compliance with ``Codd's Rule #6" for relational database systems: ``All views that are theoretically updatable, should in practice also be updatable."

1.5.5.7. '- -' as the Start of a Comment

Some other SQL databases use '`--`' to start comments. MySQL Server uses '#' as the start comment character. You can also use the C comment style `/* this is a comment */` with MySQL Server. See Section 9.5, “Comment Syntax”.

MySQL Server 3.23.3 and above support the '`--`' comment style, provided the comment is followed by a space (or by a control character such as a newline). The requirement for a space is to prevent problems with automatically generated SQL queries that have used something like the following code, where we automatically insert the value of the payment for `!payment!`:

```
UPDATE account SET credit=credit-!payment!
```

Think about what happens if the value of `payment` is a negative value such as `-1`:

```
UPDATE account SET credit=credit--1
```

`credit--1` is a legal expression in SQL, but if `--` is interpreted as the start of a comment, part of the expression is discarded. The result is a statement that has a completely different meaning than intended:

```
UPDATE account SET credit=credit
```

The statement produces no change in value at all! This illustrates that allowing comments to start with '`--`' can have serious consequences.

Using our implementation of this method of commenting in MySQL Server 3.23.3 and up, `credit--1` is actually safe.

Another safe feature is that the **mysql** command-line client removes all lines that start with '`--`'.

The following information is relevant only if you are running a MySQL version earlier than 3.23.3:

If you have an SQL program in a text file that contains '`--`' comments, you should use the **replace** utility as follows to convert the comments to use '#' characters:

```
shell> replace " --" " #" < text-file-with-funny-comments.sql \
         | mysql db_name
```

instead of the usual:

```
shell> mysql db_name < text-file-with-funny-comments.sql
```

You can also edit the command file ``in place'' to change the '`--`' comments to '#' comments:

```
shell> replace " --" " #" -- text-file-with-funny-comments.sql
```

Change them back with this command:

```
shell> replace " #" " --" -- text-file-with-funny-comments.sql
```

1.5.6. How MySQL Deals with Constraints

MySQL allows you to work both with transactional tables that allow rollback and with non-transactional tables that do not. Because of this, constraint handling is a bit different in MySQL than in other databases. We must handle the case when you have inserted or updated a lot of rows in a non-transactional table for which changes cannot be rolled back when an error occurs.

The basic philosophy is that MySQL Server tries to produce an error for anything that it can detect while parsing a statement to be executed, and tries to recover from any errors that occur while executing the statement. We do this in most cases, but not yet for all. See Section C.3, “New Features Planned for the Near Future”.

The options MySQL has when an error occurs are to stop the statement in the middle or to recover as well as possible from the problem and continue. By default, the server follows the latter course. This means, for example, that the server may coerce illegal values to the closest legal values.

Beginning with MySQL 5.0.2, several SQL mode options are available to provide greater control over how accepting to be of bad data values and whether to continue executing a statement or abort it when errors occur. Using these options, you can configure MySQL Server to act in a more traditional fashion that is like other DBMSs that reject improper input. The SQL mode can be set at runtime, which enables individual clients to select the behavior most appropriate for their requirements. See Section 5.2.2, “The Server SQL Mode”.

The following sections describe what happens for the different types of constraints.

1.5.6.1. `PRIMARY KEY` and `UNIQUE` Index Constraints

Normally, an error occurs when you try to `INSERT` or `UPDATE` a row that causes a primary key, unique key, or foreign key violation. If you are using a transactional storage engine such as `InnoDB`, MySQL automatically rolls back the statement. If you are using a non-transactional storage engine, MySQL stops processing the statement at the row for which the error occurred and leaves any remaining rows unprocessed.

If you wish to ignore such key violations, MySQL supports an `IGNORE` keyword for `INSERT` and `UPDATE`. In this case, MySQL ignores any key violations and continues processing with the next row. See Section 13.1.4, “`INSERT` Syntax”. See Section 13.1.10, “`UPDATE` Syntax”.

You can get information about the number of rows actually inserted or updated with the `mysql_info()` C API function. See Section 22.2.3.31, “`mysql_info()`”. In MySQL 4.1 and up, you also can use the `SHOW WARNINGS` statement. See Section 13.5.4.20, “`SHOW WARNINGS` Syntax”.

For the moment, only `InnoDB` tables support foreign keys. See Section 15.7.4, “`FOREIGN KEY` Constraints”. Foreign key support in `MyISAM` tables is scheduled for implementation in MySQL 5.1.

1.5.6.2. Constraints on Invalid Data

Before MySQL 5.0.2, MySQL is forgiving of illegal or improper data values and coerces them to legal values for data entry. In MySQL 5.0.2 and up, that remains the default behavior, but you can select more traditional treatment of bad values such that the server rejects them and aborts the statement in which they occur. This section describes the default (forgiving) behavior of MySQL, as well as the newer strict SQL mode and how it differs.

The following holds true when you are not using strict mode. If you insert an ``incorrect'' value into a column, such as a `NULL` into a `NOT NULL` column or a too-large numeric value into a numeric column, MySQL sets the column to the ``best possible value'' instead of producing an error:

- If you try to store an out of range value in a numeric column, MySQL Server instead stores zero, the smallest possible value, or the largest possible value in the column.
- For strings, MySQL stores either the empty string or the longest possible string that can be in the column.
- If you try to store a string that doesn't start with a number into a numeric column, MySQL Server stores 0.
- If you try to store `NULL` into a column that doesn't take `NULL` values, MySQL Server stores 0 or `''` (the empty string) instead.
- MySQL allows you to store certain incorrect date values into `DATE` and `DATETIME` columns (such as `'2000-02-31'` or `'2000-02-00'`). The idea is that it's not the job of the SQL server to validate dates. If MySQL can store a date value and retrieve exactly the same value, MySQL stores it as given. If the date is totally wrong (outside the server's ability to store it), the special date value `'0000-00-00'` is stored in the column instead.
- If an `INSERT` statement specifies no value for a column, MySQL inserts its default value if the column definition includes an explicit `DEFAULT` clause. If the definition has no such `DEFAULT` clause, MySQL inserts the implicit default value for the column data type. In general, this is the empty string for string columns, 0 for numeric columns, and `'0000-00-00'` for date columns. Implicit default values are discussed in Section 13.2.6, “`CREATE TABLE` Syntax”.

The reason for the preceding rules is that we can't check these conditions until the statement has begun executing. We can't just roll back if we encounter a problem after updating a few rows, because the storage engine may not support rollback. The option of terminating the statement is not that good; in this case, the update would be ``half done,'' which is probably the worst possible scenario. In this case, it's better to ``do the best you can'' and then continue as if nothing happened.

In MySQL 5.0.2 and up, you can select stricter treatment of input values by using the `STRICT_TRANS_TABLES` or `STRICT_ALL_TABLES` SQL modes. See Section 5.2.2, “The Server SQL Mode”.

`STRICT_TRANS_TABLES` works like this: For transactional storage engines, bad data values occurring anywhere in the statement causes the to abort and roll back. For non-transactional storage engines, the statement aborts if the error occurs in the first row to be inserted or updated. (In this case, the statement can be regarded to leave the table unchanged, just as for a transactional table.) Errors in rows after the first do not abort the statement. Instead, bad data values are adjusted and result in warnings rather than errors. In other words, with `STRICT_TRANS_TABLES`, a wrong value causes MySQL to roll back, if it can, all updates done so far.

For stricter checking, enable STRICT_ALL_TABLES. This is the same as STRICT_TRANS_TABLES except that for non-transactional storage engines, errors abort the statement even for bad data in rows following the first row. This means that if an error occurs partway through a multiple-row insert or update for a non-transactional table, a partial update results. Earlier rows are inserted or updated, but those from the point of the error on are not. To avoid this for non-transactional tables, either use single-row statements or else use STRICT_TRANS_TABLES if conversion warnings rather than errors are acceptable. To avoid problems in the first place, do not use MySQL to check column content. It is safest (and often faster) to let the application ensure that it passes only legal values to the database.

With either of the strict mode options, you can cause errors to be treated as warnings by using INSERT IGNORE or UPDATE IGNORE.

1.5.6.3. ENUM and SET Constraints

ENUM and SET columns provide an efficient way to define columns that can contain only a given set of values. However, before MySQL 5.0.2, ENUM and SET are not real constraints. This is for the same reasons that NOT NULL is not honored. See Section 1.5.6.2, “Constraints on Invalid Data”.

ENUM columns always have a default value. If you don't specify a default value, then it is NULL for columns that can have NULL, otherwise the first enumeration value is used as the default value.

If you insert an incorrect value into an ENUM column or if you force a value into an ENUM column with IGNORE, it is set to the reserved enumeration value of 0, which is displayed as an empty string in string context. See Section 11.4.4, “The ENUM Type”.

If you insert an incorrect value into a SET column, the incorrect value is ignored. For example, if the column can contain the values 'a', 'b', and 'c', an attempt to assign 'a,x,b,y' results in a value of 'a,b'. See Section 11.4.5, “The SET Type”.

As of MySQL 5.0.2, you can configure the server to use strict SQL mode. See Section 5.2.2, “The Server SQL Mode”. When strict mode is not enabled, values entered into ENUM and SET columns are handled as just described for MySQL 4.x. However, if strict mode is enabled, the definition of a ENUM or SET column does act as a constraint on values entered into the column. An error occurs for values that do not satisfy these conditions:

- An ENUM value must be one of those listed in the column definition, or the internal numeric equivalent thereof. The value cannot be the error value (that is, 0 or the empty string). For a column defined as ENUM('a','b','c'), values such as '', 'd', and 'ax' are illegal and are rejected.
- A SET value must be the empty string or a value consisting of one or more of the values listed in the column definition separated by commas. For a column defined as SET('a','b','c'), values such as 'd', and 'a,b,c,d' are illegal and are rejected.

Errors for invalid values can be suppressed in strict mode if you use INSERT IGNORE or UPDATE IGNORE. In this case, a warning is generated rather than an error. For ENUM, the value is inserted as the error member (0). For SET, the value is inserted as given except that any invalid substrings are deleted. For example, 'a,x,b,y' results in a value of 'a,b', as described earlier.

1.5.7. Known Errors and Design Deficiencies in MySQL

1.5.7.1. Errors in 3.23 Fixed in a Later MySQL Version

The following known errors or bugs are not fixed in MySQL 3.23 because fixing them would involve changing a lot of code that could introduce other even worse bugs. The bugs are also classified as ``not fatal'' or ``bearable.''

- You should avoid using spaces at the end of column names because this can cause weird behavior. (Fixed in MySQL 4.0.) (Bug #4196 [http://bugs.mysql.com/4196])
- You can get a deadlock (hung thread) if you use `LOCK TABLE` to lock multiple tables and then in the same connection use `DROP TABLE` to drop one of them while another thread is trying to lock it. (To break the deadlock, you can use `KILL` to terminate any of the threads involved.) This issue is resolved as of MySQL 4.0.12.
- `SELECT MAX(key_column) FROM t1,t2,t3...` where one of the tables are empty doesn't return `NULL` but instead returns the maximum value for the column. This issue is resolved as of MySQL 4.0.11.
- `DELETE FROM heap_table` without a `WHERE` clause doesn't work on a locked `HEAP` table.

1.5.7.2. Errors in 4.0 Fixed in a Later MySQL Version

The following known errors or bugs are not fixed in MySQL 4.0 because fixing them would involve changing a lot of code that could introduce other even worse bugs. The bugs are also classified as ``not fatal'' or ``bearable.''

- In `HAVING` you can get a crash or wrong result if you use an alias to a `RAND()` function. This is fixed in 4.1.10 but will not be fixed in 4.0 because the fix 'may' cause side effects for some applications.
- In a `UNION`, the first `SELECT` determines the type, `max_length`, and `NULL` properties for the resulting columns. This issue is resolved as of MySQL 4.1.1; the property values are based on the rows from all `UNION` parts.
- In `DELETE` with many tables, you can't refer to tables to be deleted through an alias. This is fixed as of MySQL 4.1.
- You cannot mix `UNION ALL` and `UNION DISTINCT` in the same query. If you use `ALL` for one `UNION`, it is used for all of them. This is fixed as of MySQL 4.1.2. The rules for mixed `UNION` types are given in Section 13.1.7.2, “`UNION` Syntax”.
- `FLUSH TABLES WITH READ LOCK` does not block `CREATE TABLE`, which may cause a problem with the binary log position when doing a full backup of tables and the binary log.
- `mysqldump --single-transaction --master-data` behaved like `mysqldump --master-data`, so the dump was a blocking one. This is fixed starting from MySQL 4.1.8.
- When using the `RPAD()` function (or any function adding spaces to the right) in a query that had to be resolved by using a temporary table, all resulting strings had rightmost spaces removed (i.e. `RPAD()` did not work).

1.5.7.3. Open Bugs and Design Deficiencies in MySQL

The following problems are known and fixing them is a high priority:

- If you compare a `NULL` value to a subquery using `ALL/ANY/SOME` and the subquery returns an empty result, the comparison might evaluate to the non-standard result of `NULL` rather than to `TRUE` or `FALSE`. This will be fixed in MySQL 5.0.
- Subquery optimization for `IN` is not as good as for `=`.

- Even if you are using `lower_case_table_names=2` (which enables MySQL to remember the used case for databases and table names) MySQL does not remember the used case for database names for the function `DATABASE()` or in various logs (on case-insensitive systems).
- Dropping a `FOREIGN KEY` constraint doesn't work in replication because the constraint may have another name on the slave.
- `REPLACE` (and `LOAD DATA` with the `REPLACE` option) does not trigger `ON DELETE CASCADE`.
- `DISTINCT` with `ORDER BY` doesn't work inside `GROUP_CONCAT()` if you don't use all and only those columns that are in the `DISTINCT` list.
- If one user has a long-running transaction and another user drops a table that is updated in the transaction, there is small chance that the binary log may contain the `DROP TABLE` command before the table is used in the transaction itself. We plan to fix this by having the `DROP TABLE` wait until the table is not used in any transaction.
- When inserting a big integer value (between 2^63 and 2^64#1) into a decimal/string column, it is inserted as a negative value because the number is evaluated in a signed integer context.
- `FLUSH TABLES WITH READ LOCK` does not block `COMMIT` if the server is running without binary logging, which may cause a problem (of consistency between tables) when doing a full backup.
- `ANALYZE TABLE` on a `BDB` table may in some cases make the table unusable until you restart **mysqld**. If this happens, look for errors of the following form in the MySQL error file:

  ```
  001207 22:07:56  bdb:  log_flush: LSN past current end-of-log
  ```

- Don't execute `ALTER TABLE` on a `BDB` table on which you are running multiple-statement transactions until all those transactions complete. (The transaction might be ignored.)
- `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` may cause problems on tables for which you are using `INSERT DELAYED`.
- Doing a `LOCK TABLE ...` and `FLUSH TABLES ...` doesn't guarantee that there isn't a half-finished transaction in progress on the table.
- `BDB` tables are a bit slow to open. If you have many `BDB` tables in a database, it takes a long time to use the **mysql** client on the database if you are not using the `-A` option or if you are using `rehash`. This is especially notable when you have a large table cache.
- Replication uses query-level logging: The master writes the executed queries to the binary log. This is a very fast, compact, and efficient logging method that works perfectly in most cases. Still it is possible for the data on the master and slave to become different if a query is designed in such a way that the data modification is non-deterministic; that is, left to the will of the query optimizer. (That generally is not a good practice anyway, even outside of replication!) For example:

  - `CREATE ... SELECT` or `INSERT ... SELECT` statements that insert zero or `NULL` values into an `AUTO_INCREMENT` column.
  - `DELETE` if you are deleting rows from a table that has foreign keys with `ON DELETE CASCADE` properties.
  - `REPLACE ... SELECT`, `INSERT IGNORE ... SELECT` if you have duplicate key values in the inserted data.

  If and only if all these queries have no `ORDER BY` clause guaranteeing a deterministic order.

For example, for `INSERT ... SELECT` with no `ORDER BY`, the `SELECT` may return rows in a different order (which results in a row having different ranks, hence getting a different number in the `AUTO_INCREMENT` column), depending on the choices made by the optimizers on the master and slave. A query is optimized differently on the master and slave only if:

- The files used by the two queries are not exactly the same; for example, `OPTIMIZE TABLE` was run on the master tables and not on the slave tables. (To fix this, `OPTIMIZE TABLE`, `ANALYZE TABLE`, and `REPAIR TABLE` are written to the binary log as of MySQL 4.1.1).
- The table is stored using a different storage engine on the master than on the slave. (It is possible to use different storage engines on the master and slave. For example, you can use `InnoDB` on the master, but `MyISAM` on the slave if the slave has less available disk space.)
- MySQL buffer sizes (`key_buffer_size`, and so on) are different on the master and slave.
- The master and slave run different MySQL versions, and the optimizer code differs between these versions.

This problem may also affect database restoration using **mysqlbinlog|mysql**.

The easiest way to avoid this problem in all cases is to add an `ORDER BY` clause to such non-deterministic queries to ensure that the rows are always stored or modified in the same order. In future MySQL versions, we will automatically add an `ORDER BY` clause when needed.

The following problems are known and will be fixed in due time:

- Log filenames are based on the server hostname (if you don't specify a filename with the startup option). You have to use options like `--log-bin=`*`old_host_name`*`-bin` if you change your hostname to something else. Another option is to just rename the old files to reflect your hostname change (and if these are binary logs, you also need to edit the binary log index file and fix the binlog names there). See Section 5.2.1, “**mysqld** Command-Line Options”.
- **mysqlbinlog** does not delete temporary files left after a `LOAD DATA INFILE` command. See Section 8.5, “The **mysqlbinlog** Binary Log Utility”.
- `RENAME` doesn't work with `TEMPORARY` tables or tables used in a `MERGE` table.
- Due to the way table definition files are stored, you cannot use character 255 (`CHAR(255)`) in table names, column names, or enumerations. This is scheduled to be fixed in version 5.1 when we have new table definition format files.
- When using `SET CHARACTER SET`, you can't use translated characters in database, table, and column names.
- You can't use '_' or '%' with `ESCAPE` in `LIKE ... ESCAPE`.
- If you have a `DECIMAL` column in which the same number is stored in different formats (for example, `+01.00`, `1.00`, `01.00`), `GROUP BY` may regard each value as a different value.
- You cannot build the server in another directory when using MIT-pthreads. Because this requires changes to MIT-pthreads, we are not likely to fix this. See Section 2.8.5, “MIT-pthreads Notes”.
- `BLOB` and `TEXT`values can't ``reliably'' be used in `GROUP BY` or `ORDER BY` or `DISTINCT`. Only the first `max_sort_length` bytes are used when comparing `BLOB` values in these cases. The default value of `max_sort_length` value is 1024. It can be changed at server startup time. As of MySQL 4.0.3, it can also be changed at runtime. For older versions, a workaround for most cases is

to use a substring. For example:

```
SELECT DISTINCT LEFT(blob_col,2048) FROM tbl_name;
```

- Numeric calculations are done with BIGINT or DOUBLE (both are normally 64 bits long). Which precision you get depends on the function. The general rule is that bit functions are done with BIGINT precision, IF and ELT() with BIGINT or DOUBLE precision, and the rest with DOUBLE precision. You should try to avoid using unsigned long long values if they resolve to be bigger than 63 bits (9223372036854775807) for anything other than bit fields. MySQL Server 4.0 has better BIGINT handling than 3.23.

- All string columns, except BLOB and TEXT columns, automatically have all trailing spaces removed when retrieved. For CHAR types, this is okay. The bug is that in MySQL Server, VARCHAR columns are treated the same way.

- You can have only up to 255 ENUM and SET columns in one table.

- In MIN(), MAX(), and other aggregate functions, MySQL currently compares ENUM and SET columns by their string value rather than by the string's relative position in the set.

- **mysqld_safe** redirects all messages from **mysqld** to the **mysqld** log. One problem with this is that if you execute **mysqladmin refresh** to close and reopen the log, stdout and stderr are still redirected to the old log. If you use --log extensively, you should edit **mysqld_safe** to log to *host_name*.err instead of *host_name*.log so that you can easily reclaim the space for the old log by deleting the old one and executing **mysqladmin refresh**.

- In the UPDATE statement, columns are updated from left to right. If you refer to an updated column, you get the updated value instead of the original value. For example, the following statement increments KEY by 2, not 1:

  ```
  mysql> UPDATE tbl_name SET KEY=KEY+1,KEY=KEY+1;
  ```

- You can refer to multiple temporary tables in the same query, but you cannot refer to any given temporary table more than once. For example, the following doesn't work:

  ```
  mysql> SELECT * FROM temp_table, temp_table AS t2;
  ERROR 1137: Can't reopen table: 'temp_table'
  ```

- The optimizer may handle DISTINCT differently when you are using ``hidden'' columns in a join than when you are not. In a join, hidden columns are counted as part of the result (even if they are not shown), whereas in normal queries, hidden columns don't participate in the DISTINCT comparison. We will probably change this in the future to never compare the hidden columns when executing DISTINCT.

  An example of this is:

  ```
  SELECT DISTINCT mp3id FROM band_downloads
         WHERE userid = 9 ORDER BY id DESC;
  ```

  and

  ```
  SELECT DISTINCT band_downloads.mp3id
         FROM band_downloads,band_mp3
         WHERE band_downloads.userid = 9
         AND band_mp3.id = band_downloads.mp3id
         ORDER BY band_downloads.id DESC;
  ```

In the second case, you might in MySQL Server 3.23.x get two identical rows in the result set (because the values in the hidden `id` column may differ).

Note that this happens only for queries where you don't have the `ORDER BY` columns in the result.

- Because MySQL Server allows you to work with table types that don't support transactions, and thus can't roll back data, some things behave a little differently in MySQL Server than in other SQL servers. This is just to ensure that MySQL Server never needs to do a rollback for an SQL statement. This may be a little awkward at times because column values must be checked in the application, but this actually gives you a nice speed increase because it allows MySQL Server to do some optimizations that otherwise would be very hard to do.

  If you set a column to an incorrect value, MySQL Server, instead of doing a rollback, stores the ``best possible value'' in the column. For information about how this occurs, see Section 1.5.6, “How MySQL Deals with Constraints”.

- If you execute a `PROCEDURE` on a query that returns an empty set, in some cases the `PROCEDURE` does not transform the columns.

- Creation of a table of type `MERGE` doesn't check whether the underlying tables are of compatible types.

- If you use `ALTER TABLE` first to add a `UNIQUE` index to a table used in a `MERGE` table and then to add a normal index on the `MERGE` table, the key order is different for the tables if there was an old key that was not unique in the table. This is because `ALTER TABLE` puts `UNIQUE` indexes before normal indexes to be able to detect duplicate keys as early as possible.

The following are known bugs in earlier versions of MySQL:

- In the following case you can get a core dump:

  - Delayed insert handler has pending inserts to a table.

  - `LOCK TABLE` with `WRITE`.

  - `FLUSH TABLES`.

- Before MySQL Server 3.23.2, an `UPDATE` that updated a key with a `WHERE` on the same key may have failed because the key was used to search for records and the same row may have been found multiple times:

  ```
  UPDATE tbl_name SET KEY=KEY+1 WHERE KEY > 100;
  ```

  A workaround is to use:

  ```
  UPDATE tbl_name SET KEY=KEY+1 WHERE KEY+0 > 100;
  ```

  This works because MySQL Server does not use an index on expressions in the `WHERE` clause.

- Before MySQL Server 3.23, all numeric types were treated as fixed-point fields. That means that you had to specify how many decimals a floating-point field should have. All results were returned with the correct number of decimals.

For information about platform-specific bugs, see the installation and porting instructions in Section 2.12, “Operating System-Specific Notes” and Appendix E, *Porting to Other Systems*.

Chapter 2. Installing MySQL

This chapter describes how to obtain and install MySQL:

1. **Determine whether your platform is supported.** Please note that not all supported systems are equally good for running MySQL on them. On some it is much more robust and efficient than others. See Section 2.1.1, “Operating Systems Supported by MySQL” for details.

2. **Choose which distribution to install.** Several versions of MySQL are available, and most are available in several distribution formats. You can choose from pre-packaged distributions containing binary (precompiled) programs or source code. When in doubt, use a binary distribution. We also provide public access to our current source tree for those who want to see our most recent developments and help us test new code. To determine which version and type of distribution you should use, see Section 2.1.2, “Choosing Which MySQL Distribution to Install”.

3. **Download the distribution that you want to install.** For a list of sites from which you can obtain MySQL, see Section 2.1.3, “How to Get MySQL”. You can verify the integrity of the distribution using the instructions in Section 2.1.4, “Verifying Package Integrity Using MD5 Checksums or `GnuPG`”.

4. **Install the distribution.** To install MySQL from a binary distribution, use the instructions in Section 2.2, “Standard MySQL Installation Using a Binary Distribution”. To install MySQL from a source distribution or from the current development source tree, use the instructions in Section 2.8, “MySQL Installation Using a Source Distribution”.

   Note: If you plan to upgrade an existing version of MySQL to a newer version rather than installing MySQL for the first time, see Section 2.10, “Upgrading MySQL” for information about upgrade procedures and about issues that you should consider before upgrading.

   If you encounter installation difficulties, see Section 2.12, “Operating System-Specific Notes” for information on solving problems for particular platforms.

5. **Perform any necessary post-installation setup.** After installing MySQL, read Section 2.9, “Post-Installation Setup and Testing”. This section contains important information about making sure the MySQL server is working properly. It also describes how to secure the initial MySQL user accounts, *which have no passwords* until you assign passwords. The section applies whether you install MySQL using a binary or source distribution.

6. If you want to run the MySQL benchmark scripts, Perl support for MySQL must be available. See Section 2.13, “Perl Installation Notes”.

2.1. General Installation Issues

Before installing MySQL, you should do the following:

1. Determine whether or not MySQL runs on your platform.

2. Choose a distribution to install.

3. Download the distribution and verify its integrity.

This section contains the information necessary to carry out these steps. After doing so, you can use the

instructions in later sections of the chapter to install the distribution that you choose.

2.1.1. Operating Systems Supported by MySQL

This section lists the operating systems on which you can expect to be able to run MySQL.

We use GNU Autoconf, so it is possible to port MySQL to all modern systems that have a C++ compiler and a working implementation of POSIX threads. (Thread support is needed for the server. To compile only the client code, the only requirement is a C++ compiler.) We use and develop the software ourselves primarily on Linux (SuSE and Red Hat), FreeBSD, and Sun Solaris (Versions 8 and 9).

MySQL has been reported to compile successfully on the following combinations of operating system and thread package. Note that for many operating systems, native thread support works only in the latest versions.

- AIX 4.x, 5.x with native threads. See Section 2.12.5.3, “IBM-AIX notes”.
- Amiga.
- BSDI 2.x with the MIT-pthreads package. See Section 2.12.4.5, “BSD/OS Version 2.x Notes”.
- BSDI 3.0, 3.1 and 4.x with native threads. See Section 2.12.4.5, “BSD/OS Version 2.x Notes”.
- Digital Unix 4.x with native threads. See Section 2.12.5.5, “Alpha-DEC-UNIX Notes (Tru64)”.
- FreeBSD 2.x with the MIT-pthreads package. See Section 2.12.4.1, “FreeBSD Notes”.
- FreeBSD 3.x and 4.x with native threads. See Section 2.12.4.1, “FreeBSD Notes”.
- FreeBSD 4.x with LinuxThreads. See Section 2.12.4.1, “FreeBSD Notes”.
- HP-UX 10.20 with the DCE threads or the MIT-pthreads package. See Section 2.12.5.1, “HP-UX Version 10.20 Notes”.
- HP-UX 11.x with the native threads. See Section 2.12.5.2, “HP-UX Version 11.x Notes”.
- Linux 2.0+ with LinuxThreads 0.7.1+ or `glibc` 2.0.7+ for various CPU architectures. See Section 2.12.1, “Linux Notes”.
- Mac OS X. See Section 2.12.2, “Mac OS X Notes”.
- NetBSD 1.3/1.4 Intel and NetBSD 1.3 Alpha (requires GNU make). See Section 2.12.4.2, “NetBSD Notes”.
- Novell NetWare 6.0. See Section 2.6, “Installing MySQL on NetWare”.
- OpenBSD > 2.5 with native threads. OpenBSD < 2.5 with the MIT-pthreads package. See Section 2.12.4.3, “OpenBSD 2.5 Notes”.
- OS/2 Warp 3, FixPack 29 and OS/2 Warp 4, FixPack 4. See Section 2.12.6, “OS/2 Notes”.
- SCO OpenServer with a recent port of the FSU Pthreads package. See Section 2.12.5.8, “SCO Notes”.
- SCO UnixWare 7.1.x. See Section 2.12.5.9, “SCO UnixWare Version 7.1.x Notes”.
- SGI Irix 6.x with native threads. See Section 2.12.5.7, “SGI Irix Notes”.

- Solaris 2.5 and above with native threads on SPARC and x86. See Section 2.12.3, “Solaris Notes”.
- SunOS 4.x with the MIT-pthreads package. See Section 2.12.3, “Solaris Notes”.
- Tru64 Unix. See Section 2.12.5.5, “Alpha-DEC-UNIX Notes (Tru64)”.
- Windows 9x, Me, NT, 2000, XP, and 2003. See Section 2.3, “Installing MySQL on Windows”.

Not all platforms are equally well-suited for running MySQL. How well a certain platform is suited for a high-load mission-critical MySQL server is determined by the following factors:

- General stability of the thread library. A platform may have an excellent reputation otherwise, but MySQL is only as stable as the thread library it calls, even if everything else is perfect.
- The capability of the kernel and the thread library to take advantage of symmetric multi-processor (SMP) systems. In other words, when a process creates a thread, it should be possible for that thread to run on a different CPU than the original process.
- The capability of the kernel and the thread library to run many threads that acquire and release a mutex over a short critical region frequently without excessive context switches. If the implementation of `pthread_mutex_lock()` is too anxious to yield CPU time, this hurts MySQL tremendously. If this issue is not taken care of, adding extra CPUs actually makes MySQL slower.
- General filesystem stability and performance.
- If your tables are big, the ability of the filesystem to deal with large files at all and to deal with them efficiently.
- Our level of expertise here at MySQL AB with the platform. If we know a platform well, we enable platform-specific optimizations and fixes at compile time. We can also provide advice on configuring your system optimally for MySQL.
- The amount of testing we have done internally for similar configurations.
- The number of users that have successfully run MySQL on the platform in similar configurations. If this number is high, the chances of encountering platform-specific surprises are much smaller.

Based on the preceding criteria, the best platforms for running MySQL at this point are x86 with SuSE Linux using a 2.4 kernel, and ReiserFS (or any similar Linux distribution) and SPARC with Solaris (2.7-9). FreeBSD comes third, but we really hope it joins the top club once the thread library is improved. We also hope that at some point we is able to include into the top category all other platforms on which MySQL currently compiles and runs okay, but not quite with the same level of stability and performance. This requires some effort on our part in cooperation with the developers of the operating system and library components that MySQL depends on. If you are interested in improving one of those components, are in a position to influence its development, and need more detailed instructions on what MySQL needs to run better, send an email message to the MySQL `internals` mailing list. See Section 1.4.1.1, “The MySQL Mailing Lists”.

Please note that the purpose of the preceding comparison is not to say that one operating system is better or worse than another in general. We are talking only about choosing an OS for the specific purpose of running MySQL. With this in mind, the result of this comparison would be different if we considered more factors. In some cases, the reason one OS is better than the other could simply be that we have been able to put more effort into testing and optimizing for a particular platform. We are just stating our observations to help you decide which platform to use for running MySQL.

2.1.2. Choosing Which MySQL Distribution to Install

When preparing to install MySQL, you should decide which version to use. MySQL development occurs in several release series, and you can pick the one that best fits your needs. After deciding which version to install, you can choose a distribution format. Releases are available in binary or source format.

2.1.2.1. Choosing Which Version of MySQL to Install

The first decision to make is whether you want to use a production (stable) release or a development release. In the MySQL development process, multiple release series co-exist, each at a different stage of maturity:

- MySQL 5.0 is the newest development release series and is under very active development for new features. Alpha releases have been issued to allow more widespread testing.
- MySQL 4.1 is the current stable (production-quality) release series. New releases are issued for bugfixes. No new features are added that could diminish the code stability.
- MySQL 4.0 is the previous stable (production-quality) release series. New releases are issued for bugfixes. No new features are added that could diminish the code stability.
- MySQL 3.23 is the old stable (production-quality) release series. This series is retired, so new releases are issued only to fix critical bugs.

We don't believe in a complete freeze, as this also leaves out bugfixes and things that ``must be done.'' ``Somewhat frozen'' means that we may add small things that ``almost surely do not affect anything that's currently working.'' Naturally, relevant bugfixes from an earlier series propagate to later series.

Normally, if you are beginning to use MySQL for the first time or trying to port it to some system for which there is no binary distribution, we recommend going with the production release series. Currently this is MySQL 4.1. All MySQL releases, even those from development series, are checked with the MySQL benchmarks and an extensive test suite before being issued.

If you are running an old system and want to upgrade, but don't want to take the chance of having a non-seamless upgrade, you should upgrade to the latest version in the same release series you are using (where only the last part of the version number is newer than yours). We have tried to fix only fatal bugs and make small, relatively safe changes to that version.

If you want to use new features not present in the production release series, you can use a version from a development series. Note that development releases are not as stable as production releases.

If you want to use the very latest sources containing all current patches and bugfixes, you can use one of our BitKeeper repositories. These are not ``releases'' as such, but are available as previews of the code on which future releases are based.

The MySQL naming scheme uses release names that consist of three numbers and a suffix; for example, `mysql-4.1.2-alpha`. The numbers within the release name are interpreted like this:

- The first number (`4`) is the major version and also describes the file format. All Version 4 releases have the same file format.
- The second number (`1`) is the release level. Taken together, the major version and release level constitute the release series number.
- The third number (`2`) is the version number within the release series. This is incremented for each

new release. Usually you want the latest version for the series you have chosen.

For each minor update, the last number in the version string is incremented. When there are major new features or minor incompatibilities with previous versions, the second number in the version string is incremented. When the file format changes, the first number is increased.

Release names also include a suffix to indicates the stability level of the release. Releases within a series progress through a set of suffixes to indicate how the stability level improves. The possible suffixes are:

- `alpha` indicates that the release contains some large section of new code that hasn't been 100% tested. Known bugs (usually there are none) should be documented in the News section. See Appendix D, *MySQL Change History*. There are also new commands and extensions in most alpha releases. Active development that may involve major code changes can occur in an alpha release, but everything is tested before issuing a release. For this reason, there should be no known bugs in any MySQL release.
- `beta` means that all new code has been tested. No major new features that could cause corruption in old code are added. There should be no known bugs. A version changes from alpha to beta when there haven't been any reported fatal bugs within an alpha version for at least a month and we have no plans to add any features that could make any old command unreliable.
- `gamma` is a beta that has been around a while and seems to work fine. Only minor fixes are added. This is what many other companies call a release.
- If there is no suffix, it means that the version has been run for a while at many different sites with no reports of bugs other than platform-specific bugs. Only critical bugfixes are applied to the release. This is what we call a production (stable) or `General Availability' (GA) release.

MySQL uses a naming scheme that is slightly different from most other products. In general, it's relatively safe to use any version that has been out for a couple of weeks without being replaced with a new version within the release series.

All releases of MySQL are run through our standard tests and benchmarks to ensure that they are relatively safe to use. Because the standard tests are extended over time to check for all previously found bugs, the test suite keeps getting better.

All releases have been tested at least with:

- An internal test suite

  The `mysql-test` directory contains an extensive set of test cases. We run these tests for virtually every server binary. See Section 25.1.2, “MySQL Test Suite” for more information about this test suite.

- The MySQL benchmark suite

  This suite runs a range of common queries. It is also a test to see whether the latest batch of optimizations actually made the code faster. See Section 7.1.4, “The MySQL Benchmark Suite”.

- The `crash-me` test

  This test tries to determine what features the database supports and what its capabilities and limitations are. See Section 7.1.4, “The MySQL Benchmark Suite”.

Another test is that we use the newest MySQL version in our internal production environment, on at least one machine. We have more than 100GB of data to work with.

2.1.2.2. Choosing a Distribution Format

After choosing which version of MySQL to install, you should decide whether to use a binary distribution or a source distribution. In most cases, you should probably use a binary distribution, if one exists for your platform. Binary distributions are available in native format for many platforms, such as RPM files for Linux or DMG package installers for Mac OS X. Distributions also are available as Zip archives or compressed **tar** files.

Reasons to choose a binary distribution include the following:

- Binary distributions generally are easier to install than source distributions.
- To satisfy different user requirements, we provide two different binary versions: one compiled with the non-transactional storage engines (a small, fast binary), and one configured with the most important extended options like transaction-safe tables. Both versions are compiled from the same source distribution. All native MySQL clients can connect to servers from either MySQL version.

  The extended MySQL binary distribution is marked with the `-max` suffix and is configured with the same options as **mysqld-max**. See Section 5.1.2, “The **mysqld-max** Extended MySQL Server”.

  If you want to use the `MySQL-Max` RPM, you must first install the standard `MySQL-server` RPM.

Under some circumstances, you may be better off installing MySQL from a source distribution:

- You want to install MySQL at some explicit location. The standard binary distributions are ``ready to run'' at any place, but you may want to have even more flexibility to place MySQL components where you want.
- You want to configure **mysqld** with some extra features that are not included in the standard binary distributions. Here is a list of the most common extra options that you may want to use:
  - `--with-innodb` (default for MySQL 4.0 and up)
  - `--with-berkeley-db` (not available on all platforms)
  - `--with-raid`
  - `--with-libwrap`
  - `--with-named-z-libs` (this is done for some of the binaries)
  - `--with-debug[=full]`
- You want to configure **mysqld** without some features that are included in the standard binary distributions. For example, distributions normally are compiled with support for all character sets. If you want a smaller MySQL server, you can recompile it with support for only the character sets you need.
- You have a special compiler (such as `pgcc`) or want to use compiler options that are better optimized for your processor. Binary distributions are compiled with options that should work on a variety of processors from the same processor family.

- You want to use the latest sources from one of the BitKeeper repositories to have access to all current bugfixes. For example, if you have found a bug and reported it to the MySQL development team, the bugfix is committed to the source repository and you can access it there. The bugfix does not appear in a release until a release actually is issued.
- You want to read (or modify) the C and C++ code that makes up MySQL. For this purpose, you should get a source distribution, because the source code is always the ultimate manual.
- Source distributions contain more tests and examples than binary distributions.

2.1.2.3. How and When Updates Are Released

MySQL is evolving quite rapidly here at MySQL AB and we want to share new developments with other MySQL users. We try to make a release when we have very useful features that others seem to have a need for.

We also try to help out users who request features that are easy to implement. We take note of what our licensed users want to have, and we especially take note of what our support customers want and try to help them out.

No one has to download a new release. The News section tells you if the new release has something you really want. See Appendix D, *MySQL Change History*.

We use the following policy when updating MySQL:

- Releases are issued within each series. For each release, the last number in the version is one more than the previous release within the same series.
- Production (stable) releases are meant to appear about 1-2 times a year. However, if small bugs are found, a release with only bugfixes is issued.
- Working releases/bugfixes to old releases are meant to appear about every 4-8 weeks.
- Binary distributions for some platforms are made by us for major releases. Other people may make binary distributions for other systems, but probably less frequently.
- We make fixes available as soon as we have identified and corrected small or non-critical but annoying bugs. The fixes are available immediately from our public BitKeeper repositories, and will be included in the next release.
- If by any chance a fatal bug is found in a release, we make a new release as soon as possible. (We would like other companies to do this, too!)

2.1.2.4. Release Philosophy---No Known Bugs in Releases

We put a lot of time and effort into making our releases bug-free. To our knowledge, we have not knowingly released a single MySQL version with any *known* ``fatal'' repeatable bugs. (A ``fatal'' bug is something that crashes MySQL under normal usage, produces incorrect answers for normal queries, or has a security problem.)

We have documented all open problems, bugs, and issues that are dependent on design decisions. See Section 1.5.7, “Known Errors and Design Deficiencies in MySQL”.

Our aim is to fix everything that is fixable without risk of making a stable MySQL version less stable. In certain cases, this means we can fix an issue in the development versions, but not in the stable

(production) version. Naturally, we document such issues so that users are aware of them.

Here is a description of how our build process works:

- We monitor bugs from our customer support list, the bugs database at http://bugs.mysql.com/, and the MySQL external mailing lists.
- All reported bugs for live versions are entered into the bugs database.
- When we fix a bug, we always try to make a test case for it and include it into our test system to ensure that the bug can never recur without being detected. (About 90% of all fixed bugs have a test case.)
- We create test cases for all new features we add to MySQL.
- Before we start to build a new MySQL release, we ensure that all reported repeatable bugs for the MySQL version (3.23.x, 4.0.x, etc) are fixed. If something is impossible to fix (due to some internal design decision in MySQL), we document this in the manual. See Section 1.5.7, “Known Errors and Design Deficiencies in MySQL”.
- We do a build on all platforms for which we support binaries (15+ platforms) and run our test suite and benchmark suite on all of them.
- We do not publish a binary for a platform for which the test or benchmark suite fails. If the problem is due to a general error in the source, we fix it and do the build plus tests on all systems again from scratch.
- The build and test process takes 2-3 days. If we receive a report regarding a fatal bug during this process (for example, one that causes a core dump), we fix the problem and restart the build process.
- After publishing the binaries on http://dev.mysql.com/, we send out an announcement message to the `mysql` and `announce` mailing lists. See Section 1.4.1.1, “The MySQL Mailing Lists”. The announcement message contains a list of all changes to the release and any known problems with the release. The Known Problems section in the release notes has been needed for only a handful of releases.
- To quickly give our users access to the latest MySQL features, we do a new MySQL release every 4-8 weeks. Source code snapshots are built daily and are available at http://downloads.mysql.com/snapshots.php.
- If, despite our best efforts, we get any bug reports after the release is done that there was something critically wrong with the build on a specific platform, we fix it at once and build a new `'a'` release for that platform. Thanks to our large user base, problems are found quickly.
- Our track record for making stable releases is quite good. In the last 150 releases, we had to do a new build for fewer than 10 releases. In three of these cases, the bug was a faulty `glibc` library on one of our build machines that took us a long time to track down.

2.1.2.5. MySQL Binaries Compiled by MySQL AB

As a service of MySQL AB, we provide a set of binary distributions of MySQL that are compiled on systems at our site or on systems where supporters of MySQL kindly have given us access to their machines.

In addition to the binaries provided in platform-specific package formats, we offer binary distributions for a number of platforms in the form of compressed **tar** files (`.tar.gz` files). See Section 2.2, “Standard MySQL Installation Using a Binary Distribution”.

For Windows distributions, see Section 2.3, “Installing MySQL on Windows”.

These distributions are generated using the script `Build-tools/Do-compile`, which compiles the source code and creates the binary `tar.gz` archive using **scripts/make_binary_distribution**.

These binaries are configured and built with the following compilers and options. This information can also be obtained by looking at the variables `COMP_ENV_INFO` and `CONFIGURE_LINE` inside the script **bin/mysqlbug** of every binary **tar** file distribution.

The following binaries are built on MySQL AB development systems:

- Linux 2.4.xx x86 with **gcc** 2.95.3:

  ```
  CFLAGS="-O2 -mcpu=pentiumpro" CXX=gcc CXXFLAGS="-O2 -mcpu=pentiumpro -felide-constructors" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --enable-assembler --disable-shared --with-client-ldflags=-all-static --with-mysqld-ldflags=-all-static
  ```

- Linux 2.4.x x86 with **icc** (Intel C++ Compiler 8.0):

  ```
  CC=icc CXX=icc CFLAGS="-O3 -unroll2 -ip -mp -no-gcc -restrict" CXXFLAGS="-O3 -unroll2 -ip -mp -no-gcc -restrict" ./configure --prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data --libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --enable-assembler --disable-shared --with-client-ldflags=-all-static --with-mysqld-ldflags=-all-static --with-embedded-server --with-innodb
  ```

- Linux 2.4.xx Intel Itanium 2 with **ecc** (Intel C++ Itanium Compiler 7.0):

  ```
  CC=ecc CFLAGS="-O2 -tpp2 -ip -nolib_inline" CXX=ecc CXXFLAGS="-O2 -tpp2 -ip -nolib_inline" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile
  ```

- Linux 2.4.xx Intel Itanium with **ecc** (Intel C++ Itanium Compiler 7.0):

  ```
  CC=ecc CFLAGS=-tpp1 CXX=ecc CXXFLAGS=-tpp1 ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile
  ```

- Linux 2.4.xx alpha with `ccc` (Compaq C V6.2-505 / Compaq C++ V6.3-006):

  ```
  CC=ccc CFLAGS="-fast -arch generic" CXX=cxx CXXFLAGS="-fast -arch generic -noexceptions -nortti" ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex --enable-thread-safe-client --enable-local-infile --with-mysqld-ldflags=-non_shared --with-client-ldflags=-non_shared --disable-shared
  ```

- Linux 2.x.xx ppc with **gcc** 2.95.4:

  ```
  CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors -fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql -
  ```

```
-localstatedir=/usr/local/mysql/data                                -
-libexecdir=/usr/local/mysql/bin   --with-extra-charsets=complex   -
-enable-thread-safe-client --enable-local-infile --disable-shared -
-with-embedded-server --with-innodb
```

- Linux 2.4.xx s390 with **gcc** 2.95.3:

```
CFLAGS="-O2"     CXX=gcc     CXXFLAGS="-O2     -felide-constructors"
./configure --prefix=/usr/local/mysql --with-extra-charsets=complex
--enable-thread-safe-client  --enable-local-infile  --disable-shared
--with-client-ldflags=-all-static --with-mysqld-ldflags=-all-static
```

- Linux 2.4.xx x86_64 (AMD64) with **gcc** 3.2.1:

```
CXX=gcc        ./configure        --prefix=/usr/local/mysql        -
-with-extra-charsets=complex      --enable-thread-safe-client      -
-enable-local-infile --disable-shared
```

- Sun Solaris 8 x86 with **gcc** 3.2.3:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -
fno-omit-frame-pointer   -felide-constructors   -fno-exceptions   -
fno-rtti"       ./configure       --prefix=/usr/local/mysql       -
-localstatedir=/usr/local/mysql/data                                -
-libexecdir=/usr/local/mysql/bin   --with-extra-charsets=complex   -
-enable-thread-safe-client --enable-local-infile --disable-shared -
-with-innodb
```

- Sun Solaris 8 SPARC with **gcc** 3.2:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -
fno-omit-frame-pointer   -felide-constructors   -fno-exceptions   -
fno-rtti"       ./configure       --prefix=/usr/local/mysql       -
-with-extra-charsets=complex      --enable-thread-safe-client      -
-enable-local-infile  --enable-assembler  --with-named-z-libs=no  -
-with-named-curses-libs=-lcurses --disable-shared
```

- Sun Solaris 8 SPARC 64-bit with **gcc** 3.2:

```
CC=gcc    CFLAGS="-O3    -m64    -fno-omit-frame-pointer"    CXX=gcc
CXXFLAGS="-O3  -m64  -fno-omit-frame-pointer  -felide-constructors  -
fno-exceptions  -fno-rtti"  ./configure  --prefix=/usr/local/mysql  -
-with-extra-charsets=complex      --enable-thread-safe-client      -
-enable-local-infile            --with-named-z-libs=no            -
-with-named-curses-libs=-lcurses --disable-shared
```

- Sun Solaris 9 SPARC with **gcc** 2.95.3:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -
fno-omit-frame-pointer   -felide-constructors   -fno-exceptions   -
fno-rtti"       ./configure       --prefix=/usr/local/mysql       -
-with-extra-charsets=complex      --enable-thread-safe-client      -
-enable-local-infile              --enable-assembler               -
-with-named-curses-libs=-lcurses --disable-shared
```

- Sun Solaris 9 SPARC with `cc-5.0` (Sun Forte 5.0):

```
CC=cc-5.0  CXX=CC  ASFLAGS="-xarch=v9"  CFLAGS="-Xa  -xstrconst  -mt  -
```

```
D_FORTEC_ -xarch=v9" CXXFLAGS="-noex -mt -D_FORTEC_ -xarch=v9"
./configure --prefix=/usr/local/mysql --with-extra-charsets=complex
--enable-thread-safe-client --enable-local-infile -
-enable-assembler --with-named-z-libs=no -
-enable-thread-safe-client --disable-shared
```

- IBM AIX 4.3.2 ppc with **gcc** 3.2.3:

```
CFLAGS="-O2 -mcpu=powerpc -Wa,-many " CXX=gcc CXXFLAGS="-O2 -
mcpu=powerpc -Wa,-many -felide-constructors -fno-exceptions -
fno-rtti" ./configure --prefix=/usr/local/mysql -
-with-extra-charsets=complex --enable-thread-safe-client -
-enable-local-infile --with-named-z-libs=no --disable-shared
```

- IBM AIX 4.3.3 ppc with `xlC_r` (IBM Visual Age C/C++ 6.0):

```
CC=xlc_r CFLAGS="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192"
CXX=xlC_r CXXFLAGS ="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192"
./configure --prefix=/usr/local/mysql -
-localstatedir=/usr/local/mysql/data -
-libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex -
-enable-thread-safe-client --enable-local-infile -
-with-named-z-libs=no --disable-shared --with-innodb
```

- IBM AIX 5.1.0 ppc with **gcc** 3.3:

```
CFLAGS="-O2 -mcpu=powerpc -Wa,-many" CXX=gcc CXXFLAGS="-O2 -
mcpu=powerpc -Wa,-many -felide-constructors -fno-exceptions -
fno-rtti" ./configure --prefix=/usr/local/mysql -
-with-extra-charsets=complex --enable-thread-safe-client -
-enable-local-infile --with-named-z-libs=no --disable-shared
```

- IBM AIX 5.2.0 ppc with `xlC_r` (IBM Visual Age C/C++ 6.0):

```
CC=xlc_r CFLAGS="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192"
CXX=xlC_r CXXFLAGS="-ma -O2 -qstrict -qoptimize=2 -qmaxmem=8192"
./configure --prefix=/usr/local/mysql -
-localstatedir=/usr/local/mysql/data -
-libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex -
-enable-thread-safe-client --enable-local-infile -
-with-named-z-libs=no --disable-shared --with-embedded-server -
-with-innodb
```

- HP-UX 10.20 pa-risc1.1 with **gcc** 3.1:

```
CFLAGS="-DHPUX -I/opt/dce/include -O3 -fPIC" CXX=gcc
CXXFLAGS="-DHPUX -I/opt/dce /include -felide-constructors -
fno-exceptions -fno-rtti -O3 -fPIC" ./configure -
-prefix=/usr/local/mysql --with-extra-charsets=complex -
-enable-thread-safe-client --enable-local-infile --with-pthread -
-with-named-thread-libs=-ldce --with-lib-ccflags=-fPIC -
-disable-shared
```

- HP-UX 11.00 pa-risc with `aCC` (HP ANSI C++ B3910B A.03.50):

```
CC=cc CXX=aCC CFLAGS=+DAportable CXXFLAGS=+DAportable ./configure -
-prefix=/usr/local/mysql --localstatedir=/usr/local/mysql/data -
-libexecdir=/usr/local/mysql/bin --with-extra-charsets=complex -
```

```
-enable-thread-safe-client --enable-local-infile --disable-shared -
-with-embedded-server --with-innodb
```

- HP-UX 11.11 pa-risc2.0 64bit with `aCC` (HP ANSI C++ B3910B A.03.33):

```
CC=cc    CXX=aCC    CFLAGS=+DD64    CXXFLAGS=+DD64    ./configure    -
-prefix=/usr/local/mysql       --with-extra-charsets=complex       -
-enable-thread-safe-client --enable-local-infile --disable-shared
```

- HP-UX 11.11 pa-risc2.0 32bit with `aCC` (HP ANSI C++ B3910B A.03.33):

```
CC=cc     CXX=aCC     CFLAGS="+DAportable"     CXXFLAGS="+DAportable"
./configure                --prefix=/usr/local/mysql                -
-localstatedir=/usr/local/mysql/data                                -
-libexecdir=/usr/local/mysql/bin   --with-extra-charsets=complex   -
-enable-thread-safe-client --enable-local-infile --disable-shared -
-with-innodb
```

- HP-UX 11.22 ia64 64bit with `aCC` (HP aC++/ANSI C B3910B A.05.50):

```
CC=cc    CXX=aCC    CFLAGS="+DD64    +DSitanium2"    CXXFLAGS="+DD64
+DSitanium2"      ./configure      --prefix=/usr/local/mysql      -
-localstatedir=/usr/local/mysql/data                              -
-libexecdir=/usr/local/mysql/bin   --with-extra-charsets=complex   -
-enable-thread-safe-client --enable-local-infile --disable-shared -
-with-embedded-server --with-innodb
```

- Apple Mac OS X 10.2 powerpc with **gcc** 3.1:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -
fno-omit-frame-pointer   -felide-constructors   -fno-exceptions   -
fno-rtti"       ./configure       --prefix=/usr/local/mysql       -
-with-extra-charsets=complex      --enable-thread-safe-client      -
-enable-local-infile --disable-shared
```

- FreeBSD 4.7 i386 with **gcc** 2.95.4:

```
CFLAGS=-DHAVE_BROKEN_REALPATH ./configure --prefix=/usr/local/mysql
--with-extra-charsets=complex     --enable-thread-safe-client     -
-enable-local-infile              --enable-assembler              -
-with-named-z-libs=not-used --disable-shared
```

- FreeBSD 4.7 i386 using LinuxThreads with **gcc** 2.95.4:

```
CFLAGS="-DHAVE_BROKEN_REALPATH    -D__USE_UNIX98    -D_REENTRANT    -
D_THREAD_SAFE             -I/usr/local/include/pthread/linuxthreads"
CXXFLAGS="-DHAVE_BROKEN_REALPATH   -D__USE_UNIX98   -D_REENTRANT   -
D_THREAD_SAFE             -I/usr/local/include/pthread/linuxthreads"
./configure                --prefix=/usr/local/mysql                -
-localstatedir=/usr/local/mysql/data                                -
-libexecdir=/usr/local/mysql/bin    --enable-thread-safe-client    -
-enable-local-infile               --enable-assembler               -
-with-named-thread-libs="-DHAVE_GLIBC2_STYLE_GETHOSTBYNAME_R       -
D_THREAD_SAFE    -I    /usr/local/include/pthread/linuxthreads    -
L/usr/local/lib    -llthread    -llgcc_r"    --disable-shared    -
-with-embedded-server --with-innodb
```

- QNX Neutrino 6.2.1 i386 with **gcc** 2.95.3qnx-nto 20010315:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -
fno-omit-frame-pointer   -felide-constructors   -fno-exceptions   -
fno-rtti"       ./configure       --prefix=/usr/local/mysql       -
-with-extra-charsets=complex      --enable-thread-safe-client      -
-enable-local-infile --disable-shared
```

The following binaries are built on third-party systems kindly provided to MySQL AB by other users. These are provided only as a courtesy; MySQL AB does not have full control over these systems, so we can provide only limited support for the binaries built on them.

- SCO Unix 3.2v5.0.6 i386 with **gcc** 2.95.3:

  ```
  CFLAGS="-O3  -mpentium"  LDFLAGS=-static  CXX=gcc  CXXFLAGS="-O3  -
  mpentium         -felide-constructors"         ./configure         -
  -prefix=/usr/local/mysql       --with-extra-charsets=complex       -
  -enable-thread-safe-client          --enable-local-infile          -
  -with-named-z-libs=no --enable-thread-safe-client --disable-shared
  ```

- SCO OpenUnix 8.0.0 i386 with **CC** 3.2:

  ```
  CC=cc  CFLAGS="-O"  CXX=CC  ./configure  --prefix=/usr/local/mysql  -
  -with-extra-charsets=complex      --enable-thread-safe-client      -
  -enable-local-infile            --with-named-z-libs=no            -
  -enable-thread-safe-client --disable-shared
  ```

- Compaq Tru64 OSF/1 V5.1 732 alpha with `cc/cxx` (Compaq C V6.3-029i / DIGITAL C++ V6.1-027):

  ```
  CC="cc -pthread" CFLAGS="-O4 -ansi_alias -ansi_args -fast -inline
  speed -speculate all" CXX="cxx -pthread" CXXFLAGS="-O4 -ansi_alias
  -fast   -inline   speed   -speculate   all   -noexceptions   -nortti"
  ./configure --prefix=/usr/local/mysql --with-extra-charsets=complex
  --enable-thread-safe-client         --enable-local-infile         -
  -with-named-thread-libs="-lpthread    -lmach    -lexc    -lc"    -
  -disable-shared --with-mysqld-ldflags=-all-static
  ```

- SGI Irix 6.5 IP32 with **gcc** 3.0.1:

  ```
  CC=gcc   CFLAGS="-O3   -fno-omit-frame-pointer"   CXXFLAGS="-O3   -
  fno-omit-frame-pointer   -felide-constructors   -fno-exceptions   -
  fno-rtti"       ./configure       --prefix=/usr/local/mysql       -
  -with-extra-charsets=complex      --enable-thread-safe-client      -
  -enable-local-infile --disable-shared
  ```

- FreeBSD/sparc64 5.0 with **gcc** 3.2.1:

  ```
  CFLAGS=-DHAVE_BROKEN_REALPATH ./configure --prefix=/usr/local/mysql
  --localstatedir=/usr/local/mysql/data                              -
  -libexecdir=/usr/local/mysql/bin   --with-extra-charsets=complex   -
  -enable-thread-safe-client --enable-local-infile --disable-shared -
  -with-innodb
  ```

The following compile options have been used for binary packages that MySQL AB provided in the past. These binaries no longer are being updated, but the compile options are listed here for reference purposes.

- Linux 2.2.xx SPARC with **egcs** 1.1.2:

  ```
  CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc CXXFLAGS="-O3 -
  fno-omit-frame-pointer   -felide-constructors   -fno-exceptions   -
  fno-rtti"       ./configure       --prefix=/usr/local/mysql       -
  -with-extra-charsets=complex      --enable-thread-safe-client      -
  -enable-local-infile --enable-assembler --disable-shared
  ```

- Linux 2.2.x with x686 with **gcc** 2.95.2:

  ```
  CFLAGS="-O3  -mpentiumpro"  CXX=gcc  CXXFLAGS="-O3  -mpentiumpro  -
  felide-constructors   -fno-exceptions   -fno-rtti"   ./configure   -
  -prefix=/usr/local/mysql            --enable-assembler            -
  -with-mysqld-ldflags=-all-static         --disable-shared         -
  -with-extra-charsets=complex
  ```

- SunOS 4.1.4 2 sun4c with **gcc** 2.7.2.1:

  ```
  CC=gcc  CXX=gcc  CXXFLAGS="-O3  -felide-constructors"  ./configure  -
  -prefix=/usr/local/mysql             --disable-shared             -
  -with-extra-charsets=complex --enable-assembler
  ```

- SunOS 5.5.1 (and above) sun4u with **egcs** 1.0.3a or 2.90.27 or

  gcc 2.95.2 and newer: `CC=gcc CFLAGS="-O3" CXX=gcc CXXFLAGS="-O3 -felide-constructors -fno-exceptions -fno-rtti" ./configure -prefix=/usr/local/mysql --with-low-memory -with-extra-charsets=complex --enable-assembler`

- SunOS 5.6 i86pc with **gcc** 2.8.1:

  ```
  CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql -
  -with-low-memory --with-extra-charsets=complex
  ```

- BSDI BSD/OS 3.1 i386 with **gcc** 2.7.2.1:

  ```
  CC=gcc CXX=gcc CXXFLAGS=-O ./configure --prefix=/usr/local/mysql -
  -with-extra-charsets=complex
  ```

- BSDI BSD/OS 2.1 i386 with **gcc** 2.7.2:

  ```
  CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql -
  -with-extra-charsets=complex
  ```

- AIX 4.2 with **gcc** 2.7.2.2:

  ```
  CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql -
  -with-extra-charsets=complex
  ```

Anyone who has more optimal options for any of the preceding configurations listed can always mail them to the MySQL `internals` mailing list. See Section 1.4.1.1, “The MySQL Mailing Lists”.

RPM distributions prior to MySQL 3.22 are user-contributed. Beginning with MySQL 3.22, RPM distributions are generated by MySQL AB.

If you want to compile a debug version of MySQL, you should add `--with-debug` or `--with-debug=full` to the preceding **configure** commands and remove any `-fomit-frame-pointer` options.

2.1.3. How to Get MySQL

Check the MySQL downloads page (http://dev.mysql.com/downloads/) for information about the current version and for downloading instructions. For a complete up-to-date list of MySQL download mirror sites, see http://dev.mysql.com/downloads/mirrors.html. There you can also find information about becoming a MySQL mirror site and how to report a bad or out-of-date mirror.

Our main mirror is located at http://mirrors.sunsite.dk/mysql/.

2.1.4. Verifying Package Integrity Using MD5 Checksums or GnuPG

After you have downloaded the MySQL package that suits your needs and before you attempt to install it, you should make sure that it is intact and has not been tampered with. MySQL AB offers three means of integrity checking:

- MD5 checksums
- Cryptographic signatures using `GnuPG`, the GNU Privacy Guard
- For RPM packages, the built-in RPM integrity verification mechanism

The following sections describe how to use these methods.

If you notice that the MD5 checksum or GPG signatures do not match, first try to download the respective package one more time, perhaps from another mirror site. If you repeatedly cannot successfully verify the integrity of the package, please notify us about such incidents, including the full package name and the download site you have been using, at `<webmaster@mysql.com>` or `<build@mysql.com>`. Do not report downloading problems using the bug-reporting system.

2.1.4.1. Verifying the MD5 Checksum

After you have downloaded a MySQL package, you should make sure that its MD5 checksum matches the one provided on the MySQL download pages. Each package has an individual checksum that you can verify with the following command, where `package_name` is the name of the package you downloaded:

```
shell> md5sum package_name
```

Example:

```
shell> md5sum mysql-standard-4.0.17-pc-linux-i686.tar.gz
60f5fe969d61c8f82e4f7f62657e1f06  mysql-standard-4.0.17-pc-linux-i686.tar.gz
```

You should verify that the resulting checksum (the string of hexadecimal digits) matches the one displayed on the download page immediately below the respective package.

Note: Make sure to verify the checksum of the **archive file** (e.g. the `.zip` or `.tar.gz` file) and not of the files that are contained inside of the archive!

Note that not all operating systems support the `md5sum` command. On some, it is simply called `md5` and others do not ship it at all. On Linux, it is part of the `GNU Text Utilities` package, which is available for a wide range of platforms. You can download the source code from http://www.gnu.org/software/textutils/ as well. If you have `OpenSSL` installed, you can also use the com-

mand `openssl md5 package_name` instead. A DOS/Windows implementation of the `md5` command line utility is available from http://www.fourmilab.ch/md5/. A graphical MD5 checking tool is `winMd5Sum`, which can be obtained from http://winmd5sum.solidblue.biz/.

2.1.4.2. Signature Checking Using GnuPG

Another method of verifying the integrity and authenticity of a package is to use cryptographic signatures. This is more reliable than using MD5 checksums, but requires more work.

Beginning with MySQL 4.0.10 (February 2003), MySQL AB started signing downloadable packages with `GnuPG` (`GNU Privacy Guard`). `GnuPG` is an Open Source alternative to the very well-known `Pretty Good Privacy` (`PGP`) by Phil Zimmermann. See http://www.gnupg.org/ for more information about `GnuPG` and how to obtain and install it on your system. Most Linux distributions ship with `GnuPG` installed by default. For more information about `OpenPGP`, see http://www.openpgp.org/.

To verify the signature for a specific package, you first need to obtain a copy of MySQL AB's public GPG build key. You can download the key from http://www.keyserver.net/. The key that you want to obtain is named `build@mysql.com`. Alternatively, you can cut and paste the key directly from the following text:

```
Key ID:
pub  1024D/5072E1F5 2003-02-03
     MySQL Package signing key (www.mysql.com) <build@mysql.com>
Fingerprint: A4A9 4068 76FC BD3C 4567  70C8 8C71 8D3B 5072 E1F5

Public Key (ASCII-armored):

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.0.6 (GNU/Linux)
Comment: For info see http://www.gnupg.org

mQGiBD4+owwRBAC14GIfUfCyEDSIePvEW3SAFUdJBtoQHH/nJKZyQT7h9bPlUWC3
RODjQReyCITRrdwyrKUGku2FmeVGwn2u2WmDMNABLnpprWPkBdCk96+OmSLN9brZ
fw2vOUgCmYv2hW0hyDHuvYlQA/BThQoADgj8AW6/0Lo7V1W9/8VuHP0gQwCgvzV3
BqOxRznNCRCRxAuAuVztHRcEAJooQK1+iSiunZMYD1WufeXfshc57S/+yeJkegNW
hxwR9pRWVArNYJdDRT+rf2RUe3vpquKNQU/hnEIUHJRQqYHo8gTxvxXNQc7fJYLV
K2HtkrPbP72vwsEKMYhhr0eKCbtLGfls9krjJ6sBgACyP/Vb7hiPwxh6rDZ7ITnE
kYpXBACmWpP8NJTkamEnPCia2ZoOHODANwpUkP43I7jsDmgtobZX9qnrAXw+uNDI
QJEXM6FSbi0LLtZciNlYsafwAPEOMDKpMqAK6IyisNtPvaLd8lH0bPAnWqcyefep
rv0sxxqUEMcM3o7wwgfN83POkDasDbs3pjwPhxvhz6//62zQJ7Q7TXlTUUwgUGFj
a2FnZSBzaWduaW5nIGtleSAod3d3Lm15c3FsLmNvbSkgPGJ1aWxkQG15c3FsLmNv
bT6IXQQTEQIAHQUCPj6jDAUJCWYBgAULBwoDBAMVAwIDFgIBAheAAAoJEIxxjTtQ
cuH1cY4AnilUwTXn8MatQOiG0a/bPxrvK/gCAJ4oinSNZRYTnblChwFaazt7PF3q
zIhMBBMRAgAMBQI+PqPRBYMJZgC7AAoJEElQ4SqycpHyJOEAn1mxHijft00bKXvu
cSo/pECUmppiAJ41M9MRVj5VcdH/KN/KjRtW6tHFPYhMBBMRAgAMBQI+QoIDBYMJ
YiKJAAoJELb1zU3GuiQ/lpEAoIhpp6BozKI8p6eaabzF5MlJH58pAKCu/ROofK8J
Eg2aLos+5zEYrB/LsrkCDQQ+PqMdEAgA7+GJfxbMdY4wslPnjH9rF4N2qfWsEN/l
xaZoJYc3a6M02WCnHl6ahT2/tBK2w1QI4YFteR47gCvtgb6O1JHffOo2HfLmRDRi
Rjd1DTCHqeyX7CHhcghj/dNRlW2Z0l5QFEcmV9U0Vhp3aFfWC4Ujfs3LU+hkAWzE
7zaD5cH9J7yv/6xuZVw411x0h4UqsTcWMu0iM1BzELqX1DY7LwoPEb/O9Rkbf4fm
Le11EzIaCa4PqARXQZc4dhSinMt6K3X4BrRsKTfozBu74F47D8Ilbf5vSYHbuE5p
/1oIDznkg/p8kW+3FxuWrycciqFTcNz215yyX39LXFnlLzKUb/F5GwADBQf+Lwqq
a8CGrRfsOAJxim63CHfty5mUc5rUSnTslGYEIOCR1BeQauyPZbPDsDD9MZ1ZaSaf
anFvwFG6Llx9xkU7tzq+vKLoWkm4u5xf3vn55VjnSd1aQ9eQnUcXiL4cnBGoTbOW
I39EcyzgslzBdC++MPjcQTcA7p6JUVsP6oAB3FQWg54tuUo0Ec8bsM8b3Ev42Lmu
QT5NdKHGwHsXTPtl0klk4bQk4OajHsiy1BMahpT27jWjJlMiJc+IWJ0mghkKHt92
6s/ymfdf5HkdQ1cyvsz5tryVI3Fx78XeSYfQvuuwqp2H139pXGEkg0n6KdUOetdZ
Whe70YGNPw1yjWJT1IhMBBgRAgAMBQI+PqMdBQkJZgGAAAoJEIxxjTtQcuH17p4A
n3r1QpVC9yhnW2cSAjq+kr72GX0eAJ4295kl6NxYEuFApmr1+0uUq/SlsQ==
=YJkx
-----END PGP PUBLIC KEY BLOCK-----
```

You can import the build key into your personal public GPG keyring by using `gpg --import`. For example, if you save the key in a file named `mysql_pubkey.asc`, the import command looks like this:

```
shell> gpg --import mysql_pubkey.asc
```

See the GPG documentation for more information on how to work with public keys.

After you have downloaded and imported the public build key, download your desired MySQL package and the corresponding signature, which also is available from the download page. The signature file has the same name as the distribution file with an `.asc` extension. For example:

Distribution file	`mysql-standard-4.0.17-pc-linux-i686.tar.gz`
Signature file	`mysql-standard-4.0.17-pc-linux-i686.tar.gz.asc`

Make sure that both files are stored in the same directory and then run the following command to verify the signature for the distribution file:

```
shell> gpg --verify package_name.asc
```

Example:

```
shell> gpg --verify mysql-standard-4.0.17-pc-linux-i686.tar.gz.asc
gpg: Warning: using insecure memory!
gpg: Signature made Mon 03 Feb 2003 08:50:39 PM MET
using DSA key ID 5072E1F5
gpg: Good signature from
     "MySQL Package signing key (www.mysql.com) <build@mysql.com>"
```

The `Good signature` message indicates that everything is all right. You can ignore the `insecure memory` warning.

2.1.4.3. Signature Checking Using RPM

For RPM packages, there is no separate signature. RPM packages have a built-in GPG signature and MD5 checksum. You can verify a package by running the following command:

```
shell> rpm --checksig package_name.rpm
```

Example:

```
shell> rpm --checksig MySQL-server-4.0.10-0.i386.rpm
MySQL-server-4.0.10-0.i386.rpm: md5 gpg OK
```

Note: If you are using RPM 4.1 and it complains about `(GPG) NOT OK (MISSING KEYS: GPG#5072e1f5)`, even though you have imported the MySQL public build key into your own GPG keyring, you need to import the key into the RPM keyring first. RPM 4.1 no longer uses your personal GPG keyring (or GPG itself). Rather, it maintains its own keyring because it is a system-wide application and a user's GPG public keyring is a user-specific file. To import the MySQL public key into the RPM keyring, first obtain the key as described in the previous section. Then use **rpm --import** to import the key. For example, if you have the public key stored in a file named `mysql_pubkey.asc`, import it using this command:

```
shell> rpm --import mysql_pubkey.asc
```

If you need to obtain the MySQL public key, see Section 2.1.4.2, “Signature Checking Using `GnuPG`”.

2.1.5. Installation Layouts

This section describes the default layout of the directories created by installing binary or source distributions provided by MySQL AB. If you install a distribution provided by another vendor, some other layout might be used.

On Windows, the default installation directory is `C:\mysql`. With MySQL version 4.1.5 and higher, this has changed to `C:\Program Files\MySQL\MySQL Server 4.1`, where 4.1 is the major version of the installation. The folder has the following subdirectories:

Directory	Contents of Directory
`bin`	Client programs and the **mysqld** server
`data`	Log files, databases
`Docs`	Documentation
`examples`	Example programs and scripts
`include`	Include (header) files
`lib`	Libraries
`scripts`	Utility scripts
`share`	Error message files

Installations created from Linux RPM distributions result in files under the following system directories:

Directory	Contents of Directory
`/usr/bin`	Client programs and scripts
`/usr/sbin`	The **mysqld** server
`/var/lib/mysql`	Log files, databases
`/usr/share/doc/packages`	Documentation
`/usr/include/mysql`	Include (header) files
`/usr/lib/mysql`	Libraries
`/usr/share/mysql`	Error message and character set files
`/usr/share/sql-bench`	Benchmarks

On Unix, a **tar** file binary distribution is installed by unpacking it at the installation location you choose (typically `/usr/local/mysql`) and creates the following directories in that location:

Directory	Contents of Directory
`bin`	Client programs and the **mysqld** server
`data`	Log files, databases
`docs`	Documentation, ChangeLog
`include`	Include (header) files
`lib`	Libraries

`scripts`	**mysql_install_db**
`share/mysql`	Error message files
`sql-bench`	Benchmarks

A source distribution is installed after you configure and compile it. By default, the installation step installs files under `/usr/local`, in the following subdirectories:

Directory	Contents of Directory
`bin`	Client programs and scripts
`include/mysql`	Include (header) files
`info`	Documentation in Info format
`lib/mysql`	Libraries
`libexec`	The **mysqld** server
`share/mysql`	Error message files
`sql-bench`	Benchmarks and `crash-me` test
`var`	Databases and log files

Within an installation directory, the layout of a source installation differs from that of a binary installation in the following ways:

- The **mysqld** server is installed in the `libexec` directory rather than in the `bin` directory.
- The data directory is `var` rather than `data`.
- **mysql_install_db** is installed in the `bin` directory rather than in the `scripts` directory.
- The header file and library directories are `include/mysql` and `lib/mysql` rather than `include` and `lib`.

You can create your own binary installation from a compiled source distribution by executing the `scripts/make_binary_distribution` script from the top directory of the source distribution.

2.2. Standard MySQL Installation Using a Binary Distribution

The next several sections cover the installation of MySQL on platforms where we offer packages using the native packaging format of the respective platform. (This is also known as performing a ``binary install.'') However, binary distributions of MySQL are available for many other platforms as well. See Section 2.7, “Installing MySQL on Other Unix-Like Systems” for generic installation instructions for these packages that apply to all platforms.

See Section 2.1, “General Installation Issues” for more information on what other binary distributions are available and how to obtain them.

2.3. Installing MySQL on Windows

A native Windows version of MySQL has been available from MySQL AB since version 3.21 and rep-

resents a sizable percentage of the daily downloads of MySQL. This section describes the process for installing MySQL on Windows.

With the release of MySQL 4.1.5, MySQL AB has introduced a new installer for the Windows version of MySQL, combined with a new GUI Configuration Wizard. This combination automatically installs MySQL, creates an option file, starts the server, and secures the default user accounts.

If you have installed a version of MySQL prior to version 4.1.5, you must perform the following steps:

1. Obtain and install the distribution.
2. Set up an option file if necessary.
3. Select the server that you want to use.
4. Start the server.
5. Assign passwords to the initial MySQL accounts.

This process also must be followed with newer MySQL installations where the installation package does not include an installer.

MySQL for Windows is available in two distribution formats:

- The binary distribution contains a setup program that installs everything you need so that you can start the server immediately.
- The source distribution contains all the code and support files for building the executables using the VC++ 6.0 compiler.

Generally speaking, you should use the binary distribution. It's simpler, and you need no additional tools to get MySQL up and running.

This section describes how to install MySQL on Windows using a binary distribution. To install using a source distribution, see Section 2.8.6, “Installing MySQL from Source on Windows”.

2.3.1. Windows System Requirements

To run MySQL on Windows, you need the following:

- A 32-bit Windows operating system such as 9x, Me, NT, 2000, XP, or Windows Server 2003.

  A Windows NT based operating system (NT, 2000, XP, 2003) permits you to run the MySQL server as a service. The use of a Windows NT based operating system is strongly recommended. See Section 2.3.12, “Starting MySQL as a Windows Service”.

- TCP/IP protocol support.
- A copy of the MySQL binary distribution for Windows, which can be downloaded from http://dev.mysql.com/downloads/. See Section 2.1.3, “How to Get MySQL”.

  Note: If you download the distribution via FTP, we recommend the use of an adequate FTP client with a resume feature to avoid corruption of files during the download process.

- A tool that can read `.zip` files, to unpack the distribution file.
- Enough space on the hard drive to unpack, install, and create the databases in accordance with your requirements (generally a minimum of 200 megabytes is recommended.)

You may also have the following optional requirements:

- If you plan to connect to the MySQL server via ODBC, you also need a Connector/ODBC driver. See Section 23.1, “MySQL ODBC Support”.
- If you need tables with a size larger than 4GB, install MySQL on an NTFS or newer filesystem. Don't forget to use `MAX_ROWS` and `AVG_ROW_LENGTH` when you create tables. See Section 13.2.6, “`CREATE TABLE` Syntax”.

2.3.2. Choosing An Installation Package

Starting with MySQL version 4.1.5, there are three install packages to choose from when installing MySQL on Windows. The Packages are as follows:

- **The Essentials Package**: This package has a filename similar to `mysql-essential-4.1.9-win32.msi` and contains the minimum files needed to install MySQL on Windows, including the Configuration Wizard. This package does not include optional components such as the embedded server and benchmark suite.
- **The Complete Package**: This package has a filename similar to `mysql-4.1.9-win32.zip` and contains all files needed for a complete Windows installation, including the Configuration Wizard. This package includes optional components such as the embedded server and benchmark suite.
- **The Noinstall Archive**: This package has a filename similar to `mysql-noinstall-4.1.9-win32.zip` and contains all the files found in the Complete install package, with the exception of the Configuration Wizard. This package does not include an automated installer, and must be manually installed and configured.

The Essentials package is recommended for most users.

Your choice of install package affects the installation process you must follow. If you choose to install either the Essentials or Complete install packages, see Section 2.3.3, “Installing MySQL with the Automated Installer”. If you choose to install MySQL from the Noinstall archive, see Section 2.3.6, “Installing MySQL from a noinstall Zip Archive”.

2.3.3. Installing MySQL with the Automated Installer

Starting with MySQL 4.1.5, users can use the new MySQL Installation Wizard and MySQL Configuration Wizard to install MySQL on Windows. The MySQL Installation Wizard and MySQL Configuration Wizard are designed to install and configure MySQL in such a way that new users can immediately get started using MySQL.

The MySQL Installation Wizard and MySQL Configuration Wizard are available in the Essentials and Complete install packages, and are recommended for most standard MySQL installations. Exceptions include users who need to install multiple instances of MySQL on a single server and advanced users who want complete control of server configuration.

If you are installing a version of MySQL prior to MySQL 4.1.5, please follow the instructions for in-

stalling MySQL from the Noinstall installation package. See Section 2.3.6, “Installing MySQL from a noinstall Zip Archive”.

2.3.4. Using the MySQL Installation Wizard

2.3.4.1. Introduction

MySQL Installation Wizard is a new installer for the MySQL server that uses the latest installer technologies for Microsoft Windows. The MySQL Installation Wizard, in combination with the MySQL Configuration Wizard, allows a user to install and configure a MySQL server that is ready for use immediately after installation.

The MySQL Installation Wizard is the standard installer for all MySQL server distributions, version 4.1.5 and higher. Users of previous versions of MySQL need to manually shut down and remove their existing MySQL installations before installing MySQL with the MySQL Installation Wizard. See Section 2.3.4.7, “Upgrading MySQL” for more information on upgrading from a previous version.

Microsoft has included an improved version of their Microsoft Windows Installer (MSI) in the recent versions of Windows. Using the MSI has become the de-facto standard for application installations on Windows 2000, Windows XP, and Windows Server 2003. The MySQL Installation Wizard makes use of this technology to provide a smoother and more flexible installation progress.

The Microsoft Windows Installer Engine was updated with the release of Windows XP; those using a previous version of Windows can reference this Microsoft Knowledge Base article [http://support.microsoft.com/default.aspx?scid=kb;EN-US;292539] for information on upgrading to the latest version of the Windows Installer Engine.

Further, Microsoft has introduced the WiX (Windows Installer XML) tool set recently. It is the first highly acknowledged Open Source project from Microsoft. We switched to WiX because it is an Open Source project and it allows us to handle the complete Windows installation process in a flexible way with scripts.

Improving the MySQL Installation Wizard depends on the support and feedback of users like you. If you find that the MySQL Installation Wizard is lacking some feature important to you, or if you discover a bug, please use our MySQL Bug System [http://bugs.mysql.com] to request features or report problems.

2.3.4.2. Downloading and Starting the MySQL Installation Wizard

The MySQL server install packages can be downloaded from http://dev.mysql.com/downloads/. If the package you download is contained within a Zip archive, you need to extract the archive first.

The process for starting the wizard depends on the contents of the install package you download. If there is a `setup.exe` file present, double-click it to start the install process. If there is a `.msi` file present, double-click it to start the install process.

2.3.4.3. Choosing an Install Type

There are up three installation types available: `Typical`, `Complete`, and `Custom`.

The `Typical` installation type installs the MySQL server, the **mysql** command-line client, and the command-line utilities. The command-line clients and utilities include **mysqldump**, **myisamchk**, and several other tools to help you manage the MySQL server.

The `Complete` installation type installs all components included in the installation package. The full installation package includes components such as the embedded server library, the benchmark suite, support scripts, and documentation.

The `Custom` installation type gives you complete control over which packages you wish to install and the installation path that is used. See Section 2.3.4.4, “The Custom Install Dialog” for more information on performing a custom install.

If you choose the `Typical` or `Complete` installation types and click the Next button, you advance to the confirmation screen to confirm your choices and begin the installation. If you choose the `Custom` installation type and click the Next button, you advance to the custom install dialog, described in Section 2.3.4.4, “The Custom Install Dialog”

2.3.4.4. The Custom Install Dialog

If you wish to change the installation path or the specific components that are installed by the MySQL Installation Wizard, you should choose the `Custom` installation type.

All available components are listed in a tree view on the left side of the custom install dialog. Components that are not installed have a red `X` icon, components that are installed have a gray icon. To change whether a component is installed, click on the component's icon and choose an new option from the drop-down list that appears.

You can change the default installation path by clicking the Change... button to the right of the displayed installation path.

After choosing your install components and installation path, click the Next button to advance to the confirmation dialog.

2.3.4.5. The Confirmation Dialog

Once you choose an installation type and optionally choose your installation components, you advance to the confirmation dialog. Your installation type and installation path are displayed for you to review.

To install MySQL if you are satisfied with your settings, click the Install button. To change your settings, click the Back button. To exit the MySQL Installation Wizard without installing MySQL, click the Cancel button.

After installation is complete, you are given the option of registering with the MySQL web site. Registration gives you access to post in the MySQL forums at forums.mysql.com [http://forums.mysql.com], along with the ability to report bugs at bugs.mysql.com [http://bugs.mysql.com] and to subscribe to the newsletter. The final screen of the installer provides a summary of the installation and gives you the option to launch the MySQL Configuration Wizard, which you can use to create a configuration file, install the MySQL service, and configure security.

2.3.4.6. Changes Made by MySQL Installation Wizard

Once you click the Install button, the MySQL Installation Wizard begins the installation process and makes certain changes to your system which are described in the sections that follow.

Changes to the Registry

The MySQL Installation Wizard creates one Windows registry key in a typical install situation, located in `HKEY_LOCAL_MACHINE\SOFTWARE\MySQL AB`.

The MySQL Installation Wizard creates a key named after the major version of the server that is being installed, such as `MySQL Server 4.1`. It contains two string values, `Location` and `Version`. The `Location` string contains the path to the installation directory. In a default installation it contains `C:\Program Files\MySQL\MySQL Server 4.1\`. The `Version` string contains the release number. For example, for an installation of MySQL Server 4.1.5 the key contains a value of `4.1.5`.

These registry keys are used to help external tools identify the installed location of the MySQL server, preventing a complete scan of the hard-disk to determine the installation path of the MySQL server. The registry keys are not required to run the server and when using the `noinstall` Zip archive the registry keys are not created.

Changes to the Start Menu

The MySQL Installation Wizard creates a new entry in the Windows Start menu under a common MySQL menu heading named after the major version of MySQL that you have installed. For example, if you install MySQL 4.1, the MySQL Installation Wizard creates a MySQL Server 4.1 section in the start menu.

The following entries are created within the new Start menu section:

- `MySQL Command Line Client`: This is a shortcut to the **mysql** command-line client and is configured to connect as the `root` user. The shortcut prompts for a `root` user password when connecting.
- `MySQL Server Instance Config Wizard`: This is a shortcut to the MySQL Configuration Wizard. Use this shortcut to configure a newly installed server, or to re-configure an existing server.
- `MySQL Documentation`: This is a link to the MySQL server documentation that is stored locally in the MySQL server installation directory. This option is not available when the MySQL server is installed from the `essential` installation package.

Changes to the File System

The MySQL Installation Wizard by default installs the MySQL server to `C:\Program Files\MySQL\MySQL Server 4.1`, where `Program Files` is the default location for applications in your system, and `4.1` is the major version of your MySQL server. This is the new recommended location for the MySQL server, replacing the previous default location of `c:\mysql`.

By default, all MySQL applications are stored in a common directory at `C:\Program Files\MySQL`, where `Program Files` is the default location for applications in your Windows installation. A typical MySQL installation on a developer machine may look like this:

```
C:\Program Files\MySQL\MySQL Server 4.1
C:\Program Files\MySQL\MySQL Server 5.0
C:\Program Files\MySQL\MySQL Administrator 1.0
C:\Program Files\MySQL\MySQL Query Browser 1.0
```

This approach makes it easier to manage and maintain all MySQL applications installed on a particular system.

2.3.4.7. Upgrading MySQL

From MySQL version 4.1.5, the new MySQL Installation Wizard can perform server upgrades automatically using the upgrade capabilities of MSI. That means you do not need to remove a previous installation manually before installing a new release. The installer automatically shuts down and removes the previous MySQL service before installing the new version.

Automatic upgrades are only available when upgrading between installations that have the same major and minor version numbers. For example, you can upgrade automatically from MySQL 4.1.5 to MySQL 4.1.6, but not from MySQL 4.1 to MySQL 5.0.

If you are upgrading MySQL version 4.1.4 or earlier to version 4.1.5 or later, you must first manually

shut down and remove the older installation before upgrading. Be sure to back up your databases before performing such an upgrade, so that you can restore the databases after the upgrade is completed. It is always recommended that you back up your data before performing any upgrades.

See Section 2.3.15, “Upgrading MySQL on Windows”.

2.3.5. Using the Configuration Wizard

2.3.5.1. Introduction

The MySQL Configuration Wizard helps automate the process of configuring your server under Windows. The MySQL Configuration Wizard creates a custom `my.ini` file by asking you a series of questions and then applying your responses to a template to generate a `my.ini` file that is tuned to your installation.

The MySQL Configuration Wizard is included with the MySQL server starting with MySQL version 4.1.5, but is designed to work with MySQL servers versions 4.1 and higher. The MySQL Configuration Wizard is currently available for Windows users only.

MySQL Configuration Wizard is to a large extent the result of feedback MySQL AB has received from many users over a period of several years. However, if you find it's lacking some feature important to you, or if you discover a bug, please use our MySQL Bug System [http://bugs.mysql.com] to request features or report problems.

2.3.5.2. Starting the MySQL Configuration Wizard

The MySQL Configuration Wizard is typically launched from the MySQL Installation Wizard, as the MySQL Installation Wizard exits. You can also launch the MySQL Configuration Wizard by clicking the MySQL Server Instance Config Wizard entry in the MySQL section of the `Start` menu.

In addition, you can navigate to the `bin` directory of your MySQL installation and launch the `MySQLInstanceConfig.exe` file directly.

2.3.5.3. Choosing a Maintenance Option

If the MySQL Configuration Wizard detects an existing `my.ini` file, you have the option of either reconfiguring your existing server, or removing the server instance by deleting the `my.ini` file and stopping and removing the MySQL service.

To reconfigure an existing server, choose the `Re-configure Instance` option and click the Next button. Your existing `my.ini` file is renamed to `my` *`timestamp`*`.ini.bak`, where *`timestamp`* is the date and time the existing `my.ini` file was created. To remove the existing server instance, choose the `Remove Instance` option and click the Next button.

If you choose the `Remove Instance` option, you advance to a confirmation window. Click the Execute button and the MySQL Configuration Wizard stops and removes the MySQL service and deletes the `my.ini` file. The server installation and its `data` folder are not removed.

If you choose the `Re-configure Instance` option, you advance to the `Configuration Type` dialog where you can choose the type of installation you wish to configure.

2.3.5.4. Choosing a Configuration Type

When you start the MySQL Configuration Wizard for a new MySQL installation, or choose the `Re-configure Instance` option for an existing installation, you advance to the `Configuration Type` dialog.

There are two configuration types available: `Detailed Configuration` and `Standard Configuration`. The `Standard Configuration` option is intended for new users who want to get started with MySQL quickly without having to make a lot of decisions in regards to server configuration. The `Detailed Configuration` option is intended for advanced users who want more fine-grained control of server configuration.

If you are new to MySQL and need a server configured as a single-user developer machine the `Standard Configuration` should suit your needs. Choosing the `Standard Configuration` option causes the MySQL Configuration Wizard to automatically set all configuration options with the exception of the `Service Options` and `Security Options`.

The `Standard Configuration` sets options that may be incompatible with systems where there are existing MySQL installations. If you have an existing MySQL installation on your system in addition to the installation you wish to configure, the `Detailed Configuration` option is recommended.

To complete the `Standard Configuration`, please refer to the sections on `Service Options` and `Security Options`, located at Section 2.3.5.11, “The Service Options Dialog” and Section 2.3.5.12, “The Security Options Dialog” respectively.

2.3.5.5. The Server Type Dialog

There are three different server types available to choose from, and the server type you choose affects the decisions the MySQL Configuration Wizard makes with regards to memory, disk, and processor usage.

- `Developer Machine`: Choose this option for a typical desktop workstation where MySQL is intended only for personal use. It is assumed that many other desktop applications are running. The MySQL server is configured to use minimal system resources.
- `Server Machine`: Choose this option for a server machine where the MySQL server is running alongside other server applications such as FTP, email, and web servers. The MySQL server is configured to use a medium portion of the system resources.
- `Dedicated MySQL Server Machine`: Choose this option for a server machine that is intended to run only the MySQL server. It is assumed that no other applications are running. The MySQL server is configured to use all available system resources.

2.3.5.6. The Database Usage Dialog

The `Database Usage` dialog allows you to indicate the table handlers you expect to use when creating MySQL tables. The option you choose determines whether the InnoDB table handler is available and what percentage of the server resources are available to InnoDB.

- `Multifunctional Database`: This option enables both the InnoDB and MyISAM table handlers and divides resources evenly between the two. This option is recommended for users that use both table handlers on a regular basis.
- `Transactional Database Only`: This option enables both the InnoDB and MyISAM table handlers but dedicates most server resources toward the InnoDB table handler. This option is recommended for users that use InnoDB almost exclusively and make only minimal use of MyISAM.
- `Non-Transactional Database Only`: This option disables the InnoDB table handler completely and dedicates all server resources to the MyISAM table handler. This option is recommended for users who do not use InnoDB.

2.3.5.7. The InnoDB Tablespace Dialog

Some users may want to locate the InnoDB tablespace files in a different location than the MySQL server data directory. Placing the tablespace files in a separate location can be desirable if your system has a higher capacity or higher performance storage device available, such as a RAID storage system.

To change the default location for the InnoDB tablespace files, choose a new drive from the drop-down list of drive letters and choose a new path from the drop-down list of paths. To create a custom path, click the ... button.

If you are modifying the configuration of an existing server, you must click the Modify button before you change the path. In this situation you have to manually move the existing tablespace files to the new location before starting the server.

2.3.5.8. The Concurrent Connections Dialog

It is important to set a limit to the number of concurrent connections to the MySQL server that can be established to prevent the server from running out of resources. The `Concurrent Connections` dialog allows you to choose the expected usage of your server, and sets the limit for concurrent connections accordingly. It is also possible to manually set the concurrent connection limit.

- `Decision Support (DSS)/OLAP`: Choose this option if your server does not require a large number of concurrent connections. The maximum number of connections is set at 100, with an average of 20 concurrent connections assumed.
- `Online Transaction Processing (OLTP)`: Choose this option if your server requires a large number of concurrent connections. The maximum number of connections is set at 500.
- `Manual Setting`: Choose this option to manually set the maximum number of concurrent connections to the server. Choose the number of concurrent connections from the drop-down box provided, or type the maximum number of connections into the drop-down box if the number you desire is not listed.

2.3.5.9. The Networking Options Dialog

Use the `Networking Options` dialog to enable or disable TCP/IP networking and to configure the port number that is used to connect to the MySQL server.

TCP/IP networking is enabled by default. To disable TCP/IP networking, uncheck the box next to the `Enable TCP/IP Networking` option.

Port 3306 is used by default. To change the port used to access MySQL, choose a new port number from the drop-down box or type a new port number directly into the drop-down box. If the port number you choose is in use you are prompted to confirm your choice of port number.

2.3.5.10. The Character Set Dialog

The MySQL server supports multiple character sets and it is possible to set a default server character set that is applied to all tables, columns, and databases unless overridden. Use the `Character Set` dialog to change the default character set of the MySQL server.

- `Standard Character Set`: Choose this option if you want to use `Latin1` as the default server character set. `Latin1` is used for English and many Western European languages.

- `Best Support For Multilingualism`: Choose this option if you want to use `UTF8` as the default server character set. `UTF8` can store characters from many different languages in a single character set.
- `Manual Selected Default Character Set / Collation`: Choose this option if you want to pick the server's default character set manually. Choose the desired character set from the provided drop-down list.

2.3.5.11. The Service Options Dialog

On Windows NT based platforms, the MySQL server can be installed as a service. When installed as a service, the MySQL server can be started automatically during system startup, and even restarted automatically by Windows in the event of a service failure.

The MySQL Configuration Wizard installs the MySQL server as a service by default, using the service name `MySQL`. If you do not wish to install the service, un-check the box next to the `Install As Windows Service` option. You can changed the service name by picking a new service name from the drop-down box provided or by typing a new service name into the drop-down box.

To install the MySQL server as a service but not have it started automatically at startup, un-check the box next to the `Launch the MySQL Server automatically` option.

2.3.5.12. The Security Options Dialog

It is strongly recommended that you set a `root` password for your MySQL server, and the MySQL Configuration Wizard requires you set a `root` password by default. If you do not wish to set a `root` password, un-check the box next to the `Modify Security Settings` option.

To set the `root` password, type the desired password into both the `New root password` and `Confirm` boxes. If you are re-configuring an existing server, you also need to enter the existing `root` password into the `Current root password` box.

To prevent `root` logins from across the network, check the box next to the `Root may only connect from localhost` option. This increases the security of your `root` account.

To create an anonymous user account, check the box next to the `Create An Anonymous Account` option. Creating an anonymous account can decrease server security and cause login and permission difficulties and is not recommended.

2.3.5.13. The Confirmation Dialog

The final dialog in the MySQL Configuration Wizard is the `Confirmation Dialog`. To start the configuration process, click the Execute button. To return to a previous dialog, click the Back button. To exit the MySQL Configuration Wizard without configuring the server, click the Cancel button.

After you click the Execute button, the MySQL Configuration Wizard performs a series of tasks with progress displayed onscreen as the tasks are performed.

The MySQL Configuration Wizard firsts determines various configuration file options based on your choices using a template prepared by MySQL AB developers and engineers. This template is named `my-template.ini` and is located in your server installation directory.

The MySQL Configuration Wizard then writes these options to a `my.ini` file. The final location of the `my.ini` file is displayed next to the `Write configuration file` task.

If you chose to create a service for the MySQL server the MySQL Configuration Wizard creates ans starts the service. If you are re-configuring an existing service, the MySQL Configuration Wizard re-

starts the service to apply your configuration changes.

If you chose to set a `root` password, the MySQL Configuration Wizard connects to the server, sets your new `root` password and applies any other security settings you may have selected.

After the MySQL Configuration Wizard has completed its tasks, a summary is shown. Click the Finish button to exit the MySQL Configuration Wizard.

2.3.5.14. The Location of the my.ini File

In MySQL installations prior to version 4.1.5 it was customary to name the server configuration file `my.cnf` or `my.ini` and locate the file either at `c:\my.cnf` or `c:\Windows\my.ini`.

The new MySQL Configuration Wizard places the `my.ini` file in the installation directory of the MySQL server. This helps associate configuration files with particular server instances.

To ensure that the MySQL server knows where to look for the `my.ini` file, an argument similar to this is passed to the MySQL server as part of the service installation: `--defaults-file="C:\Program Files\MySQL\MySQL Server 4.1\my.ini"`, where *`C:\Program Files\MySQL\MySQL Server 4.1`* is replaced with the installation path to the MySQL Server.

The `--defaults-file` instructs the MySQL server to read the specified file for configuration options.

2.3.5.15. Editing The my.ini File

To modify the `my.ini` file, open it with a text editor and make any necessary changes. You can also modify the server configuration with the MySQL Administrator [http://www.mysql.com/products/administrator/] utility.

MySQL clients and utilities such as the **mysql** command-line client and **mysqldump** are not able to locate the `my.ini` file located in the server installation directory. To configure the client and utility applications, create a new `my.ini` file in the `c:\Windows` directory.

2.3.6. Installing MySQL from a noinstall Zip Archive

Users who are installing from the Noinstall package, or who are installing a version of MySQL prior to 4.1.5 can use the instructions in this section to manually install MySQL. If you are installing a version prior to 4.1.5 with an install package that includes a Setup program, substitute running the Setup program for extracting the archive.

The process for installing MySQL from a Zip archive is as follows:

1. Extract the archive to the desired install directory.
2. Create an option file.
3. Choose a MySQL server type.
4. Start the MySQL server.
5. Secure the default user accounts.

This process is described in the sections that follow.

2.3.7. Extracting the Install Archive

To install MySQL manually, do the following:

1. If you are upgrading from a previous version please refer to Section 2.3.15, “Upgrading MySQL on Windows” before beginning the upgrade process.
2. If you are using a Windows NT-based operating system such as Windows NT, Windows 2000, Windows XP, or Windows Server 2003, make sure that you are logged in as a user with administrator privileges.
3. Choose an installation location. Traditionally the MySQL server is installed at `C:\mysql`, and the new MySQL Installation Wizard installs MySQL to `C:\Program Files\MySQL`. If you do not install MySQL at `C:\mysql`, you must specify the path to the install directory during startup or in an option file. See Section 2.3.8, “Creating an Option File”.
4. Extract the install archive to the chosen installation location using your preferred Zip archive tool. Some tools may extract the archive to a folder within your chosen installation location. If this occurs you can move the contents of the subfolder into the chosen installation location.

2.3.8. Creating an Option File

If you need to specify startup options when you run the server, you can indicate them on the command line or place them in an option file. For options that are used every time the server starts, you may find it most convenient to use an option file to specify your MySQL configuration. This is particularly true under the following circumstances:

- The installation or data directory locations are different from the default locations (`C:\mysql` and `C:\mysql\data`).
- You need to tune the server settings. For example, to use the `InnoDB` transactional tables in MySQL 3.23, you must manually add some extra lines to the option file, as described in Section 15.4, “`InnoDB` Configuration”. (As of MySQL 4.0, `InnoDB` creates its data files and log files in the data directory by default. This means you need not configure `InnoDB` explicitly. You may still do so if you wish, and an option file is useful in this case, too.)

When the MySQL server starts on Windows, it looks for options in two files: the `my.ini` file in the Windows directory, and the `C:\my.cnf` file. The Windows directory typically is named something like `C:\WINDOWS` or `C:\WinNT`. You can determine its exact location from the value of the `WINDIR` environment variable using the following command:

```
C:\> echo %WINDIR%
```

MySQL looks for options first in the `my.ini` file, then in the `my.cnf` file. However, to avoid confusion, it's best if you use only one file. If your PC uses a boot loader where the `C:` drive isn't the boot drive, your only option is to use the `my.ini` file. Whichever option file you use, it must be a plain text file.

You can also make use of the example option files included with your MySQL distribution. Look in your install directory for files such as my-small.cnf, my-medium.cnf, my-large.cnf, etc., which you can rename and copy to the appropriate location for use as a base configuration file.

An option file can be created and modified with any text editor, such as the `Notepad` program. For example, if MySQL is installed at `E:\mysql` and the data directory is located at `E:\mydata\data`,

you can create the option file and set up a `[mysqld]` section to specify values for the `basedir` and `datadir` parameters:

```
[mysqld]
# set basedir to your installation path
basedir=E:/mysql
# set datadir to the location of your data directory
datadir=E:/mydata/data
```

Note that Windows pathnames are specified in option files using forward slashes rather than backslashes. If you do use backslashes, you must double them:

```
[mysqld]
# set basedir to your installation path
basedir=E:\\mysql
# set datadir to the location of your data directory
datadir=E:\\mydata\\data
```

On Windows, the MySQL installer places the data directory directly under the directory where you install MySQL. If you would like to use a data directory in a different location, you should copy the entire contents of the `data` directory to the new location. For example, by default, the installer places MySQL in `C:\mysql` and the data directory in `C:\mysql\data`. If you want to use a data directory of `E:\mydata`, you must do two things:

- Move the data directory from `C:\mysql\data` to `E:\mydata`.
- Use a `--datadir` option to specify the new data directory location each time you start the server.

2.3.9. Selecting a MySQL Server type

Starting with MySQL 3.23.38, the Windows distribution includes both the normal and the MySQL-Max server binaries.

Up through the early releases of MySQL 4.1, the servers included in Windows distributions are named like this:

Binary	Description
mysqld	Compiled with full debugging and automatic memory allocation checking, and `InnoDB` and `BDB` tables.
mysqld-opt	Optimized binary. From version 4.0 on, `InnoDB` is enabled. Before 4.0, this server includes no transactional table support.
mysqld-nt	Optimized binary for Windows NT, 2000, and XP with support for named pipes.
mysqld-max	Optimized binary with support for `InnoDB` and `BDB` tables.
mysqld-max-nt	Like **mysqld-max**, but compiled with support for named pipes.

We have found that the server with the most generic name (**mysqld**) is the one that many users are likely to choose by default. However, that is also the server that results in the highest memory and CPU use due to the inclusion of full debugging support. The server named **mysqld-opt** is a better general-use server choice to make instead if you don't need debugging support and don't want the maximal feature set offered by the `-max` servers or named pipe support offered by the `-nt` servers.

To make it less likely that the debugging server would be chosen inadvertently, some name changes were made from MySQL 4.1.2 to 4.1.4: **mysqld** has been renamed to **mysqld-debug** and **mysqld-opt**

has been renamed to **mysqld**. Thus, the server that includes debugging support indicates that in its name, and the server named **mysqld** is an efficient default choice. The other servers still have their same names. The resulting servers are named like this:

Binary	Description
mysqld-debug	Compiled with full debugging and automatic memory allocation checking, and `InnoDB` and `BDB` tables.
mysqld	Optimized binary with `InnoDB` support.
mysqld-nt	Optimized binary for Windows NT, 2000, and XP with support for named pipes.
mysqld-max	Optimized binary with support for `InnoDB` and `BDB` tables.
mysqld-max-nt	Like **mysqld-max**, but compiled with support for named pipes.

The name changes were not both instituted at the same time. If you have MySQL 4.1.2 or 4.1.3, it might be that you have a server named **mysqld-debug** but not one named **mysqld**. In this case, you should have a server **mysqld-opt**, which you should choose as your default server unless you need maximal features, named pipes, or debugging support.

All of the preceding binaries are optimized for modern Intel processors, but should work on any Intel i386-class or higher processor.

As of MySQL 4.0, all Windows servers have support for symbolic linking of database directories. Before MySQL 4.0, only the debugging and Max server versions include this feature.

MySQL supports TCP/IP on all Windows platforms. The **mysqld-nt** and `mysql-max-nt` servers support named pipes on Windows NT, 2000, XP, and 2003. However, the default is to use TCP/IP regardless of the platform. (Named pipes are slower than TCP/IP in many Windows configurations.)

Named pipe use is subject to these conditions:

- Starting from MySQL 3.23.50, named pipes are enabled only if you start the server with the `--enable-named-pipe` option. It is necessary to use this option explicitly because some users have experienced problems shutting down the MySQL server when named pipes were used.
- Named pipe connections are allowed only by the **mysqld-nt** or **mysqld-max-nt** servers, and only if the server is run on a version of Windows that supports named pipes (NT, 2000, XP, 2003).
- These servers can be run on Windows 98 or Me, but only if TCP/IP is installed; named pipe connections cannot be used.
- These servers can not be run on Windows 95.

Note: Most of the examples in reference manual use **mysqld** as the server name. If you choose to use a different server, such as **mysqld-nt**, make the appropriate substitutions in the commands that are shown in the examples.

2.3.10. Starting the Server for the First Time

On Windows 95, 98, or Me, MySQL clients always connect to the server using TCP/IP. (This allows any machine on your network to connect to your MySQL server.) Because of this, you must make sure that TCP/IP support is installed on your machine before starting MySQL. You can find TCP/IP on your Windows CD-ROM.

Note that if you are using an old Windows 95 release (for example, OSR2), it's likely that you have an old Winsock package; MySQL requires Winsock 2! You can get the newest Winsock from ht-

tp://www.microsoft.com/. Windows 98 has the new Winsock 2 library, so it is unnecessary to update the library.

On NT-based systems such as Windows NT, 2000, XP, or 2003, clients have two options. They can use TCP/IP, or they can use a named pipe if the server supports named pipe connections. To get MySQL to work with TCP/IP on Windows NT 4, you must install service pack 3 (or newer).

In MySQL versions 4.1 and higher, Windows servers also support shared-memory connections if started with the `--shared-memory` option. Clients can connect through shared memory by using the `--protocol=memory` option.

For information about which server binary to run, see Section 2.3.9, “Selecting a MySQL Server type”.

This section gives a general overview of starting the MySQL server. The following sections provide more specific information for starting the MySQL server from the command line or as a Windows service.

The examples in these sections assume that MySQL is installed under the default location of `C:\mysql`. Adjust the pathnames shown in the examples if you have MySQL installed in a different location.

Testing is best done from a command prompt in a console window (a ``DOS window''). This way you can have the server display status messages in the window where they are easy to see. If something is wrong with your configuration, these messages make it easier for you to identify and fix any problems.

To start the server, enter this command:

```
C:\> C:\mysql\bin\mysqld --console
```

For servers that include `InnoDB` support, you should see the following messages as the server starts:

```
InnoDB: The first specified datafile c:\ibdata\ibdata1 did not exist:
InnoDB: a new database to be created!
InnoDB: Setting file c:\ibdata\ibdata1 size to 209715200
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file c:\iblogs\ib_logfile0 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile0 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile1 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile1 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile2 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile2 size to 31457280
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: creating foreign key constraint system tables
InnoDB: foreign key constraint system tables created
011024 10:58:25  InnoDB: Started
```

When the server finishes its startup sequence, you should see something like this, which indicates that the server is ready to service client connections:

```
mysqld: ready for connections
Version: '4.0.14-log'  socket: ''  port: 3306
```

The server continues to write to the console any further diagnostic output it produces. You can open a new console window in which to run client programs.

If you omit the `--console` option, the server writes diagnostic output to the error log in the data directory (`C:\mysql\data` by default). The error log is the file with the `.err` extension.

Note: The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.9, “Post-Installation Setup and Testing”.

2.3.11. Starting MySQL from the Windows Command Line

The MySQL server can be started manually from the command line. This can be done on any version of Windows.

To start the **mysqld** server from the command line, you should start a console window (a ``DOS window'') and enter this command:

```
C:\> C:\Program Files\MySQL\MySQL Server 4.1\bin\mysqld
```

The path used in the preceding example may vary depending on the install location of MySQL on your system.

On non-NT versions of Windows, this starts **mysqld** in the background. That is, after the server starts, you should see another command prompt. If you start the server this way on Windows NT, 2000, XP, or 2003, the server runs in the foreground and no command prompt appears until the server exits. Because of this, you should open another console window to run client programs while the server is running.

You can stop the MySQL server by executing this command:

```
C:\> C:\Program Files\MySQL\MySQL Server 4.1\bin\mysqladmin -u root shutdown
```

This invokes the MySQL administrative utility **mysqladmin** to connect to the server and tell it to shut down. The command connects as the MySQL `root` user, which is the default administrative account in the MySQL grant system. Note that users in the MySQL grant system are wholly independent from any login users under Windows.

If **mysqld** doesn't start, check the error log to see whether the server wrote any messages there to indicate the cause of the problem. The error log is located in the `C:\mysql\data` directory. It is the file with a suffix of `.err`. You can also try to start the server as **mysqld --console**; in this case, you may get some useful information on the screen that may help solve the problem.

The last option is to start **mysqld** with `--standalone --debug`. In this case, **mysqld** writes a log file `C:\mysqld.trace` that should contain the reason why **mysqld** doesn't start. See Section E.1.2, “Creating Trace Files”.

Use **mysqld --verbose --help** to display all the options that **mysqld** understands. (Prior to MySQL 4.1, omit the `--verbose` option.)

2.3.12. Starting MySQL as a Windows Service

On the NT family (Windows NT, 2000, XP, 2003), the recommended way to run MySQL is to install it as a Windows service. With the MySQL server installed as a service, Windows starts and stops it server automatically when Windows starts and stops. A MySQL server installed as a service can also be controlled from the command line using **NET** commands, or with the graphical `Services` utility.

The `Services` utility (the Windows `Service Control Manager`) can be found in the Windows `Control Panel` (under `Administrative Tools` on Windows 2000, XP, and Server 2003). It is advisable to close the `Services` utility while performing server installation or removal operations from this command line. This prevents some odd errors.

Before installing MySQL as a Windows service, you should first stop the current server if it is running by using the following command:

```
C:\> C:\mysql\bin\mysqladmin -u root shutdown
```

This invokes the MySQL administrative utility **mysqladmin** to connect to the server and tell it to shut down. The command connects as the MySQL `root` user, which is the default administrative account in the MySQL grant system. Note that users in the MySQL grant system are wholly independent from any login users under Windows.

Install the server as a service using this command:

```
C:\> mysqld --install
```

If you have problems installing **mysqld** as a service using just the server name, try installing it using its full pathname. For example:

```
C:\> C:\mysql\bin\mysqld --install
```

The service-installation command does not start the server. Instructions for that are given later in this section.

Before MySQL 4.0.2, no command-line arguments can be given following the `--install` option. MySQL 4.0.2 and up offers limited support for additional arguments:

- You can specify a service name immediately following the `--install` option. The default service name is `MySQL`.
- As of MySQL 4.0.3, if a service name is given, it can be followed by a single option. By convention, this should be `--defaults-file=`*`file_name`* to specify the name of an option file from which the server should read options when it starts.

  It is possible to use a single option other than `--defaults-file`, but this is discouraged. `--defaults-file` is more flexible because it enables you to specify multiple startup options for the server by placing them in the named option file. Also, in MySQL 5.0, use of an option different from `--defaults-file` is not supported until 5.0.3.
- As of MySQL 5.0.1, you can also specify a `--local-service` option following the service name. This causes the server to run using the `LocalService` Windows account that has limited system privileges. This account is available only for Windows XP or newer. If both `--defaults-file` and `--local-service` are given following the service name, they can be in any order.

For a MySQL server that is installed as a Windows service, the following rules determine the service name and option files that the server uses:

- If the service-installation command specifies no service name or the default service name (`MySQL`) following the `--install` option, the server uses the a service name of `MySQL` and reads options from the `[mysqld]` group in the standard option files.
- If the service-installation command specifies a service name other than `MySQL` following the `--install` option, the server uses that service name. It reads options from the group that has the same name as the service, and reads options from the standard option files.

As of MySQL 4.0.17, the server also reads options from the `[mysqld]` group from the standard option files. This allows you to use the `[mysqld]` group for options that should be used by all MySQL services, and an option group with the same name as a service for use by the server installed with that service name.

- If the service-installation command specifies a `--defaults-file` option after the service name, the server reads options only from the `[mysqld]` group of the named file and ignores the standard option files.

As a more complex example, consider the following command:

```
C:\> C:\mysql\bin\mysqld --install MySQL --defaults-file=C:\my-opts.cnf
```

Here, the default service name (`MySQL`) is given after the `--install` option. If no `--defaults-file` option had been given, this command would have the effect of causing the server to read the `[mysqld]` group from the standard option files. However, because the `--defaults-file` option is present, the server reads options from the `[mysqld]` option group, but only from the named file.

You can also specify options as ``Start parameters'' in the Windows `Services` utility before you start the MySQL service.

Note: Prior to MySQL 4.0.17, a server installed as a Windows service has problems starting if its pathname or the service name contains spaces. For this reason, with older versions, avoid installing MySQL in a directory such as `C:\Program Files` or using a service name containing spaces.

Once a MySQL server has been installed as a service, Windows starts the service automatically whenever Windows starts. The service also can be started immediately from the `Services` utility, or by using the command **NET START MySQL**. The **NET** command is not case sensitive.

When run as a service, **mysqld** has no access to a console window, so no messages can be seen there. If **mysqld** doesn't start, check the error log to see whether the server wrote any messages there to indicate the cause of the problem. The error log is located in the MySQL data directory (for example, `C:\mysql\data`). It is the file with a suffix of `.err`.

When a MySQL server has been installed as a service, and the service is running, Windows stops the service automatically when Windows shuts down. The server also can be stopped manually by using the `Services` utility, the command **NET STOP MySQL**, or the command **mysqladmin shutdown**.

From MySQL 3.23.44 on, you have the choice of installing the server as a `Manual` service if you don't wish the service to be started automatically during the boot process. To do this, use the `--install-manual` option rather than the `--install` option:

```
C:\> C:\mysql\bin\mysqld --install-manual
```

To remove a server that is installed as a service, first stop it if it is running. Then use the `--remove` option to remove it:

```
C:\> C:\mysql\bin\mysqld --remove
```

For MySQL versions older than 3.23.49, one problem with automatic MySQL service shutdown is that Windows waited only for a few seconds for the shutdown to complete, then killed the database server process if the time limit was exceeded. This had the potential to cause problems. (For example, the `InnoDB` storage engine would have to perform crash recovery at the next startup.) Starting from MySQL

3.23.49, Windows waits longer for the MySQL server shutdown to complete. If you notice this still is not enough for your installation, it is safest not to run the MySQL server as a service. Instead, start it from the command-line prompt, and stop it with **mysqladmin shutdown**.

This change to tell Windows to wait longer when stopping the MySQL server works for Windows 2000 and XP. It does not work for Windows NT, where Windows waits only 20 seconds for a service to shut down, and after that kills the service process. You can increase this default by opening the Registry Editor `\winnt\system32\regedt32.exe` and editing the value of `WaitToKillServiceTimeout` at `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control` in the Registry tree. Specify the new larger value in milliseconds. For example, the value 120000 tells Windows NT to wait up to 120 seconds.

If **mysqld** is not running as a service, you can start it from the command line. For instructions, see Section 2.3.11, “Starting MySQL from the Windows Command Line”.

Please see Section 2.3.14, “Troubleshooting a MySQL Installation Under Windows” if you encounter difficulties during installation.

2.3.13. Testing The MySQL Installation

You can test whether the MySQL server is working by executing any of the following commands:

```
C:\> C:\mysql\bin\mysqlshow
C:\> C:\mysql\bin\mysqlshow -u root mysql
C:\> C:\mysql\bin\mysqladmin version status proc
C:\> C:\mysql\bin\mysql test
```

If **mysqld** is slow to respond to TCP/IP connections from client programs on Windows 9x/Me, there is probably a problem with your DNS. In this case, start **mysqld** with the `--skip-name-resolve` option and use only `localhost` and IP numbers in the `Host` column of the MySQL grant tables.

You can force a MySQL client to use a named pipe connection rather than TCP/IP by specifying the `--pipe` option or by specifying `.` (period) as the host name. Use the `--socket` option to specify the name of the pipe. As of MySQL 4.1, you can use the `--protocol=PIPE` option instead.

There are two versions of the MySQL command-line tool on Windows:

Binary	Description
mysql	Compiled on native Windows, offering limited text editing capabilities.
mysqlc	Compiled with the Cygnus GNU compiler and libraries, which offers `readline` editing. **mysqlc** was intended for use primarily with Windows 9x/Me. It does not support the updated authentication protocol used beginning with MySQL 4.1, and is not supported in MySQL 4.1 and above. Beginning with MySQL 4.1.8, it is no longer included in MySQL Windows distributions.

To use **mysqlc**, you must have a copy of the `cygwinb19.dll` library installed somewhere that **mysqlc** can find it. If your distribution does not have the `cygwinb19.dll` library in the `bin` directory under the base directory of your MySQL installation, look for it in the `lib` directory and copy it to your Windows system directory (`\Windows\system` or a similar place).

2.3.14. Troubleshooting a MySQL Installation Under Windows

When installing and running MySQL for the first time, you may encounter certain errors that prevent the MySQL server from starting. The purpose of this section is to help you diagnose and correct some of

these errors.

Your first resource when troubleshooting server issues is the error log. The MySQL server uses the error log to record information relevant to the error that is preventing the server from starting. The error log is located in the data directory specified in your `my.ini` file. The default data directory location is `C:\mysql\data`. See Section 5.9.1, “The Error Log”.

Another source of information regarding possible errors is the console messages displayed when the MySQL service is starting. Use the **NET START mysql** command from the command line after installing **mysqld** as a service to see any error messages regarding the starting of the MySQL server as a service. See Section 2.3.12, “Starting MySQL as a Windows Service”.

The following are examples of some of the more common error messages you may encounter when installing MySQL and starting the server for the first time:

```
System error 1067 has occurred.
Fatal error: Can't open privilege tables: Table 'mysql.host' doesn't exist
```

These messages occur when the MySQL server cannot find the `mysql` privileges database or other critical files. This error is often encountered when the MySQL base or data directories are installed in different locations than the default locations (`C:\mysql` and `C:\mysql\data`, respectively).

If you have installed MySQL to a directory other than `C:\mysql` you need to ensure that the MySQL server is aware of this through the use of a configuration (`my.ini`) file. The `my.ini` file needs to be located in your Windows directory, typically located at `C:\WinNT` or `C:\WINDOWS`. You can determine its exact location from the value of the `WINDIR` environment variable by issuing the following command from the command prompt:

```
C:\> echo %WINDIR%
```

An option file can be created and modified with any text editor, such as the Notepad program. For example, if MySQL is installed at `E:\mysql` and the data directory is located at `D:\MySQLdata`, you can create the option file and set up a `[mysqld]` section to specify values for the basedir and datadir parameters:

```
[mysqld]
# set basedir to your installation path
basedir=E:/mysql
# set datadir to the location of your data directory
datadir=D:/MySQLdata
```

Note that Windows pathnames are specified in option files using forward slashes rather than backslashes. If you do use backslashes, you must double them:

```
[mysqld]
# set basedir to your installation path
basedir=C:\\Program Files\\mysql
# set datadir to the location of your data directory
datadir=D:\\MySQLdata
```

See Section 2.3.8, “Creating an Option File”.

```
Error: Cannot create Windows service for MySql. Error: 0
```

This error is encountered when you re-install or upgrade MySQL without first stopping and removing the existing MySQL service and install MySQL using the MySQL Configuration Wizard. This happens

because when the Configuration Wizard tries to install the service it finds an existing service with the same name.

One solution to this problem is to choose a service name other than `mysql` when using the configuration wizard. This will allow the new service to be installed correctly, but leaves the outdated service in place. While this is harmless it is best to remove old services that are no longer in use.

To permanently remove the old mysql service, execute the following command as a user with administrative privileges, on the command-line:

```
C:\>sc delete mysql
[SC] DeleteService SUCCESS
```

2.3.15. Upgrading MySQL on Windows

This section lists some of the steps you should take when upgrading MySQL on Windows.

1. You should always back up your current MySQL installation before performing an upgrade. See Section 5.7.1, “Database Backups”.

2. Download the latest Windows distribution of MySQL from http://dev.mysql.com.

3. Before upgrading MySQL, you must stop the server.

   If the server is installed as a service, stop the service with the following command from the command prompt:

   ```
   C:\> NET STOP MySQL
   ```

   If you are not running the MySQL server as a service, use the following command to stop the server:

   ```
   C:\> C:\mysql\bin\mysqladmin -u root shutdown
   ```

4. Exit the `WinMySQLAdmin` program if it is running.

5. When upgrading to MySQL 4.1.5 or higher from a previous version, or when upgrading from a version of MySQL installed from a Zip archive to a version of MySQL installed with the MySQL Installation Wizard, you must manually remove the previous installation and MySQL service (if the server is installed as a service).

   To remove the MySQL service, use the following command:

   ```
   C:\> C:\mysql\bin\mysqld --remove
   ```

   If you do not remove the existing service, the MySQL Installation Wizard may fail to properly install the new MySQL service.

6. If you are using the MySQL Installation Wizard, start the wizard as described in Section 2.3.4, “Using the MySQL Installation Wizard”.

7. If you are installing MySQL from a Zip archive, extract the archive. You may either overwrite your existing MySQL installation (usually located at `C:\mysql`), or install it into a different directory, such as `C:\mysql4`. Overwriting the existing installation is recommended.

8. Restart the server. For example, use **NET START MySQL** if you run MySQL as a service, or invoke **mysqld** directly otherwise.

9. Refer to Section 2.10, “Upgrading MySQL” for additional information on upgrading MySQL that is not specific to Windows.

10. If you encounter errors, see Section 2.3.14, “Troubleshooting a MySQL Installation Under Windows”.

2.3.16. MySQL on Windows Compared to MySQL on Unix

MySQL for Windows has proven itself to be very stable. The Windows version of MySQL has the same features as the corresponding Unix version, with the following exceptions:

- **Windows 95 and threads**

  Windows 95 leaks about 200 bytes of main memory for each thread creation. Each connection in MySQL creates a new thread, so you shouldn't run **mysqld** for an extended time on Windows 95 if your server handles many connections! Other versions of Windows don't suffer from this bug.

- **Limited number of ports**

  Windows systems have about 4,000 ports available for client connections, and after a connection on a port closes, it takes two to four minutes before the port can be reused. In situations where clients connect to and disconnect from the server at a high rate, it is possible for all available ports to be used up before closed ports become available again. If this happens, the MySQL server appears to be unresponsive even though it is running. Note that ports may be used by other applications running on the machine as well, in which case the number of ports available to MySQL is lower.

  For more information, see http://support.microsoft.com/default.aspx?scid=kb;en-us;196271.

- **Concurrent reads**

  MySQL depends on the `pread()` and `pwrite()` calls to be able to mix `INSERT` and `SELECT`. Currently we use mutexes to emulate `pread()`/`pwrite()`. We will, in the long run, replace the file level interface with a virtual interface so that we can use the `readfile()`/`writefile()` interface on NT, 2000, and XP to get more speed. The current implementation limits the number of open files MySQL can use to 2,048 (1,024 before MySQL 4.0.19), which means that you cannot run as many concurrent threads on NT, 2000, XP, and 2003 as on Unix.

- **Blocking read**

  MySQL uses a blocking read for each connection, which has the following implications if named pipe connections are enabled:

  - A connection is not disconnected automatically after eight hours, as happens with the Unix version of MySQL.
  - If a connection hangs, it's impossible to break it without killing MySQL.
  - **mysqladmin kill** does not work on a sleeping connection.
  - **mysqladmin shutdown** can't abort as long as there are sleeping connections.

  We plan to fix this problem when our Windows developers have figured out a nice workaround.

- **`ALTER TABLE`**

  While you are executing an `ALTER TABLE` statement, the table is locked from being used by other threads. This has to do with the fact that on Windows, you can't delete a file that is in use by another thread. In the future, we may find some way to work around this problem.

- **`DROP TABLE`**

  `DROP TABLE` on a table that is in use by a `MERGE` table does not work on Windows because the `MERGE` handler does the table mapping hidden from the upper layer of MySQL. Because Windows doesn't allow you to drop files that are open, you first must flush all `MERGE` tables (with `FLUSH TABLES`) or drop the `MERGE` table before dropping the table. We will fix this at the same time we introduce views.

- **`DATA DIRECTORY` and `INDEX DIRECTORY`**

  The `DATA DIRECTORY` and `INDEX DIRECTORY` options for `CREATE TABLE` are ignored on Windows, because Windows doesn't support symbolic links. These options also are ignored on systems that have a non-functional `realpath()` call.

- **`DROP DATABASE`**

  You cannot drop a database that is in use by some thread.

- **Killing MySQL from the Task Manager**

  You cannot kill MySQL from the Task Manager or with the shutdown utility in Windows 95. You must stop it with **mysqladmin shutdown**.

- **Case-insensitive names**

  Filenames are not case sensitive on Windows, so MySQL database and table names are also not case sensitive on Windows. The only restriction is that database and table names must be specified using the same case throughout a given statement. See Section 9.2.2, “Identifier Case Sensitivity”.

- **The '\' pathname separator character**

  Pathname components in Windows are separated by the '\' character, which is also the escape character in MySQL. If you are using `LOAD DATA INFILE` or `SELECT ... INTO OUTFILE`, use Unix-style filenames with '/' characters:

  ```
  mysql> LOAD DATA INFILE 'C:/tmp/skr.txt' INTO TABLE skr;
  mysql> SELECT * INTO OUTFILE 'C:/tmp/skr.txt' FROM skr;
  ```

  Alternatively, you must double the '\' character:

  ```
  mysql> LOAD DATA INFILE 'C:\\tmp\\skr.txt' INTO TABLE skr;
  mysql> SELECT * INTO OUTFILE 'C:\\tmp\\skr.txt' FROM skr;
  ```

- **Problems with pipes.**

  Pipes do not work reliably from the Windows command-line prompt. If the pipe includes the character `^Z` / `CHAR(24)`, Windows thinks it has encountered end-of-file and aborts the program.

  This is mainly a problem when you try to apply a binary log as follows:

```
C:\> mysqlbinlog binary-log-name | mysql --user=root
```

If you have a problem applying the log and suspect that it is because of a `^Z` / `CHAR(24)` character, you can use the following workaround:

```
C:\> mysqlbinlog binary-log-file --result-file=/tmp/bin.sql
C:\> mysql --user=root --execute "source /tmp/bin.sql"
```

The latter command also can be used to reliably read in any SQL file that may contain binary data.

- **`Access denied for user` error**

  If you attempt to run a MySQL client program to connect to a server running on the same machine, but get the error `Access denied for user '`*`some-user`*`'@'unknown' to database 'mysql'`, this means that MySQL cannot resolve your hostname properly.

  To fix this, you should create a file named `\windows\hosts` containing the following information:

```
127.0.0.1       localhost
```

Here are some open issues for anyone who might want to help us improve MySQL on Windows:

PD: Commented this one out as obsolete until I hear otherwise

- Make a single-user `MYSQL.DLL` server. This should include everything in a standard MySQL server, except thread creation. This makes MySQL much easier to use in applications that don't need a true client/server and don't need to access the server from other hosts.
- Add some nice start and shutdown icons to the MySQL installation.
- It would be really nice to be able to kill **mysqld** from the Task Manager in Windows 95. For the moment, you must use **mysqladmin shutdown**.

- Port `readline` to Windows for use in the **mysql** command-line tool.
- GUI versions of the standard MySQL clients (**mysql**, **mysqlshow**, **mysqladmin**, and **mysqldump**) would be nice.
- It would be nice if the socket read and write functions in `net.c` were interruptible. This would make it possible to kill open threads with **mysqladmin kill** on Windows.
- Add macros to use the faster thread-safe increment/decrement methods provided by Windows.

2.4. Installing MySQL on Linux

The recommended way to install MySQL on Linux is by using the RPM packages. The MySQL RPMs are currently built on a SuSE Linux 7.3 system, but should work on most versions of Linux that support **rpm** and use `glibc`. To obtain RPM packages, see Section 2.1.3, “How to Get MySQL”.

Note: RPM distributions of MySQL often are provided by other vendors. Be aware that they may differ in features and capabilities from those built by MySQL AB, and that the instructions in this manual do not necessarily apply to installing them. The vendor's instructions should be consulted instead.

If you have problems with an RPM file (for example, if you receive the error ``Sorry, the host '*xxxx*' could not be looked up''), see Section 2.12.1.2, “Linux Binary Distribution Notes”.

In most cases, you only need to install the `MySQL-server` and `MySQL-client` packages to get a functional MySQL installation. The other packages are not required for a standard installation. If you want to run a MySQL-Max server that has additional capabilities, you should also install the `MySQL-Max` RPM. However, you should do so only *after* installing the `MySQL-server` RPM. See Section 5.1.2, “The **mysqld-max** Extended MySQL Server”.

If you get a dependency failure when trying to install the MySQL 4.0 packages (for example, ``error: removing these packages would break dependencies: libmysqlclient.so.10 is needed by ...''), you should also install the package `MySQL-shared-compat`, which includes both the shared libraries for backward compatibility (`libmysqlclient.so.12` for MySQL 4.0 and `libmysqlclient.so.10` for MySQL 3.23).

Many Linux distributions still ship with MySQL 3.23 and they usually link applications dynamically to save disk space. If these shared libraries are in a separate package (for example, `MySQL-shared`), it is sufficient to simply leave this package installed and just upgrade the MySQL server and client packages (which are statically linked and do not depend on the shared libraries). For distributions that include the shared libraries in the same package as the MySQL server (for example, Red Hat Linux), you could either install our 3.23 `MySQL-shared` RPM, or use the `MySQL-shared-compat` package instead.

The following RPM packages are available:

- `MySQL-server-`*`VERSION`*`.i386.rpm`

  The MySQL server. You need this unless you only want to connect to a MySQL server running on another machine. Note: Server RPM files were called `MySQL-`*`VERSION`*`.i386.rpm` before MySQL 4.0.10. That is, they did not have `-server` in the name.

- `MySQL-Max-`*`VERSION`*`.i386.rpm`

  The MySQL-Max server. This server has additional capabilities that the one provided in the `MySQL-server` RPM does not. You must install the `MySQL-server` RPM first, because the `MySQL-Max` RPM depends on it.

- `MySQL-client-VERSION.i386.rpm`

  The standard MySQL client programs. You probably always want to install this package.

- `MySQL-bench-VERSION.i386.rpm`

  Tests and benchmarks. Requires Perl and the `DBD::mysql` module.

- `MySQL-devel-VERSION.i386.rpm`

  The libraries and include files that are needed if you want to compile other MySQL clients, such as the Perl modules.

- `MySQL-shared-VERSION.i386.rpm`

  This package contains the shared libraries (`libmysqlclient.so*`) that certain languages and applications need to dynamically load and use MySQL.

- `MySQL-shared-compat-VERSION.i386.rpm`

  This package includes the shared libraries for both MySQL 3.23 and MySQL 4.0. Install this package instead of `MySQL-shared` if you have applications installed that are dynamically linked against MySQL 3.23 but you want to upgrade to MySQL 4.0 without breaking the library dependencies. This package has been available since MySQL 4.0.13.

- `MySQL-embedded-VERSION.i386.rpm`

  The embedded MySQL server library (from MySQL 4.0).

- `MySQL-VERSION.src.rpm`

  This contains the source code for all of the previous packages. It can also be used to rebuild the RPMs on other architectures (for example, Alpha or SPARC).

To see all files in an RPM package (for example, a `MySQL-server` RPM), run:

```
shell> rpm -qpl MySQL-server-VERSION.i386.rpm
```

To perform a standard minimal installation, run:

```
shell> rpm -i MySQL-server-VERSION.i386.rpm
shell> rpm -i MySQL-client-VERSION.i386.rpm
```

To install just the client package, run:

```
shell> rpm -i MySQL-client-VERSION.i386.rpm
```

RPM provides a feature to verify the integrity and authenticity of packages before installing them. If you would like to learn more about this feature, see Section 2.1.4, “Verifying Package Integrity Using MD5 Checksums or `GnuPG`”.

The server RPM places data under the `/var/lib/mysql` directory. The RPM also creates a login account for a user named `mysql` (if one does not exist) to use for running the MySQL server, and creates the appropriate entries in `/etc/init.d/` to start the server automatically at boot time. (This means that if you have performed a previous installation and have made changes to its startup script, you may want to make a copy of the script so that you don't lose it when you install a newer RPM.) See Sec-

tion 2.9.2.2, “Starting and Stopping MySQL Automatically” for more information on how MySQL can be started automatically on system startup.

If you want to install the MySQL RPM on older Linux distributions that do not support initialization scripts in `/etc/init.d` (directly or via a symlink), you should create a symbolic link that points to the location where your initialization scripts actually are installed. For example, if that location is `/etc/rc.d/init.d`, use these commands before installing the RPM to create `/etc/init.d` as a symbolic link that points there:

```
shell> cd /etc
shell> ln -s rc.d/init.d .
```

However, all current major Linux distributions should support the new directory layout that uses `/etc/init.d`, because it is required for LSB (Linux Standard Base) compliance.

If the RPM files that you install include `MySQL-server`, the **mysqld** server should be up and running after installation. You should be able to start using MySQL.

If something goes wrong, you can find more information in the binary installation section. See Section 2.7, “Installing MySQL on Other Unix-Like Systems”.

Note: The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.9, “Post-Installation Setup and Testing”.

2.5. Installing MySQL on Mac OS X

Beginning with MySQL 4.0.11, you can install MySQL on Mac OS X 10.2.x (``Jaguar'') and up using a Mac OS X binary package in PKG format instead of the binary tarball distribution. Please note that older versions of Mac OS X (for example, 10.1.x) are not supported by this package.

The package is located inside a disk image (`.dmg`) file that you first need to mount by double-clicking its icon in the Finder. It should then mount the image and display its contents.

To obtain MySQL, see Section 2.1.3, “How to Get MySQL”.

Note: Before proceeding with the installation, be sure to shut down all running MySQL server instances by either using the MySQL Manager Application (on Mac OS X Server) or via **mysqladmin shutdown** on the command line.

To actually install the MySQL PKG file, double-click on the package icon. This launches the Mac OS X Package Installer, which guides you through the installation of MySQL.

Due to a bug in the Mac OS X package installer, you may see this error message in the destination disk selection dialog:

```
You cannot install this software on this disk. (null)
```

If this error occurs, simply click the `Go Back` button once to return to the previous screen. Then click `Continue` to advance to the destination disk selection again, and you should be able to choose the destination disk correctly. We have reported this bug to Apple and it is investigating this problem.

The Mac OS X PKG of MySQL installs itself into `/usr/local/mysql-`*`VERSION`* and also installs a symbolic link, `/usr/local/mysql`, pointing to the new location. If a directory named `/usr/local/mysql` exists, it is renamed to `/usr/local/mysql.bak` first. Additionally, the installer creates the grant tables in the `mysql` database by executing **mysql_install_db** after the installation.

The installation layout is similar to that of a **tar** file binary distribution; all MySQL binaries are located in the directory `/usr/local/mysql/bin`. The MySQL socket file is created as `/tmp/mysql.sock` by default. See Section 2.1.5, “Installation Layouts”.

MySQL installation requires a Mac OS X user account named `mysql`. A user account with this name should exist by default on Mac OS X 10.2 and up.

If you are running Mac OS X Server, you have a version of MySQL installed. The versions of MySQL that ship with Mac OS X Server versions are shown in the following table:

Mac OS X Server Version	MySQL Version
10.2-10.2.2	3.23.51
10.2.3-10.2.6	3.23.53
10.3	4.0.14
10.3.2	4.0.16

This manual section covers the installation of the official MySQL Mac OS X PKG only. Make sure to read Apple's help information about installing MySQL: Run the ``Help View'' application, select ``Mac OS X Server'' help, do a search for ``MySQL,'' and read the item entitled ``Installing MySQL.''

For pre-installed versions of MySQL on Mac OS X Server, note especially that you should start **mysqld** with **safe_mysqld** instead of **mysqld_safe** if MySQL is older than version 4.0.

If you previously used Marc Liyanage's MySQL packages for Mac OS X from http://www.entropy.ch, you can simply follow the update instructions for packages using the binary installation layout as given on his pages.

If you are upgrading from Marc's 3.23.xx versions or from the Mac OS X Server version of MySQL to the official MySQL PKG, you also need to convert the existing MySQL privilege tables to the current format, because some new security privileges have been added. See Section 2.10.7, “Upgrading the Grant Tables”.

If you would like to automatically start up MySQL during system startup, you also need to install the MySQL Startup Item. Starting with MySQL 4.0.15, it is part of the Mac OS X installation disk images as a separate installation package. Simply double-click the `MySQLStartupItem.pkg` icon and follow the instructions to install it.

Note that the Startup Item need be installed only once! There is no need to install it each time you upgrade the MySQL package later.

The Startup Item is installed into `/Library/StartupItems/MySQLCOM`. (Before MySQL 4.1.2, the location was `/Library/StartupItems/MySQL`, but that collided with the MySQL Startup Item installed by Mac OS X Server.) Startup Item installation adds a variable `MYSQLCOM=-YES-` to the system configuration file `/etc/hostconfig`. If you would like to disable the automatic startup of MySQL, simply change this variable to `MYSQLCOM=-NO-`.

On Mac OS X Server, the default MySQL installation uses the variable `MYSQL` in the `/etc/hostconfig` file. The MySQL AB Startup Item installer disables this variable by setting it to `MYSQL=-NO-`. This avoids boot time conflicts with the `MYSQLCOM` variable used by the MySQL AB Startup Item. However, it does not shut down a running MySQL server. You should do that yourself.

After the installation, you can start up MySQL by running the following commands in a terminal window. You must have administrator privileges to perform this task.

If you have installed the Startup Item:

```
shell> sudo /Library/StartupItems/MySQLCOM/MySQLCOM start
(Enter your password, if necessary)
(Press Control-D or enter "exit" to exit the shell)
```

For versions of MySQL older than 4.1.3, substitute `/Library/StartupItems/MySQLCOM/MySQLCOM` with `/Library/StartupItems/MySQL/MySQL` above.

If you don't use the Startup Item, enter the following command sequence:

```
shell> cd /usr/local/mysql
shell> sudo ./bin/mysqld_safe
(Enter your password, if necessary)
(Press Control-Z)
shell> bg
(Press Control-D or enter "exit" to exit the shell)
```

You should be able to connect to the MySQL server, for example, by running `/usr/local/mysql/bin/mysql`.

Note: The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.9, “Post-Installation Setup and Testing”.

You might want to add aliases to your shell's resource file to make it easier to access commonly used programs such as **mysql** and **mysqladmin** from the command line. The syntax for **tcsh** is:

```
alias mysql /usr/local/mysql/bin/mysql
alias mysqladmin /usr/local/mysql/bin/mysqladmin
```

For **bash**, use:

```
alias mysql=/usr/local/mysql/bin/mysql
alias mysqladmin=/usr/local/mysql/bin/mysqladmin
```

Even better, add `/usr/local/mysql/bin` to your `PATH` environment variable. For example, add the following line to your `$HOME/.tcshrc` file if your shell is **tcsh**:

```
setenv PATH ${PATH}:/usr/local/mysql/bin
```

If no `.tcshrc` file exists in your home directory, create it with a text editor.

If you are upgrading an existing installation, please note that installing a new MySQL PKG does not remove the directory of an older installation. Unfortunately, the Mac OS X Installer does not yet offer the functionality required to properly upgrade previously installed packages.

To use your existing databases with the new installation, you'll need to copy the contents of the old data directory to the new data directory. Make sure that neither the old server nor the new one is running when you do this. After you have copied over the MySQL database files from the previous installation and have successfully started the new server, you should consider removing the old installation files to save disk space. Additionally, you should also remove older versions of the Package Receipt directories located in `/Library/Receipts/mysql-VERSION.pkg`.

2.6. Installing MySQL on NetWare

Porting MySQL to NetWare was an effort spearheaded by Novell. Novell customers should be pleased to note that NetWare 6.5 ships with bundled MySQL binaries, complete with an automatic commercial use license for all servers running that version of NetWare.

MySQL for NetWare is compiled using a combination of `Metrowerks CodeWarrior for NetWare` and special cross-compilation versions of the GNU autotools.

The latest binary packages for NetWare can be obtained at http://dev.mysql.com/downloads/. See Section 2.1.3, “How to Get MySQL”.

In order to host MySQL, the NetWare server must meet these requirements:

- NetWare 6.5 Support Pack 2 installed and updated with the latest LibC, or NetWare 6.0 with Support Pack 4 installed and updated with the latest LibC. NetWare 6.5 Support Pack 2 and other updates are available at http://support.novell.com/filefinder/18197/index.html. NetWare 6.0 Support Pack 4 and other updates are available at http://support.novell.com/filefinder/13659/index.html. The latest LibC is available at http://developer.novell.com/ndk/libc.htm. Steps to update LibC can be found here: http://developer.novell.com/ndk/doc/libc/index.html?page=/ndk/doc/libc/libc_enu/data/ajjl0r0.html
- The system must meet Novell's minimum requirements to run the respective version of NetWare.
- MySQL data, as well as the binaries themselves, must be installed on an NSS volume; traditional volumes are not supported.

To install MySQL for NetWare, use the following procedure:

1. If you are upgrading from a prior installation, stop the MySQL server. This is done from the server console, using the following command:

   ```
   SERVER:  mysqladmin -u root shutdown
   ```

2. Log on to the target server from a client machine with access to the location where you are installing MySQL.
3. Extract the binary package Zip file onto the server. Be sure to allow the paths in the Zip file to be used. It is safe to simply extract the file to `SYS:\`.

   If you are upgrading from a prior installation, you may need to copy the data directory (for example, `SYS:MYSQL\DATA`), as well as `my.cnf`, if you have customized it. You can then delete the old copy of MySQL.
4. You might want to rename the directory to something more consistent and easy to use. We recommend using `SYS:MYSQL`; examples in this manual use this name to refer to the installation directory in general.
5. At the server console, add a search path for the directory containing the MySQL NLMs. For example:

   ```
   SERVER:  SEARCH ADD SYS:MYSQL\BIN
   ```

6. Initialize the data directory and the grant tables, if needed, by executing **mysql_install_db** at the server console.
7. Start the MySQL server using **mysqld_safe** at the server console.

8. To finish the installation, you should also add the following commands to `autoexec.ncf`. For example, if your MySQL installation is in `SYS:MYSQL` and you want MySQL to start automatically, you could add these lines:

   ```
   #Starts the MySQL 4.0.x database server
   SEARCH ADD SYS:MYSQL\BIN
   MYSQLD_SAFE
   ```

   If you are running MySQL on NetWare 6.0, we strongly suggest that you use the `--skip-external-locking` option on the command line:

   ```
   #Starts the MySQL 4.0.x database server
   SEARCH ADD SYS:MYSQL\BIN
   MYSQLD_SAFE --skip-external-locking
   ```

   It is also necessary to use `CHECK TABLE` and `REPAIR TABLE` instead of **myisamchk**, because **myisamchk** makes use of external locking. External locking is known to have problems on NetWare 6.0; the problem has been eliminated in NetWare 6.5.

   mysqld_safe on NetWare provides a screen presence. When you unload (shut down) the **mysqld_safe** NLM, the screen does not by default go away. Instead, it prompts for user input:

   ```
   *<NLM has terminated; Press any key to close the screen>*
   ```

   If you want NetWare to close the screen automatically instead, use the `--autoclose` option to **mysqld_safe**. For example:

   ```
   #Starts the MySQL 4.0.x database server
   SEARCH ADD SYS:MYSQL\BIN
   MYSQLD_SAFE --autoclose
   ```

9. When installing the 4.1.x version either for the first time or upgrading the 4.0.x version to 4.1.x, download and install Perl module for MySQL 4.1 from http://forge.novell.com/modules/xfmod/project/showfiles.php?group_id=1126 and PHP Extension for MySQL 4.1 from http://forge.novell.com/modules/xfmod/project/showfiles.php?group_id=1078

The behavior of **mysqld_safe** on NetWare is described further in Section 5.1.3, “The **mysqld_safe** Server Startup Script”.

If there was an existing installation of MySQL on the server, be sure to check for existing MySQL startup commands in `autoexec.ncf`, and edit or delete them as necessary.

Note: The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.9, “Post-Installation Setup and Testing”.

2.7. Installing MySQL on Other Unix-Like Systems

This section covers the installation of MySQL binary distributions that are provided for various platforms in the form of compressed **tar** files (files with a `.tar.gz` extension). See Section 2.1.2.5, “MySQL Binaries Compiled by MySQL AB” for a detailed list.

To obtain MySQL, see Section 2.1.3, “How to Get MySQL”.

MySQL **tar** file binary distributions have names of the form `mysql-VERSION-OS.tar.gz`, where *VERSION* is a number (for example, `4.0.17`), and *OS* indicates the type of operating system for which the distribution is intended (for example, `pc-linux-i686`).

In addition to these generic packages, we also offer binaries in platform-specific package formats for selected platforms. See Section 2.2, “Standard MySQL Installation Using a Binary Distribution” for more information on how to install these.

You need the following tools to install a MySQL **tar** file binary distribution:

- GNU `gunzip` to uncompress the distribution.
- A reasonable **tar** to unpack the distribution. GNU **tar** is known to work. Some operating systems come with a pre-installed version of **tar** that is known to have problems. For example, Mac OS X **tar** and Sun **tar** are known to have problems with long filenames. On Mac OS X, you can use the pre-installed **gnutar** program. On other systems with a deficient **tar**, you should install GNU **tar** first.

If you run into problems, *please always use* ***mysqlbug*** when posting questions to a MySQL mailing list. Even if the problem isn't a bug, **mysqlbug** gathers system information that helps others solve your problem. By not using **mysqlbug**, you lessen the likelihood of getting a solution to your problem. You can find **mysqlbug** in the `bin` directory after you unpack the distribution. See Section 1.4.1.3, “How to Report Bugs or Problems”.

The basic commands you must execute to install and use a MySQL binary distribution are:

```
shell> groupadd mysql
shell> useradd -g mysql mysql
shell> cd /usr/local
shell> gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
shell> ln -s full-path-to-mysql-VERSION-OS mysql
shell> cd mysql
shell> scripts/mysql_install_db --user=mysql
shell> chown -R root  .
shell> chown -R mysql data
shell> chgrp -R mysql .
shell> bin/mysqld_safe --user=mysql &
```

For versions of MySQL older than 4.0, substitute **bin/safe_mysqld** for **bin/mysqld_safe** in the final command.

Note: This procedure does not set up any passwords for MySQL accounts. After following the procedure, proceed to Section 2.9, “Post-Installation Setup and Testing”.

A more detailed version of the preceding description for installing a binary distribution follows:

1. Add a login user and group for **mysqld** to run as:

   ```
   shell> groupadd mysql
   shell> useradd -g mysql mysql
   ```

   These commands add the `mysql` group and the `mysql` user. The syntax for **useradd** and **groupadd** may differ slightly on different versions of Unix. They may also be called **adduser** and

addgroup.

You might want to call the user and group something else instead of `mysql`. If so, substitute the appropriate name in the following steps.

2. Pick the directory under which you want to unpack the distribution, and change location into it. In the following example, we unpack the distribution under `/usr/local`. (The instructions, therefore, assume that you have permission to create files and directories in `/usr/local`. If that directory is protected, you need to perform the installation as `root`.)

   ```
   shell> cd /usr/local
   ```

3. Obtain a distribution file from one of the sites listed in Section 2.1.3, “How to Get MySQL”. For a given release, binary distributions for all platforms are built from the same MySQL source distribution.

4. Unpack the distribution, which creates the installation directory. Then create a symbolic link to that directory:

   ```
   shell> gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
   shell> ln -s full-path-to-mysql-VERSION-OS mysql
   ```

   The **tar** command creates a directory named `mysql-VERSION-OS`. The `ln` command makes a symbolic link to that directory. This lets you refer more easily to the installation directory as `/usr/local/mysql`.

   With GNU **tar**, no separate invocation of `gunzip` is necessary. You can replace the first line with the following alternative command to uncompress and extract the distribution:

   ```
   shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
   ```

5. Change location into the installation directory:

   ```
   shell> cd mysql
   ```

   You can find several files and subdirectories in the `mysql` directory. The most important for installation purposes are the `bin` and `scripts` subdirectories.

   - `bin`

     This directory contains client programs and the server. You should add the full pathname of this directory to your `PATH` environment variable so that your shell finds the MySQL programs properly. See Appendix F, *Environment Variables*.

   - `scripts`

     This directory contains the **mysql_install_db** script used to initialize the `mysql` database containing the grant tables that store the server access permissions.

6. If you haven't installed MySQL before, you must create the MySQL grant tables:

   ```
   shell> scripts/mysql_install_db --user=mysql
   ```

   If you run the command as `root`, you should use the `--user` option as shown. The value of the option should be the name of the login account that you created in the first step to use for running

the server. If you run the command while logged in as that user, you can omit the `--user` option.

Note that for MySQL versions older than 3.22.10, **mysql_install_db** left the server running after creating the grant tables. This is no longer true; you need to restart the server after performing the remaining steps in this procedure.

7. Change the ownership of program binaries to `root` and ownership of the data directory to the user that you run **mysqld** as. Assuming that you are located in the installation directory (`/usr/local/mysql`), the commands look like this:

```
shell> chown -R root  .
shell> chown -R mysql data
shell> chgrp -R mysql .
```

   The first command changes the owner attribute of the files to the `root` user. The second changes the owner attribute of the data directory to the `mysql` user. The third changes the group attribute to the `mysql` group.

8. If you would like MySQL to start automatically when you boot your machine, you can copy `support-files/mysql.server` to the location where your system has its startup files. More information can be found in the `support-files/mysql.server` script itself and in Section 2.9.2.2, “Starting and Stopping MySQL Automatically”.

9. You can set up new accounts using the **bin/mysql_setpermission** script if you install the `DBI` and `DBD::mysql` Perl modules. For instructions, see Section 2.13, “Perl Installation Notes”.

10. If you would like to use **mysqlaccess** and have the MySQL distribution in some non-standard place, you must change the location where **mysqlaccess** expects to find the **mysql** client. Edit the `bin/mysqlaccess` script at approximately line 18. Search for a line that looks like this:

```
$MYSQL     = '/usr/local/bin/mysql';    # path to mysql executable
```

   Change the path to reflect the location where `mysql` actually is stored on your system. If you do not do this, you get a `Broken pipe` error when you run **mysqlaccess**.

After everything has been unpacked and installed, you should test your distribution.

You can start the MySQL server with the following command:

```
shell> bin/mysqld_safe --user=mysql &
```

For versions of MySQL older than 4.0, substitute **bin/safe_mysqld** for **bin/mysqld_safe** in the command.

More information about **mysqld_safe** is given in Section 5.1.3, “The **mysqld_safe** Server Startup Script”.

Note: The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.9, “Post-Installation Setup and Testing”.

2.8. MySQL Installation Using a Source Distribution

Before you proceed with the source installation, check first to see whether our binary is available for your platform and whether it works for you. We put a lot of effort into making sure that our binaries are built with the best possible options.

To obtain a source distribution for MySQL, Section 2.1.3, “How to Get MySQL”.

MySQL source distributions are provided as compressed **tar** archives and have names of the form `mysql-VERSION.tar.gz`, where *`VERSION`* is a number like `5.0.3-alpha`.

You need the following tools to build and install MySQL from source:

- GNU `gunzip` to uncompress the distribution.
- A reasonable **tar** to unpack the distribution. GNU **tar** is known to work. Some operating systems come with a pre-installed version of **tar** that is known to have problems. For example, Mac OS X **tar** and Sun **tar** are known to have problems with long filenames. On Mac OS X, you can use the pre-installed **gnutar** program. On other systems with a deficient **tar**, you should install GNU **tar** first.
- A working ANSI C++ compiler. **gcc** 2.95.2 or later, **egcs** 1.0.2 or later or **egcs 2.91.66**, SGI C++, and SunPro C++ are some of the compilers that are known to work. `libg++` is not needed when using **gcc**. **gcc** 2.7.x has a bug that makes it impossible to compile some perfectly legal C++ files, such as `sql/sql_base.cc`. If you have only **gcc** 2.7.x, you must upgrade your **gcc** to be able to compile MySQL. **gcc** 2.8.1 is also known to have problems on some platforms, so it should be avoided if a new compiler exists for the platform.

  gcc 2.95.2 or later is recommended when compiling MySQL 3.23.x.
- A good **make** program. GNU **make** is always recommended and is sometimes required. If you have problems, we recommend trying GNU **make** 3.75 or newer.

If you are using a version of **gcc** recent enough to understand the `-fno-exceptions` option, it is *very important* that you use this option. Otherwise, you may compile a binary that crashes randomly. We also recommend that you use `-felide-constructors` and `-fno-rtti` along with `-fno-exceptions`. When in doubt, do the following:

```
CFLAGS="-O3" CXX=gcc CXXFLAGS="-O3 -felide-constructors \
       -fno-exceptions -fno-rtti" ./configure \
       --prefix=/usr/local/mysql --enable-assembler \
       --with-mysqld-ldflags=-all-static
```

On most systems, this gives you a fast and stable binary.

If you run into problems, *please always use **mysqlbug*** when posting questions to a MySQL mailing list. Even if the problem isn't a bug, **mysqlbug** gathers system information that helps others solve your problem. By not using **mysqlbug**, you lessen the likelihood of getting a solution to your problem. You can find **mysqlbug** in the `scripts` directory after you unpack the distribution. See Section 1.4.1.3, “How to Report Bugs or Problems”.

2.8.1. Source Installation Overview

The basic commands you must execute to install a MySQL source distribution are:

```
shell> groupadd mysql
shell> useradd -g mysql mysql
shell> gunzip < mysql-VERSION.tar.gz | tar -xvf -
```

```
shell> cd mysql-VERSION
shell> ./configure --prefix=/usr/local/mysql
shell> make
shell> make install
shell> cp support-files/my-medium.cnf /etc/my.cnf
shell> cd /usr/local/mysql
shell> bin/mysql_install_db --user=mysql
shell> chown -R root  .
shell> chown -R mysql var
shell> chgrp -R mysql .
shell> bin/mysqld_safe --user=mysql &
```

For versions of MySQL older than 4.0, substitute **bin/safe_mysqld** for **bin/mysqld_safe** in the final command.

If you start from a source RPM, do the following:

```
shell> rpmbuild --rebuild --clean MySQL-VERSION.src.rpm
```

This makes a binary RPM that you can install. For older versions of RPM, you may have to replace the command **rpmbuild** with **rpm** instead.

Note: This procedure does not set up any passwords for MySQL accounts. After following the procedure, proceed to Section 2.9, “Post-Installation Setup and Testing”, for post-installation setup and testing.

A more detailed version of the preceding description for installing MySQL from a source distribution follows:

1. Add a login user and group for **mysqld** to run as:

   ```
   shell> groupadd mysql
   shell> useradd -g mysql mysql
   ```

   These commands add the `mysql` group and the `mysql` user. The syntax for **useradd** and **groupadd** may differ slightly on different versions of Unix. They may also be called **adduser** and **addgroup**.

   You might want to call the user and group something else instead of `mysql`. If so, substitute the appropriate name in the following steps.

2. Pick the directory under which you want to unpack the distribution, and change location into it.

3. Obtain a distribution file from one of the sites listed in Section 2.1.3, “How to Get MySQL”.

4. Unpack the distribution into the current directory:

   ```
   shell> gunzip < /path/to/mysql-VERSION.tar.gz | tar xvf -
   ```

   This command creates a directory named `mysql-VERSION`.

   With GNU **tar**, no separate invocation of `gunzip` is necessary. You can use the following alternative command to uncompress and extract the distribution:

   ```
   shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
   ```

5. Change location into the top-level directory of the unpacked distribution:

```
shell> cd mysql-VERSION
```

Note that currently you must configure and build MySQL from this top-level directory. You cannot build it in a different directory.

6. Configure the release and compile everything:

```
shell> ./configure --prefix=/usr/local/mysql
shell> make
```

When you run **configure**, you might want to specify some options. Run **./configure --help** for a list of options. Section 2.8.2, “Typical **configure** Options”, discusses some of the more useful options.

If **configure** fails and you are going to send mail to a MySQL mailing list to ask for assistance, please include any lines from `config.log` that you think can help solve the problem. Also include the last couple of lines of output from **configure**. Post the bug report using the **mysqlbug** script. See Section 1.4.1.3, “How to Report Bugs or Problems”.

If the compile fails, see Section 2.8.4, “Dealing with Problems Compiling MySQL” for help.

7. Install the distribution:

```
shell> make install
```

If you want to set up an option file, use one of those present in the `support-files` directory as a template. For example:

```
shell> cp support-files/my-medium.cnf /etc/my.cnf
```

You might need to run these commands as `root`.

If you want to configure support for `InnoDB` tables, you should edit the `/etc/my.cnf` file, remove the # character before the option lines that start with `innodb_...`, and modify the option values to be what you want. See Section 4.3.2, “Using Option Files” and Section 15.4, “`InnoDB` Configuration”.

8. Change location into the installation directory:

```
shell> cd /usr/local/mysql
```

9. If you haven't installed MySQL before, you must create the MySQL grant tables:

```
shell> bin/mysql_install_db --user=mysql
```

If you run the command as `root`, you should use the `--user` option as shown. The value of the option should be the name of the login account that you created in the first step to use for running the server. If you run the command while logged in as that user, you can omit the `--user` option.

Note that for MySQL versions older than 3.22.10, **mysql_install_db** left the server running after creating the grant tables. This is no longer true; you need to restart the server after performing the remaining steps in this procedure.

10. Change the ownership of program binaries to `root` and ownership of the data directory to the user

that you run **mysqld** as. Assuming that you are located in the installation directory (`/usr/local/mysql`), the commands look like this:

```
shell> chown -R root  .
shell> chown -R mysql var
shell> chgrp -R mysql .
```

The first command changes the owner attribute of the files to the `root` user. The second changes the owner attribute of the data directory to the `mysql` user. The third changes the group attribute to the `mysql` group.

11. If you would like MySQL to start automatically when you boot your machine, you can copy `support-files/mysql.server` to the location where your system has its startup files. More information can be found in the `support-files/mysql.server` script itself and in Section 2.9.2.2, “Starting and Stopping MySQL Automatically”.

12. You can set up new accounts using the **bin/mysql_setpermission** script if you install the `DBI` and `DBD::mysql` Perl modules. For instructions, see Section 2.13, “Perl Installation Notes”.

After everything has been installed, you should initialize and test your distribution using this command:

```
shell> /usr/local/mysql/bin/mysqld_safe --user=mysql &
```

For versions of MySQL older than 4.0, substitute **safe_mysqld** for **mysqld_safe** in the command.

If that command fails immediately and prints `mysqld ended`, you can find some information in the *`host_name`*`.err` file in the data directory.

More information about **mysqld_safe** is given in Section 5.1.3, “The **mysqld_safe** Server Startup Script”.

Note: The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.9, “Post-Installation Setup and Testing”.

2.8.2. Typical configure Options

The **configure** script gives you a great deal of control over how you configure a MySQL source distribution. Typically you do this using options on the **configure** command line. You can also affect **configure** using certain environment variables. See Appendix F, *Environment Variables*. For a list of options supported by **configure**, run this command:

```
shell> ./configure --help
```

Some of the more commonly used **configure** options are described here:

- To compile just the MySQL client libraries and client programs and not the server, use the `--without-server` option:

  ```
  shell> ./configure --without-server
  ```

  If you don't have a C++ compiler, **mysql** cannot be compiled (it is the one client program that requires C++). In this case, you can remove the code in **configure** that tests for the C++ compiler and

then run **./configure** with the `--without-server` option. The compile step should still try to build **mysql**, but you can ignore any warnings about `mysql.cc`. (If **make** stops, try **make -k** to tell it to continue with the rest of the build even if errors occur.)

- If you want to build the embedded MySQL library (`libmysqld.a`) you should use the `--with-embedded-server` option.

- If you don't want your log files and database directories located under `/usr/local/var`, use a **configure** command something like one of these:

```
shell> ./configure --prefix=/usr/local/mysql
shell> ./configure --prefix=/usr/local \
           --localstatedir=/usr/local/mysql/data
```

  The first command changes the installation prefix so that everything is installed under `/usr/local/mysql` rather than the default of `/usr/local`. The second command preserves the default installation prefix, but overrides the default location for database directories (normally `/usr/local/var`) and changes it to `/usr/local/mysql/data`. After you have compiled MySQL, you can change these options with option files. See Section 4.3.2, “Using Option Files”.

- If you are using Unix and you want the MySQL socket located somewhere other than the default location (normally in the directory `/tmp` or `/var/run`), use a **configure** command like this:

```
shell> ./configure \
           --with-unix-socket-path=/usr/local/mysql/tmp/mysql.sock
```

  The socket filename must be an absolute pathname. You can also change the location of `mysql.sock` later by using a MySQL option file. See Section A.4.5, “How to Protect or Change the MySQL Socket File `/tmp/mysql.sock`”.

- If you want to compile statically linked programs (for example, to make a binary distribution, to get more speed, or to work around problems with some Red Hat Linux distributions), run **configure** like this:

```
shell> ./configure --with-client-ldflags=-all-static \
           --with-mysqld-ldflags=-all-static
```

- If you are using **gcc** and don't have `libg++` or `libstdc++` installed, you can tell **configure** to use **gcc** as your C++ compiler:

```
shell> CC=gcc CXX=gcc ./configure
```

  When you use **gcc** as your C++ compiler, it does not attempt to link in `libg++` or `libstdc++`. This may be a good idea to do even if you have these libraries installed, because some versions of them have caused strange problems for MySQL users in the past.

  The following list indicates some compilers and environment variable settings that are commonly used with each one.

  - **gcc** 2.7.2:

```
CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors"
```

  - **egcs** 1.0.3a:

```
CC=gcc CXX=gcc CXXFLAGS="-O3 -felide-constructors \
-fno-exceptions -fno-rtti"
```

- **gcc** 2.95.2:

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro \
-felide-constructors -fno-exceptions -fno-rtti"
```

- `pgcc` 2.90.29 or newer:

```
CFLAGS="-O3 -mpentiumpro -mstack-align-double" CXX=gcc \
CXXFLAGS="-O3 -mpentiumpro -mstack-align-double \
-felide-constructors -fno-exceptions -fno-rtti"
```

In most cases, you can get a reasonably optimized MySQL binary by using the options from the preceding list and adding the following options to the **configure** line:

```
--prefix=/usr/local/mysql --enable-assembler \
--with-mysqld-ldflags=-all-static
```

The full **configure** line would, in other words, be something like the following for all recent **gcc** versions:

```
CFLAGS="-O3 -mpentiumpro" CXX=gcc CXXFLAGS="-O3 -mpentiumpro \
-felide-constructors -fno-exceptions -fno-rtti" ./configure \
--prefix=/usr/local/mysql --enable-assembler \
--with-mysqld-ldflags=-all-static
```

The binaries we provide on the MySQL Web site at http://www.mysql.com/ are all compiled with full optimization and should be perfect for most users. See Section 2.1.2.5, “MySQL Binaries Compiled by MySQL AB”. There are some configuration settings you can tweak to make an even faster binary, but these are only for advanced users. See Section 7.5.4, “How Compiling and Linking Affects the Speed of MySQL”.

If the build fails and produces errors about your compiler or linker not being able to create the shared library `libmysqlclient.so.#` (where '#' is a version number), you can work around this problem by giving the `--disable-shared` option to **configure**. In this case, **configure** does not build a shared `libmysqlclient.so.#` library.

- By default, MySQL uses the `latin1` (ISO-8859-1) character set. To change the default set, use the `--with-charset` option:

  ```
  shell> ./configure --with-charset=CHARSET
  ```

  `CHARSET` may be one of `big5`, `cp1251`, `cp1257`, `czech`, `danish`, `dec8`, `dos`, `euc_kr`, `gb2312`, `gbk`, `german1`, `hebrew`, `hp8`, `hungarian`, `koi8_ru`, `koi8_ukr`, `latin1`, `latin2`, `sjis`, `swe7`, `tis620`, `ujis`, `usa7`, or `win1251ukr`. See Section 5.8.1, “The Character Set Used for Data and Sorting”.

  As of MySQL 4.1.1, the default collation may also be specified. MySQL uses the `latin1_swedish_ci` collation. To change this, use the `--with-collation` option:

```
shell> ./configure --with-collation=COLLATION
```

To change both the character set and the collation, use both the `--with-charset` and `--with-collation` options. The collation must be a legal collation for the character set. (Use the `SHOW COLLATION` statement to determine which collations are available for each character set.)

If you want to convert characters between the server and the client, you should take a look at the `SET CHARACTER SET` statement. See Section 13.5.3, "`SET` Syntax".

Warning: If you change character sets after having created any tables, you have to run **myisamchk -r -q --set-character-set=*`charset`*** on every table. Your indexes may be sorted incorrectly otherwise. (This can happen if you install MySQL, create some tables, then reconfigure MySQL to use a different character set and reinstall it.)

With the **configure** option `--with-extra-charsets=LIST`, you can define which additional character sets should be compiled into the server. `LIST` is either a list of character set names separated by spaces, `complex` to include all character sets that can't be dynamically loaded, or `all` to include all character sets into the binaries.

- To configure MySQL with debugging code, use the `--with-debug` option:

  ```
  shell> ./configure --with-debug
  ```

  This causes a safe memory allocator to be included that can find some errors and that provides output about what is happening. See Section E.1, "Debugging a MySQL Server".

- If your client programs are using threads, you also must compile a thread-safe version of the MySQL client library with the `--enable-thread-safe-client` configure option. This creates a `libmysqlclient_r` library with which you should link your threaded applications. See Section 22.2.15, "How to Make a Threaded Client".

- Options that pertain to particular systems can be found in the system-specific section of this manual. See Section 2.12, "Operating System-Specific Notes".

2.8.3. Installing from the Development Source Tree

Caution: You should read this section only if you are interested in helping us test our new code. If you just want to get MySQL up and running on your system, you should use a standard release distribution (either a binary or source distribution).

To obtain our most recent development source tree, use these instructions:

1. Download BitKeeper from http://www.bitmover.com/cgi-bin/download.cgi. You need Bitkeeper 3.0 or newer to access our repository.

2. Follow the instructions to install it.

3. After BitKeeper has been installed, first go to the directory you want to work from, and then use one of the following commands to clone the MySQL version branch of your choice:

   To clone the old 3.23 branch, use this command:

   ```
   shell> bk clone bk://mysql.bkbits.net/mysql-3.23 mysql-3.23
   ```

To clone the 4.0 stable (production) branch, use this command:

```
shell> bk clone bk://mysql.bkbits.net/mysql-4.0 mysql-4.0
```

To clone the 4.1 stable (production) branch, use this command:

```
shell> bk clone bk://mysql.bkbits.net/mysql-4.1 mysql-4.1
```

To clone the 5.0 development branch, use this command:

```
shell> bk clone bk://mysql.bkbits.net/mysql-5.0 mysql-5.0
```

In the preceding examples, the source tree is set up in the `mysql-3.23/`, `mysql-4.0/`, `mysql-4.1/`, or `mysql-5.0/` subdirectory of your current directory.

If you are behind a firewall and can only initiate HTTP connections, you can also use BitKeeper via HTTP.

If you are required to use a proxy server, set the environment variable `http_proxy` to point to your proxy:

```
shell> export http_proxy="http://your.proxy.server:8080/"
```

Replace the `bk://` with `http://` when doing a clone. Example:

```
shell> bk clone http://mysql.bkbits.net/mysql-4.1 mysql-4.1
```

The initial download of the source tree may take a while, depending on the speed of your connection. Please be patient.

4. You need GNU **make**, **autoconf** 2.58 (or newer), **automake** 1.8, **libtool** 1.5, and **m4** to run the next set of commands. Even though many operating systems come with their own implementation of **make**, chances are high that the compilation fails with strange error messages. Therefore, it is highly recommended that you use GNU **make** (sometimes named **gmake**) instead.

   Fortunately, a large number of operating systems ship with the GNU toolchain preinstalled or supply installable packages of these. In any case, they can also be downloaded from the following locations:

   - http://www.gnu.org/software/autoconf/
   - http://www.gnu.org/software/automake/
   - http://www.gnu.org/software/libtool/
   - http://www.gnu.org/software/m4/
   - http://www.gnu.org/software/make/

   If you are trying to configure MySQL 4.1 or later, you also need GNU **bison** 1.75 or later. Older versions of **bison** may report this error:

```
sql_yacc.yy:#####: fatal error: maximum table size (32767) exceeded
```

Note: The maximum table size is not actually exceeded; the error is caused by bugs in older versions of **bison**.

Versions of MySQL before version 4.1 may also compile with other **yacc** implementations (for example, BSD **yacc** 91.7.30). For later versions, GNU **bison** is required.

The following example shows the typical commands required to configure a source tree. The first `cd` command changes location into the top-level directory of the tree; replace `mysql-4.0` with the appropriate directory name.

```
shell> cd mysql-4.0
shell> bk -r edit
shell> aclocal; autoheader; autoconf; automake
shell> (cd innobase; aclocal; autoheader; autoconf; automake)
shell> (cd bdb/dist; sh s_all)
shell> ./configure  # Add your favorite options here
make
```

The command lines that change directory into the `innobase` and `bdb/dist` directories are used to configure the `InnoDB` and Berkeley DB (`BDB`) storage engines. You can omit these command lines if you to not require `InnoDB` or `BDB` support.

If you get some strange errors during this stage, verify that you really have **libtool** installed.

A collection of our standard configuration scripts is located in the `BUILD/` subdirectory. You may find it more convenient to use the `BUILD/compile-pentium-debug` script than the preceding set of shell commands. To compile on a different architecture, modify the script by removing flags that are Pentium-specific.

5. When the build is done, run **make install**. Be careful with this on a production machine; the command may overwrite your live release installation. If you have another installation of MySQL, we recommend that you run **./configure** with different values for the `--prefix`, `--with-tcp-port`, and `--unix-socket-path` options than those used for your production server.

6. Play hard with your new installation and try to make the new features crash. Start by running **make test**. See Section 25.1.2, “MySQL Test Suite”.

7. If you have gotten to the **make** stage and the distribution does not compile, please report it in our bugs database at http://bugs.mysql.com/. If you have installed the latest versions of the required GNU tools, and they crash trying to process our configuration files, please report that also. However, if you execute `aclocal` and get a `command not found` error or a similar problem, do not report it. Instead, make sure that all the necessary tools are installed and that your `PATH` variable is set correctly so that your shell can find them.

8. After the initial `bk clone` operation to obtain the source tree, you should run `bk pull` periodically to get updates.

9. You can examine the change history for the tree with all the diffs by using `bk revtool`. If you see some funny diffs or code that you have a question about, do not hesitate to send email to the MySQL `internals` mailing list. See Section 1.4.1.1, “The MySQL Mailing Lists”. Also, if you think you have a better idea on how to do something, send an email message to the same address with a patch. `bk diffs` produces a patch for you after you have made changes to the source. If you do not have the time to code your idea, just send a description.

10. BitKeeper has a nice help utility that you can access via `bk helptool`.

11. Please note that any commits (made via `bk ci` or `bk citool`) triggers the posting of a message with the changeset to our internals mailing list, as well as the usual openlogging.org submission with just the changeset comments. Generally, you wouldn't need to use commit (since the public tree does not allow `bk push`), but rather use the `bk diffs` method described previously.

You can also browse changesets, comments, and source code online. For example, to browse this information for MySQL 4.1, go to http://mysql.bkbits.net:8080/mysql-4.1.

The manual is in a separate tree that can be cloned with:

```
shell> bk clone bk://mysql.bkbits.net/mysqldoc mysqldoc
```

There are also public BitKeeper trees for MySQL Control Center and MyODBC. They can be cloned respectively as follows.

To clone MySQL Control center, use this command:

```
shell> bk clone http://mysql.bkbits.net/mysqlcc mysqlcc
```

To clone MyODBC, use this command:

```
shell> bk clone http://mysql.bkbits.net/myodbc3 myodbc3
```

To clone Connector/NET, use this command:

```
shell> bk clone http://mysql.bkbits.net/connector-net connector-net
```

2.8.4. Dealing with Problems Compiling MySQL

All MySQL programs compile cleanly for us with no warnings on Solaris or Linux using **gcc**. On other systems, warnings may occur due to differences in system include files. See Section 2.8.5, “MIT-pthreads Notes” for warnings that may occur when using MIT-pthreads. For other problems, check the following list.

The solution to many problems involves reconfiguring. If you do need to reconfigure, take note of the following:

- If **configure** is run after it has previously been run, it may use information that was gathered during its previous invocation. This information is stored in `config.cache`. When **configure** starts up, it looks for that file and reads its contents if it exists, on the assumption that the information is still correct. That assumption is invalid when you reconfigure.
- Each time you run **configure**, you must run **make** again to recompile. However, you may want to remove old object files from previous builds first because they were compiled using different configuration options.

To prevent old configuration information or object files from being used, run these commands before rerunning **configure**:

```
shell> rm config.cache
shell> make clean
```

Alternatively, you can run **make distclean**.

The following list describes some of the problems when compiling MySQL that have been found to occur most often:

- If you get errors such as the ones shown here when compiling `sql_yacc.cc`, you probably have run out of memory or swap space:

  ```
  Internal compiler error: program cc1plus got fatal signal 11
  Out of virtual memory
  Virtual memory exhausted
  ```

  The problem is that **gcc** requires a huge amount of memory to compile `sql_yacc.cc` with inline functions. Try running **configure** with the `--with-low-memory` option:

  ```
  shell> ./configure --with-low-memory
  ```

  This option causes `-fno-inline` to be added to the compile line if you are using **gcc** and `-O0` if you are using something else. You should try the `--with-low-memory` option even if you have so much memory and swap space that you think you can't possibly have run out. This problem has been observed to occur even on systems with generous hardware configurations and the `--with-low-memory` option usually fixes it.

- By default, **configure** picks **c++** as the compiler name and GNU **c++** links with `-lg++`. If you are using **gcc**, that behavior can cause problems during configuration such as this:

  ```
  configure: error: installation or configuration problem:
  C++ compiler cannot create executables.
  ```

  You might also observe problems during compilation related to **g++**, `libg++`, or `libstdc++`.

  One cause of these problems is that you may not have **g++**, or you may have **g++** but not `libg++`, or `libstdc++`. Take a look at the `config.log` file. It should contain the exact reason why your C++ compiler didn't work. To work around these problems, you can use **gcc** as your C++ compiler. Try setting the environment variable `CXX` to `"gcc -O3"`. For example:

  ```
  shell> CXX="gcc -O3" ./configure
  ```

  This works because **gcc** compiles C++ sources as well as **g++** does, but does not link in `libg++` or `libstdc++` by default.

  Another way to fix these problems is to install **g++**, `libg++`, and `libstdc++`. We would, however, like to recommend that you not use `libg++` or `libstdc++` with MySQL because this only increases the binary size of **mysqld** without giving you any benefits. Some versions of these libraries have also caused strange problems for MySQL users in the past.

  Using **gcc** as the C++ compiler is also required if you want to compile MySQL with RAID functionality (see Section 13.2.6, “`CREATE TABLE` Syntax” for more info on RAID table type) and you are using GNU **gcc** version 3 and above. If you get errors like those following during the linking stage when you configure MySQL to compile with the option `--with-raid`, try to use **gcc** as your C++ compiler by defining the `CXX` environment variable:

  ```
  gcc -O3 -DDBUG_OFF -rdynamic -o isamchk isamchk.o sort.o  libnisam.a
  ../mysys/libmysys.a ../dbug/libdbug.a ../strings/libmystrings.a
   -lpthread -lz -lcrypt -lnsl -lm -lpthread
  ```

```
../mysys/libmysys.a(raid.o)(.text+0x79): In function
`my_raid_create':: undefined reference to `operator new(unsigned)'
../mysys/libmysys.a(raid.o)(.text+0xdd): In function
`my_raid_create':: undefined reference to `operator delete(void*)'
../mysys/libmysys.a(raid.o)(.text+0x129): In function
`my_raid_open':: undefined reference to `operator new(unsigned)'
../mysys/libmysys.a(raid.o)(.text+0x189): In function
`my_raid_open':: undefined reference to `operator delete(void*)'
../mysys/libmysys.a(raid.o)(.text+0x64b): In function
`my_raid_close':: undefined reference to `operator delete(void*)'
collect2: ld returned 1 exit status
```

- If your compile fails with errors such as any of the following, you must upgrade your version of **make** to GNU **make**:

  ```
  making all in mit-pthreads
  make: Fatal error in reader: Makefile, line 18:
  Badly formed macro assignment
  ```

  Or:

  ```
  make: file `Makefile' line 18: Must be a separator (:
  ```

  Or:

  ```
  pthread.h: No such file or directory
  ```

  Solaris and FreeBSD are known to have troublesome **make** programs.

  GNU **make** Version 3.75 is known to work.

- If you want to define flags to be used by your C or C++ compilers, do so by adding the flags to the `CFLAGS` and `CXXFLAGS` environment variables. You can also specify the compiler names this way using `CC` and `CXX`. For example:

  ```
  shell> CC=gcc
  shell> CFLAGS=-O3
  shell> CXX=gcc
  shell> CXXFLAGS=-O3
  shell> export CC CFLAGS CXX CXXFLAGS
  ```

  See Section 2.1.2.5, “MySQL Binaries Compiled by MySQL AB”, for a list of flag definitions that have been found to be useful on various systems.

- If you get an error message like this, you need to upgrade your **gcc** compiler:

  ```
  client/libmysql.c:273: parse error before `__attribute__'
  ```

  gcc 2.8.1 is known to work, but we recommend using **gcc** 2.95.2 or **egcs** 1.0.3a instead.

- If you get errors such as those shown here when compiling **mysqld**, **configure** didn't correctly detect the type of the last argument to `accept()`, `getsockname()`, or `getpeername()`:

  ```
  cxx: Error: mysqld.cc, line 645: In this statement, the referenced
       type of the pointer value ''length'' is ''unsigned long'',
       which is not compatible with ''int''.
  ```

```
new_sock = accept(sock, (struct sockaddr *)&cAddr, &length);
```

To fix this, edit the `config.h` file (which is generated by **configure**). Look for these lines:

```
/* Define as the base type of the last arg to accept */
#define SOCKET_SIZE_TYPE XXX
```

Change `XXX` to `size_t` or `int`, depending on your operating system. (Note that you have to do this each time you run **configure** because **configure** regenerates `config.h`.)

- The `sql_yacc.cc` file is generated from `sql_yacc.yy`. Normally the build process doesn't need to create `sql_yacc.cc`, because MySQL comes with an pre-generated copy. However, if you do need to re-create it, you might encounter this error:

  ```
  "sql_yacc.yy", line xxx fatal: default action causes potential...
  ```

  This is a sign that your version of **yacc** is deficient. You probably need to install **bison** (the GNU version of **yacc**) and use that instead.

- On Debian Linux 3.0, you need to install `gawk` instead of the default `mawk` if you want to compile MySQL 4.1 or higher with Berkeley DB support.

- If you need to debug **mysqld** or a MySQL client, run **configure** with the `--with-debug` option, then recompile and link your clients with the new client library. See Section E.2, “Debugging a MySQL Client”.

- If you get a compilation error on Linux (for example, SuSE Linux 8.1 or Red Hat Linux 7.3) similar to the following one:

  ```
  libmysql.c:1329: warning: passing arg 5 of `gethostbyname_r' from
  incompatible pointer type
  libmysql.c:1329: too few arguments to function `gethostbyname_r'
  libmysql.c:1329: warning: assignment makes pointer from integer
  without a cast
  make[2]: *** [libmysql.lo] Error 1
  ```

  By default, the **configure** script attempts to determine the correct number of arguments by using **g++** the GNU C++ compiler. This test yields wrong results if **g++** is not installed. There are two ways to work around this problem:

  - Make sure that the GNU C++ **g++** is installed. On some Linux distributions, the required package is called `gpp`; on others, it is named **gcc-c++**.

  - Use **gcc** as your C++ compiler by setting the `CXX` environment variable to **gcc**:

    ```
    export CXX="gcc"
    ```

  Please note that you need to run **configure** again afterward.

2.8.5. MIT-pthreads Notes

This section describes some of the issues involved in using MIT-pthreads.

On Linux, you should *not* use MIT-pthreads. Use the installed LinuxThreads implementation instead. See Section 2.12.1, “Linux Notes”.

If your system does not provide native thread support, you need to build MySQL using the MIT-pthreads package. This includes older FreeBSD systems, SunOS 4.x, Solaris 2.4 and earlier, and some others. See Section 2.1.1, “Operating Systems Supported by MySQL”.

Beginning with MySQL 4.0.2, MIT-pthreads is no longer part of the source distribution. If you require this package, you need to download it separately from http://www.mysql.com/Downloads/Contrib/pthreads-1_60_beta6-mysql.tar.gz

After downloading, extract this source archive into the top level of the MySQL source directory. It creates a new subdirectory named `mit-pthreads`.

- On most systems, you can force MIT-pthreads to be used by running **configure** with the `--with-mit-threads` option:

  ```
  shell> ./configure --with-mit-threads
  ```

  Building in a non-source directory is not supported when using MIT-pthreads because we want to minimize our changes to this code.

- The checks that determine whether to use MIT-pthreads occur only during the part of the configuration process that deals with the server code. If you have configured the distribution using `--without-server` to build only the client code, clients do not know whether MIT-pthreads is being used and use Unix socket connections by default. Because Unix socket files do not work under MIT-pthreads on some platforms, this means you need to use `-h` or `--host` when you run client programs.

- When MySQL is compiled using MIT-pthreads, system locking is disabled by default for performance reasons. You can tell the server to use system locking with the `--external-locking` option. This is needed only if you want to be able to run two MySQL servers against the same data files, which is not recommended.

- Sometimes the pthread `bind()` command fails to bind to a socket without any error message (at least on Solaris). The result is that all connections to the server fail. For example:

  ```
  shell> mysqladmin version
  mysqladmin: connect to server at '' failed;
  error: 'Can't connect to mysql server on localhost (146)'
  ```

  The solution to this is to kill the **mysqld** server and restart it. This has only happened to us when we have forced down the server and done a restart immediately.

- With MIT-pthreads, the `sleep()` system call isn't interruptible with `SIGINT` (break). This is only noticeable when you run **mysqladmin --sleep**. You must wait for the `sleep()` call to terminate before the interrupt is served and the process stops.

- When linking, you may receive warning messages like these (at least on Solaris); they can be ignored:

  ```
  ld: warning: symbol `_iob' has differing sizes:
      (file /my/local/pthreads/lib/libpthread.a(findfp.o) value=0x4;
  file /usr/lib/libc.so value=0x140);
      /my/local/pthreads/lib/libpthread.a(findfp.o) definition taken
  ld: warning: symbol `__iob' has differing sizes:
      (file /my/local/pthreads/lib/libpthread.a(findfp.o) value=0x4;
  ```

```
file /usr/lib/libc.so value=0x140);
    /my/local/pthreads/lib/libpthread.a(findfp.o) definition taken
```

- Some other warnings also can be ignored:

```
implicit declaration of function `int strtoll(...)'
implicit declaration of function `int strtoul(...)'
```

- We haven't gotten `readline` to work with MIT-pthreads. (This isn't needed, but may be interesting for someone.)

2.8.6. Installing MySQL from Source on Windows

These instructions describe how to build MySQL binaries from source for versions 4.1 and above on Windows. Instructions are provided for building binaries from a standard source distribution or from the BitKeeper tree that contains the latest development source.

Note: The instructions in this document are strictly for users who want to test MySQL on Windows from the latest source distribution or from the BitKeeper tree. For production use, MySQL AB does not advise using a MySQL server built by yourself from source. Normally, it is best to use precompiled binary distributions of MySQL that are built specifically for optimal performance on Windows by MySQL AB. Instructions for installing a binary distributions are available at Section 2.3, “Installing MySQL on Windows”.

To build MySQL on Windows from source, you need the following compiler and resources available on your Windows system:

- VC++ 6.0 compiler (updated with 4 or 5 SP and pre-processor package). The pre-processor package is necessary for the macro assembler. More details can be found at http://msdn.microsoft.com/vstudio/downloads/updates/sp/vs6/sp5/faq.aspx.
- Approximately 45MB disk space.
- 64MB RAM.

You'll also need a MySQL source distribution for Windows. There are two ways you can get a source distribution for MySQL version 4.1 and above:

1. Obtain a source distribution packaged by MySQL AB for the particular version of MySQL in which you are interested. Prepackaged source distributions are available for released versions of MySQL and can be obtained from http://dev.mysql.com/downloads/.
2. You can package a source distribution yourself from the latest BitKeeper developer source tree. If you plan to do this, you must create the package on a Unix system and then transfer it to your Windows system. (The reason for this is that some of the configuration and build steps require tools that work only on Unix.) The BitKeeper approach thus requires:
   - A system running Unix, or a Unix-like system such as Linux.
   - BitKeeper 3.0 installed on that system. You can obtain BitKeeper from http://www.bitkeeper.com/.

If you are using a Windows source distribution, you can go directly to Section 2.8.6.1, “Building MySQL Using VC++”. To build from the BitKeeper tree, proceed to Section 2.8.6.2, “Creating a Windows Source Package from the Latest Development Source”.

If you find something not working as expected, or you have suggestions about ways to improve the current build process on Windows, please send a message to the `win32` mailing list. See Section 1.4.1.1, “The MySQL Mailing Lists”.

2.8.6.1. Building MySQL Using VC++

Note: VC++ workspace files for MySQL 4.1 and above are compatible with Microsoft Visual Studio 6.0 and above (7.0/.NET) editions and tested by MySQL AB staff before each release.

Follow this procedure to build MySQL:

1. Create a work directory (for example, `C:\workdir`).
2. Unpack the source distribution in the aforementioned directory using `WinZip` or other Windows tool that can read `.zip` files.
3. Start the VC++ 6.0 compiler.
4. In the `File` menu, select `Open Workspace`.
5. Open the `mysql.dsw` workspace you find in the work directory.
6. From the `Build` menu, select the `Set Active Configuration` menu.
7. Click over the screen selecting `mysqld - Win32 Debug` and click OK.
8. Press `F7` to begin the build of the debug server, libraries, and some client applications.
9. Compile the release versions that you want in the same way.
10. Debug versions of the programs and libraries are placed in the `client_debug` and `lib_debug` directories. Release versions of the programs and libraries are placed in the `client_release` and `lib_release` directories. Note that if you want to build both debug and release versions, you can select the `Build All` option from the `Build` menu.
11. Test the server. The server built using the preceding instructions expects that the MySQL base directory and data directory are `C:\mysql` and `C:\mysql\data` by default. If you want to test your server using the source tree root directory and its data directory as the base directory and data directory, you need to tell the server their pathnames. You can either do this on the command line with the `--basedir` and `--datadir` options, or place appropriate options in an option file (the `my.ini` file in your Windows directory or `C:\my.cnf`). If you have an existing data directory elsewhere that you want to use, you can specify its pathname instead.
12. Start your server from the `client_release` or `client_debug` directory, depending on which server you want to use. The general server startup instructions are at Section 2.3, “Installing MySQL on Windows”. You'll need to adapt the instructions appropriately if you want to use a different base directory or data directory.
13. When the server is running in standalone fashion or as a service based on your configuration, try to connect to it from the **mysql** interactive command-line utility that exists in your `client_release` or `client_debug` directory.

When you are satisfied that the programs you have built are working correctly, stop the server. Then in-

stall MySQL as follows:

1. Create the directories where you want to install MySQL. For example, to install into `C:\mysql`, use these commands:

   ```
   C:\> mkdir C:\mysql
   C:\> mkdir C:\mysql\bin
   C:\> mkdir C:\mysql\data
   C:\> mkdir C:\mysql\share
   C:\> mkdir C:\mysql\scripts
   ```

   If you want to compile other clients and link them to MySQL, you should also create several additional directories:

   ```
   C:\> mkdir C:\mysql\include
   C:\> mkdir C:\mysql\lib
   C:\> mkdir C:\mysql\lib\debug
   C:\> mkdir C:\mysql\lib\opt
   ```

   If you want to benchmark MySQL, create this directory:

   ```
   C:\> mkdir C:\mysql\sql-bench
   ```

   Benchmarking requires Perl support.

2. From the `workdir` directory, copy into the `C:\mysql` directory the following directories:

   ```
   C:\> cd \workdir
   C:\workdir> copy client_release\*.exe C:\mysql\bin
   C:\workdir> copy client_debug\mysqld.exe C:\mysql\bin\mysqld-debug.exe
   C:\workdir> xcopy scripts\*.* C:\mysql\scripts /E
   C:\workdir> xcopy share\*.* C:\mysql\share /E
   ```

   If you want to compile other clients and link them to MySQL, you should also copy several libraries and header files:

   ```
   C:\workdir> copy lib_debug\mysqlclient.lib C:\mysql\lib\debug
   C:\workdir> copy lib_debug\libmysql.* C:\mysql\lib\debug
   C:\workdir> copy lib_debug\zlib.* C:\mysql\lib\debug
   C:\workdir> copy lib_release\mysqlclient.lib C:\mysql\lib\opt
   C:\workdir> copy lib_release\libmysql.* C:\mysql\lib\opt
   C:\workdir> copy lib_release\zlib.* C:\mysql\lib\opt
   C:\workdir> copy include\*.h C:\mysql\include
   C:\workdir> copy libmysql\libmysql.def C:\mysql\include
   ```

   If you want to benchmark MySQL, you should also do this:

   ```
   C:\workdir> xcopy sql-bench\*.* C:\mysql\bench /E
   ```

Set up and start the server in the same way as for the binary Windows distribution. See Section 2.3, “Installing MySQL on Windows”.

2.8.6.2. Creating a Windows Source Package from the Latest Devel-

opment Source

To create a Windows source package from the current BitKeeper source tree, use the following instructions. Please note that this procedure must be performed on a system running a Unix or Unix-like operating system. For example, the procedure is known to work well on Linux.

1. Clone the BitKeeper source tree for MySQL (version 4.1 or above, as desired). For more information on how to clone the source tree, see the instructions at Section 2.8.3, “Installing from the Development Source Tree”.

2. Configure and build the distribution so that you have a server binary to work with. One way to do this is to run the following command in the top-level directory of your source tree:

   ```
   shell> ./BUILD/compile-pentium-max
   ```

3. After making sure that the build process completed successfully, run the following utility script from top-level directory of your source tree:

   ```
   shell> ./scripts/make_win_src_distribution
   ```

   This script creates a Windows source package to be used on your Windows system. You can supply different options to the script based on your needs. It accepts the following options:

   - `--help`

     Display a help message.

   - `--debug`

     Print information about script operations, do not create package.

   - `--tmp`

     Specify the temporary location.

   - `--suffix`

     Suffix name for the package.

   - `--dirname`

     Directory name to copy files (intermediate).

   - `--silent`

     Do not print verbose list of files processed.

   - `--tar`

     Create `tar.gz` package instead of `.zip` package.

   By default, **make_win_src_distribution** creates a Zip-format archive with the name `mysql-`*`VERSION`*`-win-src.zip`, where *`VERSION`* represents the version of your MySQL source tree.

4. Copy or upload to your Windows machine the Windows source package that you have just created. To compile it, use the instructions in Section 2.8.6.1, “Building MySQL Using VC++”.

2.8.7. Compiling MySQL Clients on Windows

In your source files, you should include `my_global.h` before `mysql.h`:

```
#include <my_global.h>
#include <mysql.h>
```

`my_global.h` includes any other files needed for Windows compatibility (such as `windows.h`) if you compile your program on Windows.

You can either link your code with the dynamic `libmysql.lib` library, which is just a wrapper to load in `libmysql.dll` on demand, or link with the static `mysqlclient.lib` library.

The MySQL client libraries are compiled as threaded libraries, so you should also compile your code to be multi-threaded.

2.9. Post-Installation Setup and Testing

After installing MySQL, there are some issues you should address. For example, on Unix, you should initialize the data directory and create the MySQL grant tables. On all platforms, an important security concern is that the initial accounts in the grant tables have no passwords. You should assign passwords to prevent unauthorized access to the MySQL server. For MySQL 4.1.3 and up, you can create time zone tables to enable recognition of named time zones. (Currently, these tables can be populated only on Unix. This problem will be addressed soon for Windows.)

The following sections include post-installation procedures that are specific to Windows systems and to Unix systems. Another section, Section 2.9.2.3, “Starting and Troubleshooting the MySQL Server”, applies to all platforms; it describes what to do if you have trouble getting the server to start. Section 2.9.3, “Securing the Initial MySQL Accounts” also applies to all platforms. You should follow its instructions to make sure that you have properly protected your MySQL accounts by assigning passwords to them.

When you are ready to create additional user accounts, you can find information on the MySQL access control system and account management in Section 5.5, “The MySQL Access Privilege System” and Section 5.6, “MySQL User Account Management”.

2.9.1. Windows Post-Installation Procedures

On Windows, the data directory and the grant tables do not have to be created. MySQL Windows distributions include the grant tables with a set of preinitialized accounts in the `mysql` database under the data directory. You do not run the **mysql_install_db** script that is used on Unix. However, if you did not install MySQL using the Windows Installation Wizard, you should assign passwords to the accounts. See Section 2.3.4.1, “Introduction”. The procedure for this is given in Section 2.9.3, “Securing the Initial MySQL Accounts”.

Before setting up passwords, you might want to try running some client programs to make sure that you can connect to the server and that it is operating properly. Make sure the server is running (see Section 2.3.10, “Starting the Server for the First Time”), then issue the following commands to verify that you can retrieve information from the server. The output should be similar to what is shown here:

```
C:\> C:\mysql\bin\mysqlshow
+-----------+
| Databases |
+-----------+
| mysql     |
| test      |
+-----------+
```

```
C:\> C:\mysql\bin\mysqlshow mysql
Database: mysql
+--------------+
|    Tables    |
+--------------+
| columns_priv |
| db           |
| func         |
| host         |
| tables_priv  |
| user         |
+--------------+

C:\> C:\mysql\bin\mysql -e "SELECT Host,Db,User FROM db" mysql
+------+-------+------+
| host | db    | user |
+------+-------+------+
| %    | test% |      |
+------+-------+------+
```

If you are running a version of Windows that supports services and you want the MySQL server to run automatically when Windows starts, see Section 2.3.12, “Starting MySQL as a Windows Service”.

2.9.2. Unix Post-Installation Procedures

After installing MySQL on Unix, you need to initialize the grant tables, start the server, and make sure that the server works okay. You may also wish to arrange for the server to be started and stopped automatically when your system starts and stops. You should also assign passwords to the accounts in the grant tables.

On Unix, the grant tables are set up by the **mysql_install_db** program. For some installation methods, this program is run for you automatically:

- If you install MySQL on Linux using RPM distributions, the server RPM runs **mysql_install_db**.
- If you install MySQL on Mac OS X using a PKG distribution, the installer runs **mysql_install_db**.

Otherwise, you'll need to run **mysql_install_db** yourself.

The following procedure describes how to initialize the grant tables (if that has not previously been done) and then start the server. It also suggests some commands that you can use to test whether the server is accessible and working properly. For information about starting and stopping the server automatically, see Section 2.9.2.2, “Starting and Stopping MySQL Automatically”.

After you complete the procedure and have the server running, you should assign passwords to the accounts created by **mysql_install_db**. Instructions for doing so are given in Section 2.9.3, “Securing the Initial MySQL Accounts”.

In the examples shown here, the server runs under the user ID of the `mysql` login account. This assumes that such an account exists. Either create the account if it does not exist, or substitute the name of a different existing login account that you plan to use for running the server.

1. Change location into the top-level directory of your MySQL installation, represented here by *BASEDIR*:

   ```
   shell> cd BASEDIR
   ```

BASEDIR is likely to be something like /usr/local/mysql or /usr/local. The following steps assume that you are located in this directory.

2. If necessary, run the **mysql_install_db** program to set up the initial MySQL grant tables containing the privileges that determine how users are allowed to connect to the server. You'll need to do this if you used a distribution type that doesn't run the program for you.

   Typically, **mysql_install_db** needs to be run only the first time you install MySQL, so you can skip this step if you are upgrading an existing installation, However, **mysql_install_db** does not overwrite any existing privilege tables, so it should be safe to run in any circumstances.

   To initialize the grant tables, use one of the following commands, depending on whether **mysql_install_db** is located in the bin or scripts directory:

   ```
   shell> bin/mysql_install_db --user=mysql
   shell> scripts/mysql_install_db --user=mysql
   ```

   The **mysql_install_db** script creates the data directory, the mysql database that holds all database privileges, and the test database that you can use to test MySQL. The script also creates privilege table entries for root accounts and anonymous-user accounts. The accounts have no passwords initially. A description of their initial privileges is given in Section 2.9.3, “Securing the Initial MySQL Accounts”. Briefly, these privileges allow the MySQL root user to do anything, and allow anybody to create or use databases with a name of test or starting with test_.

   It is important to make sure that the database directories and files are owned by the mysql login account so that the server has read and write access to them when you run it later. To ensure this, the --user option should be used as shown if you run **mysql_install_db** as root. Otherwise, you should execute the script while logged in as mysql, in which case you can omit the --user option from the command.

   mysql_install_db creates several tables in the mysql database: user, db, host, tables_priv, columns_priv, func, and possibly others depending on your version of MySQL.

   If you don't want to have the test database, you can remove it with **mysqladmin -u root drop test** after starting the server.

   If you have problems with **mysql_install_db**, see Section 2.9.2.1, “Problems Running **mysql_install_db**”.

   There are some alternatives to running the **mysql_install_db** script as it is provided in the MySQL distribution:

   - If you want the initial privileges to be different from the standard defaults, you can modify **mysql_install_db** before you run it. However, a preferable technique is to use GRANT and REVOKE to change the privileges after the grant tables have been set up. In other words, you can run **mysql_install_db**, and then use mysql -u root mysql to connect to the server as the MySQL root user so that you can issue the GRANT and REVOKE statements.

     If you want to install MySQL on a lot of machines with the same privileges, you can put the GRANT and REVOKE statements in a file and execute the file as a script using mysql after running **mysql_install_db**. For example:

     ```
     shell> bin/mysql_install_db --user=mysql
     shell> bin/mysql -u root < your_script_file
     ```

By doing this, you can avoid having to issue the statements manually on each machine.

- It is possible to re-create the grant tables completely after they have previously been created. You might want to do this if you're just learning how to use `GRANT` and `REVOKE` and have made so many modifications after running **mysql_install_db** that you want to wipe out the tables and start over.

  To re-create the grant tables, remove all the `.frm`, `.MYI`, and `.MYD` files in the directory containing the `mysql` database. (This is the directory named `mysql` under the data directory, which is listed as the `datadir` value when you run **mysqld --help**.) Then run the **mysql_install_db** script again.

  Note: For MySQL versions older than 3.22.10, you should not delete the `.frm` files. If you accidentally do this, you should copy them back into the `mysql` directory from your MySQL distribution before running **mysql_install_db**.

- You can start **mysqld** manually using the `--skip-grant-tables` option and add the privilege information yourself using **mysql**:

  ```
  shell> bin/mysqld_safe --user=mysql --skip-grant-tables &
  shell> bin/mysql mysql
  ```

  From **mysql**, manually execute the SQL commands contained in **mysql_install_db**. Make sure that you run **mysqladmin flush-privileges** or **mysqladmin reload** afterward to tell the server to reload the grant tables.

  Note that by not using **mysql_install_db**, you not only have to populate the grant tables manually, you also have to create them first.

3. Start the MySQL server:

   ```
   shell> bin/mysqld_safe --user=mysql &
   ```

   For versions of MySQL older than 4.0, substitute **bin/safe_mysqld** for **bin/mysqld_safe** in this command.

   It is important that the MySQL server be run using an unprivileged (non-`root`) login account. To ensure this, the `--user` option should be used as shown if you run `mysql_safe` as `root`. Otherwise, you should execute the script while logged in as `mysql`, in which case you can omit the `--user` option from the command.

   Further instructions for running MySQL as an unprivileged user are given in Section A.3.2, “How to Run MySQL as a Normal User”.

   If you neglected to create the grant tables before proceeding to this step, the following message appears in the error log file when you start the server:

   ```
   mysqld: Can't find file: 'host.frm'
   ```

   If you have other problems starting the server, see Section 2.9.2.3, “Starting and Troubleshooting the MySQL Server”.

4. Use **mysqladmin** to verify that the server is running. The following commands provide simple tests to check whether the server is up and responding to connections:

```
shell> bin/mysqladmin version
shell> bin/mysqladmin variables
```

The output from **mysqladmin version** varies slightly depending on your platform and version of MySQL, but should be similar to that shown here:

```
shell> bin/mysqladmin version
mysqladmin  Ver 8.40 Distrib 4.0.18, for linux on i586
Copyright (C) 2000 MySQL AB & MySQL Finland AB & TCX DataKonsult AB
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL license

Server version          4.0.18-log
Protocol version        10
Connection              Localhost via Unix socket
TCP port                3306
UNIX socket             /tmp/mysql.sock
Uptime:                 16 sec

Threads: 1  Questions: 9  Slow queries: 0
Opens: 7  Flush tables: 2  Open tables: 0
Queries per second avg: 0.000
Memory in use: 132K  Max memory used: 16773K
```

To see what else you can do with **mysqladmin**, invoke it with the `--help` option.

5. Verify that you can shut down the server:

```
shell> bin/mysqladmin -u root shutdown
```

6. Verify that you can restart the server. Do this by using **mysqld_safe** or by invoking **mysqld** directly. For example:

```
shell> bin/mysqld_safe --user=mysql --log &
```

If **mysqld_safe** fails, see Section 2.9.2.3, “Starting and Troubleshooting the MySQL Server”.

7. Run some simple tests to verify that you can retrieve information from the server. The output should be similar to what is shown here:

```
shell> bin/mysqlshow
+-----------+
| Databases |
+-----------+
| mysql     |
| test      |
+-----------+

shell> bin/mysqlshow mysql
Database: mysql
+--------------+
|    Tables    |
+--------------+
| columns_priv |
| db           |
| func         |
| host         |
| tables_priv  |
```

```
| user         |
+--------------+

shell> bin/mysql -e "SELECT Host,Db,User FROM db" mysql
+------+--------+------+
| host | db     | user |
+------+--------+------+
| %    | test   |      |
| %    | test_% |      |
+------+--------+------+
```

8. There is a benchmark suite in the `sql-bench` directory (under the MySQL installation directory) that you can use to compare how MySQL performs on different platforms. The benchmark suite is written in Perl. It uses the Perl DBI module to provide a database-independent interface to the various databases, and some other additional Perl modules are required to run the benchmark suite. You must have the following modules installed:

   ```
   DBI
   DBD::mysql
   Data::Dumper
   Data::ShowTable
   ```

   These modules can be obtained from CPAN (http://www.cpan.org/). See Section 2.13.1, “Installing Perl on Unix”.

   The `sql-bench/Results` directory contains the results from many runs against different databases and platforms. To run all tests, execute these commands:

   ```
   shell> cd sql-bench
   shell> perl run-all-tests
   ```

   If you don't have the `sql-bench` directory, you probably installed MySQL using RPM files other than the source RPM. (The source RPM includes the `sql-bench` benchmark directory.) In this case, you must first install the benchmark suite before you can use it. Beginning with MySQL 3.22, there are separate benchmark RPM files named `mysql-bench-`*`VERSION`*`-i386.rpm` that contain benchmark code and data.

   If you have a source distribution, there are also tests in its `tests` subdirectory that you can run. For example, to run `auto_increment.tst`, execute this command from the top-level directory of your source distribution:

   ```
   shell> mysql -vvf test < ./tests/auto_increment.tst
   ```

   The expected result of the test can be found in the `./tests/auto_increment.res` file.

9. At this point, you should have the server running. However, none of the initial MySQL accounts have a password, so you should assign passwords using the instructions in Section 2.9.3, “Securing the Initial MySQL Accounts”.

As of MySQL 4.1.3, the installation procedure creates time zone tables in the `mysql` database. However, you must populate the tables manually. Instructions to do this are given in Section 5.8.8, “MySQL Server Time Zone Support”.

2.9.2.1. Problems Running mysql_install_db

The purpose of the **mysql_install_db** script is to generate new MySQL privilege tables. It does not overwrite existing MySQL privilege tables, and it does not affect any other data.

If you want to re-create your privilege tables, first stop the **mysqld** server if it's running. Then rename the `mysql` directory under the data directory to save it, and then run **mysql_install_db**. For example:

```
shell> mv mysql-data-directory/mysql mysql-data-directory/mysql-old
shell> mysql_install_db --user=mysql
```

This section lists problems you might encounter when you run **mysql_install_db**:

- **mysql_install_db doesn't install the grant tables**

  You may find that **mysql_install_db** fails to install the grant tables and terminates after displaying the following messages:

  ```
  Starting mysqld daemon with databases from XXXXXX
  mysqld ended
  ```

  In this case, you should examine the error log file very carefully. The log should be located in the directory `XXXXXX` named by the error message, and should indicate why **mysqld** didn't start. If you don't understand what happened, include the log when you post a bug report. See Section 1.4.1.3, “How to Report Bugs or Problems”.

- **There is a mysqld process running**

  This indicates that the server is running, in which case the grant tables have probably been created. If so, you don't have to run **mysql_install_db** at all because it need be run only once (when you install MySQL the first time).

- **Installing a second mysqld server doesn't work when one server is running**

  This can happen when you have an existing MySQL installation, but want to put a new installation in a different location. For example, you might have a production installation, but you want to create a second installation for testing purposes. Generally the problem that occurs when you try to run a second server is that it tries to use a network interface that is in use by the first server. In this case, you should see one of the following error messages:

  ```
  Can't start server: Bind on TCP/IP port:
  Address already in use
  Can't start server: Bind on unix socket...
  ```

  For instructions on setting up multiple servers, see Section 5.10, “Running Multiple MySQL Servers on the Same Machine”.

- **You don't have write access to `/tmp`**

  If you don't have write access to create temporary files or a Unix socket file in the default location (the `/tmp` directory), an error occurs when you run **mysql_install_db** or the **mysqld** server.

  You can specify different temporary directory and Unix socket file locations by executing these commands prior to starting **mysql_install_db** or **mysqld**:

```
shell> TMPDIR=/some_tmp_dir/
shell> MYSQL_UNIX_PORT=/some_tmp_dir/mysql.sock
shell> export TMPDIR MYSQL_UNIX_PORT
```

`some_tmp_dir` should be the full pathname to some directory for which you have write permission.

After this, you should be able to run **mysql_install_db** and start the server with these commands:

```
shell> bin/mysql_install_db --user=mysql
shell> bin/mysqld_safe --user=mysql &
```

If **mysql_install_db** is located in the `scripts` directory, modify the first command to use `scripts/mysql_install_db`.

See Section A.4.5, "How to Protect or Change the MySQL Socket File `/tmp/mysql.sock`". See Appendix F, *Environment Variables*.

2.9.2.2. Starting and Stopping MySQL Automatically

Generally, you start the **mysqld** server in one of these ways:

- By invoking **mysqld** directly. This works on any platform.
- By running the MySQL server as a Windows service. This can be done on versions of Windows that support services (such as NT, 2000, XP, and 2003). The service can be set to start the server automatically when Windows starts, or as a manual service that you start on request. For instructions, see Section 2.3.12, "Starting MySQL as a Windows Service".
- By invoking **mysqld_safe**, which tries to determine the proper options for **mysqld** and then runs it with those options. This script is used on systems based on BSD Unix. See Section 5.1.3, "The **mysqld_safe** Server Startup Script".
- By invoking **mysql.server**. This script is used primarily at system startup and shutdown on systems that use System V-style run directories, where it usually is installed under the name `mysql`. The **mysql.server** script starts the server by invoking **mysqld_safe**. See Section 5.1.4, "The **mysql.server** Server Startup Script".
- On Mac OS X, you can install a separate MySQL Startup Item package to enable the automatic startup of MySQL on system startup. The Startup Item starts the server by invoking **mysql.server**. See Section 2.5, "Installing MySQL on Mac OS X" for details.

The **mysql.server** and **mysqld_safe** scripts and the Mac OS X Startup Item can be used to start the server manually, or automatically at system startup time. **mysql.server** and the Startup Item also can be used to stop the server.

To start or stop the server manually using the **mysql.server** script, invoke it with `start` or `stop` arguments:

```
shell> mysql.server start
shell> mysql.server stop
```

Before **mysql.server** starts the server, it changes location to the MySQL installation directory, and then invokes **mysqld_safe**. If you want the server to run as some specific user, add an appropriate `user` option to the `[mysqld]` group of the `/etc/my.cnf` option file, as shown later in this section. (It is possible that you'll need to edit **mysql.server** if you've installed a binary distribution of MySQL in a nonstandard location. Modify it to `cd` into the proper directory before it runs **mysqld_safe**. If you do this, your modified version of **mysql.server** may be overwritten if you upgrade MySQL in the future, so you should make a copy of your edited version that you can reinstall.)

mysql.server stop brings down the server by sending a signal to it. You can also stop the server manually by executing **mysqladmin shutdown**.

To start and stop MySQL automatically on your server, you need to add start and stop commands to the appropriate places in your `/etc/rc*` files.

If you use the Linux server RPM package (`MySQL-server-`*`VERSION`*`.rpm`), the **mysql.server** script is installed in the `/etc/init.d` directory with the name `mysql`. You need not install it manually. See Section 2.4, “Installing MySQL on Linux” for more information on the Linux RPM packages.

Some vendors provide RPM packages that install a startup script under a different name such as **mysqld**.

If you install MySQL from a source distribution or using a binary distribution format that does not install **mysql.server** automatically, you can install it manually. The script can be found in the `support-files` directory under the MySQL installation directory or in a MySQL source tree.

To install **mysql.server** manually, copy it to the `/etc/init.d` directory with the name **mysql**, and then make it executable. Do this by changing location into the appropriate directory where **mysql.server** is located and executing these commands:

```
shell> cp mysql.server /etc/init.d/mysql
shell> chmod +x /etc/init.d/mysql
```

Older Red Hat systems use the `/etc/rc.d/init.d` directory rather than `/etc/init.d`. Adjust the preceding commands accordingly. Alternatively, first create `/etc/init.d` as a symbolic link that points to `/etc/rc.d/init.d`:

```
shell> cd /etc
shell> ln -s rc.d/init.d .
```

After installing the script, the commands needed to activate it to run at system startup depend on your operating system. On Linux, you can use `chkconfig`:

```
shell> chkconfig --add mysql
```

On some Linux systems, the following command also seems to be necessary to fully enable the **mysql** script:

```
shell> chkconfig --level 345 mysql on
```

On FreeBSD, startup scripts generally should go in `/usr/local/etc/rc.d/`. The `rc(8)` manual page states that scripts in this directory are executed only if their basename matches the `*.sh` shell filename pattern. Any other files or directories present within the directory are silently ignored. In other words, on FreeBSD, you should install the `mysql.server` script as `/usr/local/etc/rc.d/mysql.server.sh` to enable automatic startup.

As an alternative to the preceding setup, some operating systems also use `/etc/rc.local` or `/etc/init.d/boot.local` to start additional services on startup. To start up MySQL using this

method, you could append a command like the one following to the appropriate startup file:

```
/bin/sh -c 'cd /usr/local/mysql; ./bin/mysqld_safe --user=mysql &'
```

For other systems, consult your operating system documentation to see how to install startup scripts.

You can add options for **mysql.server** in a global `/etc/my.cnf` file. A typical `/etc/my.cnf` file might look like this:

```
[mysqld]
datadir=/usr/local/mysql/var
socket=/var/tmp/mysql.sock
port=3306
user=mysql

[mysql.server]
basedir=/usr/local/mysql
```

The **mysql.server** script understands the following options: `basedir`, `datadir`, and `pid-file`. If specified, they *must* be placed in an option file, not on the command line. **mysql.server** understands only `start` and `stop` as command-line arguments.

The following table shows which option groups the server and each startup script read from option files:

Script	Option Groups
mysqld	`[mysqld]`, `[server]`, `[mysqld-major-version]`
mysql.server	`[mysqld]`, `[mysql.server]`
mysqld_safe	`[mysqld]`, `[server]`, `[mysqld_safe]`

`[mysqld-major-version]` means that groups with names like `[mysqld-4.0]`, `[mysqld-4.1]`, and `[mysqld-5.0]` are read by servers having versions 4.0.x, 4.1.x, 5.0.x, and so forth. This feature was added in MySQL 4.0.14. It can be used to specify options that can be read only by servers within a given release series.

For backward compatibility, **mysql.server** also reads the `[mysql_server]` group and **mysqld_safe** also reads the `[safe_mysqld]` group. However, you should update your option files to use the `[mysql.server]` and `[mysqld_safe]` groups instead when you begin using MySQL 4.0 or later.

See Section 4.3.2, “Using Option Files”.

2.9.2.3. Starting and Troubleshooting the MySQL Server

If you have problems starting the server, here are some things you can try:

- Specify any special options needed by the storage engines you are using.
- Make sure that the server knows where to find the data directory.
- Make sure the server can use the data directory. The ownership and permissions of the data directory and its contents must be set such that the server can access and modify them.
- Check the error log to see why the server doesn't start.
- Verify that the network interfaces the server wants to use are available.

Some storage engines have options that control their behavior. You can create a `my.cnf` file and set startup options for the engines you plan to use. If you are going to use storage engines that support transactional tables (`InnoDB`, `BDB`), be sure that you have them configured the way you want before starting the server:

- If you are using `InnoDB` tables, refer to the `InnoDB`-specific startup options. In MySQL 3.23, you must configure `InnoDB` explicitly or the server fails to start. From MySQL 4.0 on, `InnoDB` uses default values for its configuration options if you specify none. See Section 15.4, "`InnoDB` Configuration".
- If you are using `BDB` (Berkeley DB) tables, you should familiarize yourself with the different `BDB`-specific startup options. See Section 14.4.3, "`BDB` Startup Options".

When the **mysqld** server starts, it changes location to the data directory. This is where it expects to find databases and where it expects to write log files. On Unix, the server also writes the pid (process ID) file in the data directory.

The data directory location is hardwired in when the server is compiled. This is where the server looks for the data directory by default. If the data directory is located somewhere else on your system, the server does not work properly. You can find out what the default path settings are by invoking **mysqld** with the `--verbose` and `--help` options. (Prior to MySQL 4.1, omit the `--verbose` option.)

If the defaults don't match the MySQL installation layout on your system, you can override them by specifying options on the command line to **mysqld** or **mysqld_safe**. You can also list the options in an option file.

To specify the location of the data directory explicitly, use the `--datadir` option. However, normally you can tell **mysqld** the location of the base directory under which MySQL is installed and it looks for the data directory there. You can do this with the `--basedir` option.

To check the effect of specifying path options, invoke **mysqld** with those options followed by the `--verbose` and `--help` options. For example, if you change location into the directory where **mysqld** is installed, and then run the following command, it shows the effect of starting the server with a base directory of `/usr/local`:

```
shell> ./mysqld --basedir=/usr/local --verbose --help
```

You can specify other options such as `--datadir` as well, but note that `--verbose` and `--help` must be the last options. (Prior to MySQL 4.1, omit the `--verbose` option.)

Once you determine the path settings you want, start the server without `--verbose` and `--help`.

If **mysqld** is currently running, you can find out what path settings it is using by executing this command:

```
shell> mysqladmin variables
```

Or:

```
shell> mysqladmin -h host_name variables
```

`host_name` is the name of the MySQL server host.

If you get `Errcode 13` (which means `Permission denied`) when starting **mysqld**, this means that the access privileges of the data directory or its contents do not allow the server access. In this case,

you change the permissions for the involved files and directories so that the server has the right to use them. You can also start the server as `root`, but this can raise security issues and should be avoided.

On Unix, change location into the data directory and check the ownership of the data directory and its contents to make sure the server has access. For example, if the data directory is `/usr/local/mysql/var`, use this command:

```
shell> ls -la /usr/local/mysql/var
```

If the data directory or its files or subdirectories are not owned by the account that you use for running the server, change their ownership to that account:

```
shell> chown -R mysql /usr/local/mysql/var
shell> chgrp -R mysql /usr/local/mysql/var
```

If the server fails to start up correctly, check the error log file to see if you can find out why. Log files are located in the data directory (typically `C:\mysql\data` on Windows, `/usr/local/mysql/data` for a Unix binary distribution, and `/usr/local/var` for a Unix source distribution). Look in the data directory for files with names of the form *`host_name`*`.err` and *`host_name`*`.log`, where *`host_name`* is the name of your server host. (Older servers on Windows use `mysql.err` as the error log name.) Then check the last few lines of these files. On Unix, you can use `tail` to display the last few lines:

```
shell> tail host_name.err
shell> tail host_name.log
```

The error log contains information that indicates why the server couldn't start. For example, you might see something like this in the log:

```
000729 14:50:10  bdb:  Recovery function for LSN 1 27595 failed
000729 14:50:10  bdb:  warning: ./test/t1.db: No such file or directory
000729 14:50:10  Can't init databases
```

This means that you didn't start **mysqld** with the `--bdb-no-recover` option and Berkeley DB found something wrong with its own log files when it tried to recover your databases. To be able to continue, you should move away the old Berkeley DB log files from the database directory to some other place, where you can later examine them. The `BDB` log files are named in sequence beginning with `log.0000000001`, where the number increases over time.

If you are running **mysqld** with `BDB` table support and **mysqld** dumps core at startup, this could be due to problems with the `BDB` recovery log. In this case, you can try starting **mysqld** with `--bdb-no-recover`. If that helps, then you should remove all `BDB` log files from the data directory and try starting **mysqld** again without the `--bdb-no-recover` option.

If either of the following errors occur, it means that some other program (perhaps another **mysqld** server) is using the TCP/IP port or Unix socket file that **mysqld** is trying to use:

```
Can't start server: Bind on TCP/IP port: Address already in use
Can't start server: Bind on unix socket...
```

Use **ps** to determine whether you have another **mysqld** server running. If so, shut down the server before starting **mysqld** again. (If another server is running, and you really want to run multiple servers, you can find information about how to do so in Section 5.10, “Running Multiple MySQL Servers on the Same Machine”.)

If no other server is running, try to execute the command `telnet your-host-name tcp-`

`ip-port-number`. (The default MySQL port number is 3306.) Then press Enter a couple of times. If you don't get an error message like `telnet: Unable to connect to remote host: Connection refused`, some other program is using the TCP/IP port that **mysqld** is trying to use. You'll need to track down what program this is and disable it, or else tell **mysqld** to listen to a different port with the `--port` option. In this case, you'll also need to specify the port number for client programs when connecting to the server via TCP/IP.

Another reason the port might be inaccessible is that you have a firewall running that blocks connections to it. If so, modify the firewall settings to allow access to the port.

If the server starts but you can't connect to it, you should make sure that you have an entry in `/etc/hosts` that looks like this:

```
127.0.0.1       localhost
```

This problem occurs only on systems that don't have a working thread library and for which MySQL must be configured to use MIT-pthreads.

If you can't get **mysqld** to start, you can try to make a trace file to find the problem by using the `--debug` option. See Section E.1.2, “Creating Trace Files”.

See Section 2.3.14, “Troubleshooting a MySQL Installation Under Windows”, for more information on troubleshooting Windows installations.

2.9.3. Securing the Initial MySQL Accounts

Part of the MySQL installation process is to set up the `mysql` database containing the grant tables:

- Windows distributions contain preinitialized grant tables that are installed automatically.
- On Unix, the grant tables are populated by the **mysql_install_db** program. Some installation methods run this program for you. Others require that you execute it manually. For details, see Section 2.9.2, “Unix Post-Installation Procedures”.

The grant tables define the initial MySQL user accounts and their access privileges. These accounts are set up as follows:

- Two accounts are created with a username of `root`. These are superuser accounts that can do anything. The initial `root` account passwords are empty, so anyone can connect to the MySQL server as `root` *without a password* and be granted all privileges.
  - On Windows, one `root` account is for connecting from the local host and the other allows connections from any host.
  - On Unix, both `root` accounts are for connections from the local host. Connections must be made from the local host by specifying a hostname of `localhost` for one account, or the actual hostname or IP number for the other.
- Two anonymous-user accounts are created, each with an empty username. The anonymous accounts have no passwords, so anyone can use them to connect to the MySQL server.
  - On Windows, one anonymous account is for connections from the local host. It has all privileges, just like the `root` accounts. The other is for connections from any host and has all privileges for the `test` database or other databases with names that start with `test`.
  - On Unix, both anonymous accounts are for connections from the local host. Connections must be

made from the local host by specifying a hostname of `localhost` for one account, or the actual hostname or IP number for the other. These accounts have all privileges for the `test` database or other databases with names that start with `test_`.

As noted, none of the initial accounts have passwords. This means that your MySQL installation is unprotected until you do something about it:

- If you want to prevent clients from connecting as anonymous users without a password, you should either assign passwords to the anonymous accounts or else remove them.
- You should assign passwords to the MySQL `root` accounts.

The following instructions describe how to set up passwords for the initial MySQL accounts, first for the anonymous accounts and then for the `root` accounts. Replace ``*newpwd*'' in the examples with the actual password that you want to use. The instructions also cover how to remove the anonymous accounts, should you prefer not to allow anonymous access at all.

You might want to defer setting the passwords until later, so that you don't need to specify them while you perform additional setup or testing. However, be sure to set them before using your installation for any real production work.

To assign passwords to the anonymous accounts, you can use either `SET PASSWORD` or `UPDATE`. In both cases, be sure to encrypt the password using the `PASSWORD()` function.

To use `SET PASSWORD` on Windows, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR ''@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR ''@'%' = PASSWORD('newpwd');
```

To use `SET PASSWORD` on Unix, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR ''@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR ''@'host_name' = PASSWORD('newpwd');
```

In the second `SET PASSWORD` statement, replace *`host_name`* with the name of the server host. This is the name that is specified in the `Host` column of the non-`localhost` record for `root` in the `user` table. If you don't know what hostname this is, issue the following statement before using `SET PASSWORD`:

```
mysql> SELECT Host, User FROM mysql.user;
```

Look for the record that has `root` in the `User` column and something other than `localhost` in the `Host` column. Then use that `Host` value in the second `SET PASSWORD` statement.

The other way to assign passwords to the anonymous accounts is by using `UPDATE` to modify the `user` table directly. Connect to the server as `root` and issue an `UPDATE` statement that assigns a value to the `Password` column of the appropriate `user` table records. The procedure is the same for Windows and Unix. The following `UPDATE` statement assigns a password to both anonymous accounts at once:

```
shell> mysql -u root
mysql> UPDATE mysql.user SET Password = PASSWORD('newpwd')
```

```
    ->     WHERE User = '';
mysql> FLUSH PRIVILEGES;
```

After you update the passwords in the `user` table directly using `UPDATE`, you must tell the server to re-read the grant tables with `FLUSH PRIVILEGES`. Otherwise, the change goes unnoticed until you restart the server.

If you prefer to remove the anonymous accounts instead, do so as follows:

```
shell> mysql -u root
mysql> DELETE FROM mysql.user WHERE User = '';
mysql> FLUSH PRIVILEGES;
```

The `DELETE` statement applies both to Windows and to Unix. On Windows, if you want to remove only the anonymous account that has the same privileges as `root`, do this instead:

```
shell> mysql -u root
mysql> DELETE FROM mysql.user WHERE Host='localhost' AND User='';
mysql> FLUSH PRIVILEGES;
```

This account allows anonymous access but has full privileges, so removing it improves security.

You can assign passwords to the `root` accounts in several ways. The following discussion demonstrates three methods:

- Use the `SET PASSWORD` statement
- Use the **mysqladmin** command-line client program
- Use the `UPDATE` statement

To assign passwords using `SET PASSWORD`, connect to the server as `root` and issue two `SET PASSWORD` statements. Be sure to encrypt the password using the `PASSWORD()` function.

For Windows, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'%' = PASSWORD('newpwd');
```

For Unix, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'host_name' = PASSWORD('newpwd');
```

In the second `SET PASSWORD` statement, replace *`host_name`* with the name of the server host. This is the same hostname that you used when you assigned the anonymous account passwords.

To assign passwords to the `root` accounts using **mysqladmin**, execute the following commands:

```
shell> mysqladmin -u root password "newpwd"
shell> mysqladmin -u root -h host_name password "newpwd"
```

These commands apply both to Windows and to Unix. In the second command, replace *host_name* with the name of the server host. The double quotes around the password are not always necessary, but you should use them if the password contains spaces or other characters that are special to your command interpreter.

If you are using a server from a *very* old version of MySQL, the **mysqladmin** commands to set the password fail with the message `parse error near 'SET password'`. The solution to this problem is to upgrade the server to a newer version of MySQL.

You can also use `UPDATE` to modify the `user` table directly. The following `UPDATE` statement assigns a password to both `root` accounts at once:

```
shell> mysql -u root
mysql> UPDATE mysql.user SET Password = PASSWORD('newpwd')
    ->     WHERE User = 'root';
mysql> FLUSH PRIVILEGES;
```

The `UPDATE` statement applies both to Windows and to Unix.

After the passwords have been set, you must supply the appropriate password whenever you connect to the server. For example, if you want to use **mysqladmin** to shut down the server, you can do so using this command:

```
shell> mysqladmin -u root -p shutdown
Enter password: (enter root password here)
```

Note: If you forget your `root` password after setting it up, the procedure for resetting it is covered in Section A.4.1, “How to Reset the Root Password”.

To set up new accounts, you can use the `GRANT` statement. For instructions, see Section 5.6.2, “Adding New User Accounts to MySQL”.

2.10. Upgrading MySQL

As a general rule, we recommend that when upgrading from one release series to another, you should go to the next series rather than skipping a series. For example, if you currently are running MySQL 3.23 and wish to upgrade to a newer series, upgrade to MySQL 4.0 rather than to 4.1 or 5.0.

The following items form a checklist of things you should do whenever you perform an upgrade:

- Read the upgrading section for the release series to which you are upgrading. Read the change notes as well. These provide information about new features you can use. For example, before upgrading from MySQL 4.1 to 5.0, read the 5.0 upgrading section (see Section 2.10.1, “Upgrading from Version 4.1 to 5.0”) and read the 5.0 change notes (see Appendix D, *MySQL Change History*).
- Before you do an upgrade, back up your databases.
- If you are running MySQL Server on Windows, see Section 2.3.15, “Upgrading MySQL on Windows”.
- An upgrade may involve changes to the grant tables that are stored in the `mysql` database. Occasionally new columns or tables are added to support new features. To take advantage of these features, be sure that your grant tables are up to date. The upgrade procedure is described in Section 2.10.7, “Upgrading the Grant Tables”.
- If you are using replication, see Section 6.6, “Upgrading a Replication Setup” for information on up-

grading your replication setup.

- If you install a MySQL-Max distribution that includes a server named **mysqld-max**, then upgrade later to a non-Max version of MySQL, **mysqld_safe** still attempts to run the old **mysqld-max** server. If you perform such an upgrade, you should manually remove the old **mysqld-max** server to ensure that **mysqld_safe** runs the new **mysqld** server.

You can always move the MySQL format files and data files between different versions on the same architecture as long as you stay within versions for the same release series of MySQL. The current production release series is 4.1. If you change the character set when running MySQL, you must run **myisamchk -r -q --set-character-set=*charset*** on all `MyISAM` tables. Otherwise, your indexes may not be ordered correctly, because changing the character set may also change the sort order.

Normally you can upgrade MySQL to a newer MySQL version without having to do any changes to your tables. Please confirm if the upgrade notes to the particular version you are upgrading to tells you anything about this. If there would be any incompatibilities you can use **mysqldump** to dump your tables before upgrading. After upgrading, reload the dump file using **mysql** or `mysqlimport` to recreate your tables.

If you are cautious about using new versions, you can always rename your old **mysqld** before installing a newer one. For example, if you are using MySQL 4.0.18 and want to upgrade to 4.1.1, rename your current server from **mysqld** to **mysqld-4.0.18**. If your new **mysqld** then does something unexpected, you can simply shut it down and restart with your old **mysqld**.

If, after an upgrade, you experience problems with recompiled client programs, such as `Commands out of sync` or unexpected core dumps, you probably have used old header or library files when compiling your programs. In this case, you should check the date for your `mysql.h` file and `libmysqlclient.a` library to verify that they are from the new MySQL distribution. If not, recompile your programs with the new headers and libraries.

If problems occur, such as that the new **mysqld** server doesn't want to start or that you can't connect without a password, verify that you don't have some old `my.cnf` file from your previous installation. You can check this with the `--print-defaults` option (for example, **mysqld --print-defaults**). If this displays anything other than the program name, you have an active `my.cnf` file that affects server or client operation.

It is a good idea to rebuild and reinstall the Perl `DBD::mysql` module whenever you install a new release of MySQL. The same applies to other MySQL interfaces as well, such as the PHP `mysql` extension and the Python `MySQLdb` module.

2.10.1. Upgrading from Version 4.1 to 5.0

In general, you should do the following when upgrading to MySQL 5.0 from 4.1:

- Check the items in the change list found later in this section to see whether any of them might affect your applications. Note particularly any that are marked **Incompatible change**; these result in incompatibilities with earlier versions of MySQL, and may require your attention *before you upgrade*.

- Read the 5.0 news items to see what significant new features you can use in 5.0. See Section D.1, “Changes in release 5.0.x (Development)”.

- If you are running MySQL Server on Windows, see Section 2.3.15, “Upgrading MySQL on Windows”. Also, two of the Windows MySQL servers were renamed. See Section 2.3.9, “Selecting a MySQL Server type”.

- MySQL 5.0 adds support for stored procedures. This support requires the `proc` table in the `mysql`

database. To create this file, you should run the **mysql_fix_privilege_tables** script as described in Section 2.10.7, “Upgrading the Grant Tables”.

- MySQL 5.0 adds support for views. This support requires the extra privilege columns in the `user` and `db` tables in the `mysql` database. To create these columns, you should run the **mysql_fix_privilege_tables** script as described in Section 2.10.7, “Upgrading the Grant Tables”.
- If you are using replication, see Section 6.6, “Upgrading a Replication Setup” for information on upgrading your replication setup.

Several visible behaviors have changed between MySQL 4.1 and mySQL 5.0 to make MySQL more compatible with standard SQL. These changes may affect your applications.

The following list describes changes that may affect applications and that you should watch out for when upgrading to version 5.0.

Server Changes:

- **Incompatible change:** As of MySQL 5.0.3, the server by default no longer loads user-defined functions unless they have at least one auxiliary symbol defined in addition to the main function symbol. This behavior can be overridden with the `--allow-suspicious-udfs` option. See Section 25.2.3.6, “User-defined Function Security Precautions”.
- **Incompatible change:** The update log is removed, so if you enabled it in the past, you should enable the binary log instead. See Section D.1, “Changes in release 5.0.x (Development)”.
- **Incompatible change:** Support for the `ISAM` storage engine has been removed. If you have `ISAM` tables, you should convert them before upgrading. For example, to convert an `ISAM` table to use the `MyISAM` storage engine, use this statement:

  ```
  ALTER TABLE tbl_name ENGINE = MyISAM;
  ```

  Use a similar statement for each `ISAM` table.
- **Incompatible change:** Support for `RAID` options in `MyISAM` tables has been removed. If you have tables that use these options, you should convert them before upgrading. One way to do this is to dump them with **mysqldump**, edit the dump file to remove the `RAID` options in the `CREATE TABLE` statements, and reload the dump file. Another way to convert a table is to use `ALTER TABLE` to convert it to use a different storage engine. The use `ALTER TABLE` to convert it back if you want to continue to use the table as a `MyISAM` table. This approach requires some care, however. It cannot be used if the table contains `MyISAM`-only features such as `FULLTEXT` indexes or columns with spatial data types.

  The `.MYD` files for `RAID` tables in a given database are stored under the database directory in subdirectories that have names consisting of two hex digits in the range from `00` to `ff`. After converting all tables that use `RAID` options, these `RAID`-related subdirectories still will exist but can be removed. Verify that they are empty, and then remove them manually. (If they are not empty, there is some `RAID` table that has not been converted.)

SQL Changes:

- As of MySQL 5.0.3, trailing spaces no longer are removed from values stored in `VARCHAR` and `VARBINARY` columns. The maximum length for `VARCHAR` or `VARBINARY` now is 65,535 characters or bytes, respectively.

- As of MySQL 5.0.3, `BIT` is a separate data type, not a synonym for `TINYINT(1)`. See Section 11.1.1, “Overview of Numeric Types”.
- MySQL 5.0.2 adds several SQL modes that allow stricter control over rejecting records that have invalid or missing values. See Section 5.2.2, “The Server SQL Mode”. See Section 1.5.6.2, “Constraints on Invalid Data”. If you want to enable this control but continue to use MySQL's capability for storing incorrect dates such as `'2004-02-31'`, you should start the server with `--sql_mode=TRADITIONAL,ALLOW_INVALID_DATES`.
- As of MySQL 5.0.2, the `SCHEMA` and `SCHEMAS` keywords are accepted as synonyms for `DATABASE` and `DATABASES`.
- User variables are not case sensitive. In MySQL 4.1, `SET @x = 0; SET @X = 1; SELECT @x;` creates two variables and returns `0`. In MySQL 5.0, it creates one variable and returns `1`.

C API Changes:

- The `reconnect` flag in the `MYSQL` structure is set to 0 by `mysql_real_connect()`. Only those client programs which didn't explicitly set this flag to 0 or 1 after `mysql_real_connect()` experience a change. Having automatic reconnection enabled by default was considered too dangerous (after reconnection, table locks, temporary tables, user and session variables are lost).

2.10.2. Upgrading from Version 4.0 to 4.1

In general, you should do the following when upgrading to MySQL 4.1 from 4.0:

- Check the items in the change list found later in this section to see whether any of them might affect your applications. Note particularly any that are marked **Incompatible change**; these result in incompatibilities with earlier versions of MySQL.
- Read the 4.1 news items to see what significant new features you can use in 4.1. See Section D.2, “Changes in release 4.1.x (Production)”.
- Character set support has been improved. The server supports multiple character sets.
- **Important note:** MySQL 4.1 stores table names and column names in `UTF8`. If you have table names or column names that use characters outside of the standard 7-bit US-ASCII range, you may have to do a `mysqldump` of your tables in MySQL 4.0 and restore them after upgrading to MySQL 4.1. The symptom for this problem is that you get a `table not found` error when trying to access your tables. In this case, you should be able to downgrade back to MySQL 4.0 and access your data.
- If you are running MySQL Server on Windows, see Section 2.3.15, “Upgrading MySQL on Windows”.
- After upgrading, update the grant tables to have the new longer `Password` column that is needed for more secure handling of passwords. The procedure uses **mysql_fix_privilege_tables** and is described in Section 2.10.7, “Upgrading the Grant Tables”. If you don't do this, MySQL does not use the new more secure protocol to authenticate. Implications of the password-handling change for applications are given later in this section.
- If you are using replication, see Section 6.6, “Upgrading a Replication Setup” for information on upgrading your replication setup.

- The Berkeley DB table handler is updated to DB 4.1 (from 3.2) which has a new log format. If you have to downgrade back to 4.0 you must use **mysqldump** to dump your `BDB` tables in text format and delete all `log.XXXXXXXXXX` files before you start MySQL 4.0 and reload the data.

- MySQL 4.1.3 introduces support for per-connection time zones. See Section 5.8.8, “MySQL Server Time Zone Support”. To enable recognition of named time zones, you should create the time zone tables in the `mysql` database. For instructions, see Section 2.9, “Post-Installation Setup and Testing”.

- If you are using an old `DBD-mysql` module (`Msql-MySQL-modules`) you have to upgrade to use the newer `DBD-mysql` module. Anything above `DBD-mysql` 2.xx should be fine.

  If you don't upgrade, some methods (such as `DBI->do()`) do not notice error conditions correctly.

- The `--defaults-file=option-file-name` option gives an error if the option file doesn't exist.

- Some notes about upgrading from MySQL 4.0 to MySQL 4.1 on Netware: Make sure to upgrade Perl and PHP versions. Download Perl 5 for Netware from http://forge.novell.com/modules/xfmod/project/?perl5 and PHP from http://forge.novell.com/modules/xfmod/project/?php. Download and install Perl module for MySQL 4.1 from http://forge.novell.com/modules/xfmod/project/showfiles.php?group_id=1126 and PHP Extension for MySQL 4.1 from http://forge.novell.com/modules/xfmod/project/showfiles.php?group_id=1078.

Several visible behaviors have changed between MySQL 4.0 and MySQL 4.1 to fix some critical bugs and make MySQL more compatible with standard SQL. These changes may affect your applications.

Some of the 4.1 behaviors can be tested in 4.0 before performing a full upgrade to 4.1. We have added to later MySQL 4.0 releases (from 4.0.12 on) a `--new` startup option for **mysqld**. See Section 5.2.1, “**mysqld** Command-Line Options”.

This option gives you the 4.1 behavior for the most critical changes. You can also enable these behaviors for a given client connection with the `SET @@new=1` command, or turn them off if they are on with `SET @@new=0`.

If you believe that some of the 4.1 changes affect you, we recommend that before upgrading to 4.1, you download the latest MySQL 4.0 version and run it with the `--new` option by adding the following to your config file:

```
[mysqld-4.0]
new
```

That way you can test the new behaviors in 4.0 to make sure that your applications work with them. This helps you have a smooth, painless transition when you perform a full upgrade to 4.1 later. Putting the `--new` option in the `[mysqld-4.0]` option group ensures that you don't accidentally later run the 4.1 version with the `--new` option.

The following lists describe changes that may affect applications and that you should watch out for when upgrading to version 4.1.

Server Changes:

- All tables and string columns have a character set. See Chapter 10, *Character Set Support*. Character set information is displayed by `SHOW CREATE TABLE` and **mysqldump**. (MySQL versions 4.0.6 and above can read the new dump files; older versions cannot.) This change should not affect applic-

ations that use only one character set.

- If you have table columns that store character data represented in a character set that the 4.1 server supports directly, you can convert the columns to the proper character set using the instructions in Section 10.10.2, “Converting 4.0 Character Columns to 4.1 Format”. Also, database, table, and column identifiers are stored internally using Unicode (UTF8) regardless of the default character set. See Section 9.2, “Database, Table, Index, Column, and Alias Names”.

- **Incompatible change:** There are conditions under which you should rebuild tables. In general, to rebuild a table, dump it with **mysqldump** and reload the dump file. Some items in the following list indicate alternatives means for rebuilding.

  - If you have created or used `InnoDB` tables with `TIMESTAMP` columns in MySQL versions 4.1.0 to 4.1.3, you have to rebuild those tables when you upgrade to MySQL 4.1.4 or later. The storage format in those MySQL versions for a `TIMESTAMP` column was incorrect. If you upgrade from MySQL 4.0 to 4.1.4 or later, then no rebuild of tables with `TIMESTAMP` columns is needed.

  - Starting from MySQL 4.1.3, `InnoDB` uses the same character set comparison functions as MySQL for non-`latin1_swedish_ci` character strings that are not `BINARY`. This changes the sorting order of space and characters with a code < ASCII(32) in those character sets. For `latin1_swedish_ci` character strings and `BINARY` strings, `InnoDB` uses its own pad-spaces-at-end comparison method, which stays unchanged. If you have an `InnoDB` table created with MySQL 4.1.2 or earlier, with an index on a non-`latin1` character set (in the case of 4.1.0 and 4.1.1, with any character set) and the table contains any `CHAR`/`VARCHAR`/or `TEXT` columns that are not `BINARY` but may contain characters with a code < ASCII(32), then you should do `ALTER TABLE` or `OPTIMIZE TABLE` on it to regenerate the index, after upgrading to MySQL 4.1.3 or later. Also, `MyISAM` tables have to be rebuilt or repaired in these cases.

  - If you have used column prefix indexes on UTF8 columns or other multi-byte character set columns in MySQL 4.1.0 to 4.1.5, you must rebuild the tables when you upgrade to MySQL 4.1.6 or later.

  - If you have used accent characters (characters with byte values of 128 to 255) in database names, table names, constraint names, or column names in versions of MySQL earlier than 4.1, you cannot upgrade to MySQL 4.1 directly, because 4.1 uses UTF8 to store metadata names. Use `RENAME TABLE` to overcome this if the accent character is in the table name or the database name, or rebuild the table.

  - String comparison works according to SQL standard: Instead of stripping end spaces before comparison, we now extend the shorter string with spaces. The problem with this is that now `'a' > 'a\t'`, which it wasn't before. If you have any tables where you have an indexed `CHAR`, `VARCHAR` or `TEXT` column in which the last character in the index may be less than `ASCII(32)`, you should use `REPAIR TABLE` or **mysqlcheck** to ensure that the table is correct.

- **Incompatible change:** MySQL interprets length specifications in character column definitions in characters. (Earlier versions interpret them in bytes.) For example, `CHAR(N)` means *N* characters, not *N* bytes.

  For single-byte character sets, this change makes no difference. However, if you upgrade to MySQL 4.1 and configure the server to use a multi-byte character set, the apparent length of character columns changes. Suppose that a 4.0 table contains a `CHAR(8)` column used to store `ujis` characters. Eight bytes can store from two to four `ujis` characters. If you upgrade to 4.1 and configure the server to use `ujis` as its default character set, the server interprets character column lengths based on the maximum size of a `ujis` character, which is three bytes. The number of three-byte characters that fit in eight bytes is two. Consequently, if you use `SHOW CREATE TABLE` to view the table definition, MySQL displays `CHAR(2)`. You can retrieve existing data from the table, but you can only store new values containing up to two characters. To correct this issue, use `ALTER TABLE` to change the column definition. For example:

```
ALTER TABLE tbl_name MODIFY col_name CHAR(8);
```

- The table definition format used in `.frm` files has changed slightly in 4.1. MySQL 4.0 versions from 4.0.11 on can read the new `.frm` format directly, but older versions cannot. If you need to move tables from 4.1 to a version earlier than 4.0.11, you should use **mysqldump**. See Section 8.8, “The **mysqldump** Database Backup Program”.
- **Important note:** If you upgrade to MySQL 4.1.1 or higher, it is difficult to downgrade back to 4.0 or 4.1.0! That is because, for earlier versions, `InnoDB` is not aware of multiple tablespaces.
- **Incompatible change:** In connection with the support for per-connection time zones in MySQL 4.1.3, the `timezone` system variable was renamed to `system_time_zone`.
- Windows servers support connections from local clients using shared memory if run with the `--shared-memory` option. If you are running multiple servers this way on the same Windows machine, you should use a different `--shared-memory-base-name` option for each server.
- **Incompatible change:** The interface to aggregate user-defined functions has changed a bit as of MySQL 4.1.1. You must declare a `xxx_clear()` function for each aggregate function `XXX()`. `xxx_clear()` is used instead of `xxx_reset()`. See Section 25.2.3.2, “UDF Calling Sequences for aggregate functions”.
- As of MySQL 4.1.10a, the server by default no longer loads user-defined functions unless they have at least one auxiliary symbol defined in addition to the main function symbol. This behavior can be overridden with the `--allow-suspicious-udfs` option. See Section 25.2.3.6, “User-defined Function Security Precautions”.

Client Changes:

- **mysqldump** has the `--opt` and `--quote-names` options enabled by default. You can turn them off with `--skip-opt` and `--skip-quote-names`.

SQL Changes:

- **Incompatible change:** In MySQL 4.1.2, the `Type` column in the output from `SHOW TABLE STATUS` was renamed to `Engine`.
- **Incompatible change:** String comparison works according to SQL standard: Instead of stripping end spaces before comparison, we now extend the shorter string with spaces. The problem with this is that now `'a' > 'a\t'`, which it wasn't before. If you have any tables where you have an indexed `CHAR`, `VARCHAR` or `TEXT` column in which the last character in the index may be less than `ASCII(32)`, you should use `REPAIR TABLE` or **mysqlcheck** to ensure that the table is correct.
- When using multiple-table `DELETE` statements, you should use the alias of the tables from which you want to delete, not the actual table name. For example, instead of doing this:

  ```
  DELETE test FROM test AS t1, test2 WHERE ...
  ```

  Do this:

  ```
  DELETE t1 FROM test AS t1, test2 WHERE ...
  ```

This corrects a problem that was present in MySQL 4.0.

- **Incompatible change:** `TIMESTAMP` is returned as a string in `'YYYY-MM-DD HH:MM:SS'` format (from 4.0.12 the `--new` option can be used to make a 4.0 server behave as 4.1 in this respect). See Section 11.3.1.2, "`TIMESTAMP` Properties as of MySQL 4.1".

  If you want to have the value returned as a number (as MySQL 4.0 does) you should add `+0` to `TIMESTAMP` columns when you retrieve them:

  ```
  mysql> SELECT ts_col + 0 FROM tbl_name;
  ```

  Display widths for `TIMESTAMP` columns are no longer supported. For example, if you declare a column as `TIMESTAMP(10)`, the `(10)` is ignored.

  These changes were necessary for SQL standards compliance. In a future version, a further change will be made (backward compatible with this change), allowing the timestamp length to indicate the desired number of digits for fractions of a second.

- **Incompatible change:** Binary values such as `0xFFDF` are assumed to be strings instead of numbers. This fixes some problems with character sets where it's convenient to input a string as a binary value. With this change, you should use `CAST()` if you want to compare binary values numerically as integers:

  ```
  mysql> SELECT CAST(0xFEFF AS UNSIGNED INTEGER)
      ->        < CAST(0xFF AS UNSIGNED INTEGER);
          -> 0
  ```

  If you don't use `CAST()`, a lexical string comparison is done:

  ```
  mysql> SELECT 0xFEFF < 0xFF;
          -> 1
  ```

  Using binary items in a numeric context or comparing them using the `=` operator should work as before. (The `--new` option can be used from 4.0.13 on to make a 4.0 server behave as 4.1 in this respect.)

- For functions that produce a `DATE`, `DATETIME`, or `TIME` value, the result returned to the client is fixed up to have a temporal type. For example, in MySQL 4.1, you get this result:

  ```
  mysql> SELECT CAST('2001-1-1' AS DATETIME);
         -> '2001-01-01 00:00:00'
  ```

  In MySQL 4.0, the result is different:

  ```
  mysql> SELECT CAST('2001-1-1' AS DATETIME);
         -> '2001-01-01'
  ```

- `DEFAULT` values no longer can be specified for `AUTO_INCREMENT` columns. (In 4.0, a `DEFAULT` value is silently ignored; in 4.1, an error occurs.)

- `LIMIT` no longer accepts negative arguments. Use some large number (maximum 18446744073709551615) instead of -1.

- `SERIALIZE` is no longer a valid mode value for the `sql_mode` variable. You should use `SET TRANSACTION ISOLATION LEVEL SERIALIZABLE` instead. `SERIALIZE` is no longer valid

for the `--sql-mode` option for **mysqld**, either. Use `--transaction-isolation=SERIALIZABLE` instead.

- **Incompatible change:** Before MySQL 4.1.1, the statement parser was less strict and its string-to-date conversion would ignore everything up to the first digit. As a result, invalid statements such as the following were accepted:

  ```
  INSERT INTO t (datetime_col) VALUES ('stuff 2005-02-11 10:17:01');
  ```

  As of MySQL 4.1.1, the parser is more strict and treats the string as an invalid date, so the preceding statement results in a warning.

C API Changes:

- **Incompatible change:** The `mysql_shutdown()` C API function has an extra parameter as of MySQL 4.1.3: `SHUTDOWN`-level. You should convert any `mysql_shutdown(X)` call you have in your application to `mysql_shutdown(X,SHUTDOWN_DEFAULT)`.

- Some C API calls such as `mysql_real_query()` return `1` on error, not `-1`. You may have to change some old applications if they use constructs like this:

  ```
  if (mysql_real_query(mysql_object, query, query_length) == -1)
  {
    printf("Got error");
  }
  ```

  Change the call to test for a non-zero value instead:

  ```
  if (mysql_real_query(mysql_object, query, query_length) != 0)
  {
    printf("Got error");
  }
  ```

Password-Handling Changes:

The password hashing mechanism has changed in 4.1 to provide better security, but this may cause compatibility problems if you still have clients that use the client library from 4.0 or earlier. (It is very likely that you have 4.0 clients in situations where clients connect from remote hosts that have not yet upgraded to 4.1.) The following list indicates some possible upgrade strategies. They represent various tradeoffs between the goal of compatibility with old clients and the goal of security.

- Only upgrade the client to use 4.1 client libraries (not the server). No behavior changes (except the return value of some API calls), but you cannot use any of the new features provided by the 4.1 client/server protocol, either. (MySQL 4.1 has an extended client/server protocol that offers such features as prepared statements and multiple result sets.) See Section 22.2.4, “C API Prepared Statements”.

- Upgrade to 4.1 and run the **mysql_fix_privilege_tables** script to widen the `Password` column in the `user` table so that it can hold long password hashes. But run the server with the `--old-passwords` option to provide backward compatibility that allows pre-4.1 clients to continue to connect to their short-hash accounts. Eventually, when all your clients are upgraded to 4.1, you can stop using the `--old-passwords` server option. You can also change the passwords for your MySQL accounts to use the new more secure format. A pure-4.1 installation is the most secure.

Further background on password hashing with respect to client authentication and password-changing operations may be found in Section 5.5.9, “Password Hashing in MySQL 4.1” and Section A.2.3, “`Client does not support authentication protocol`”.

2.10.3. Upgrading from Version 3.23 to 4.0

In general, you should do the following when upgrading to MySQL 4.0 from 3.23:

- Check the items in the change list found later in this section to see whether any of them might affect your applications. Note particularly any that are marked **Incompatible change**; these result in incompatibilities with earlier versions of MySQL.
- Read the 4.0 news items to see what significant new features you can use in 4.0. See Section D.3, “Changes in release 4.0.x (Production)”.
- If you are running MySQL Server on Windows, see Section 2.3.15, “Upgrading MySQL on Windows”.
- After upgrading, update the grant tables to add new privileges and features. The procedure uses the **mysql_fix_privilege_tables** script and is described in Section 2.10.7, “Upgrading the Grant Tables”.
- If you are using replication, see Section 6.6, “Upgrading a Replication Setup” for information on upgrading your replication setup.
- Edit any MySQL startup scripts or option files to not use any of the deprecated options described later in this section.
- Convert your old `ISAM` files to `MyISAM` files. One way to do this is with the `mysql_convert_table_format` script. (This is a Perl script; it requires that DBI be installed.) To convert the tables in a given database, use this command:

  ```
  shell> mysql_convert_table_format database db_name
  ```

  Note that this should be used only if all tables in the given database are `ISAM` or `MyISAM` tables. To avoid converting tables of other types to `MyISAM`, you can explicitly list the names of your `ISAM` tables after the database name on the command line.

  Individual tables can be changed to `MyISAM` by using the following `ALTER TABLE` statement for each table to be converted:

  ```
  mysql> ALTER TABLE tbl_name TYPE=MyISAM;
  ```

  If you are not sure of the table type for a given table, use this statement:

  ```
  mysql> SHOW TABLE STATUS LIKE 'tbl_name';
  ```

- Ensure that you don't have any MySQL clients that use shared libraries (like the Perl `DBD::mysql` module). If you do, you should recompile them, because the data structures used in `libmysqlclient.so` have changed. The same applies to other MySQL interfaces as well, such as the Python `MySQLdb` module.

MySQL 4.0 works even if you don't perform the preceding actions, but you cannot use the new security privileges in MySQL 4.0 and you may run into problems when upgrading later to MySQL 4.1 or newer. The `ISAM` file format still works in MySQL 4.0, but is deprecated and is not compiled in by default as

of MySQL 4.1. MyISAM tables should be used instead.

Old clients should work with a MySQL 4.0 server without any problems.

Even if you perform the preceding actions, you can still downgrade to MySQL 3.23.52 or newer if you run into problems with the MySQL 4.0 series. In this case, you must use **mysqldump** to dump any tables that use full-text indexes and reload the dump file into the 3.23 server. This is necessary because 4.0 uses a new format for full-text indexing.

The following lists describe changes that may affect applications and that you should watch out for when upgrading to version 4.0.

Server Changes:

- As of MySQL 4.0.24, the server by default no longer loads user-defined functions unless they have at least one auxiliary symbol defined in addition to the main function symbol. This behavior can be overridden with the --allow-suspicious-udfs option. See Section 25.2.3.6, “User-defined Function Security Precautions”.

- MySQL 4.0 has a lot of new privileges in the mysql.user table. See Section 5.5.3, “Privileges Provided by MySQL”.

  To get these new privileges to work, you must update the grant tables. The procedure is described in Section 2.10.7, “Upgrading the Grant Tables”. Until you do this, all accounts have the SHOW DATABASES, CREATE TEMPORARY TABLES, and LOCK TABLES privileges. SUPER and EXECUTE privileges take their value from PROCESS. REPLICATION SLAVE and REPLICATION CLIENT take their values from FILE.

  If you have any scripts that create new MySQL user accounts, you may want to change them to use the new privileges. If you are not using GRANT commands in the scripts, this is a good time to change your scripts to use GRANT instead of modifying the grant tables directly.

  From version 4.0.2 on, the option --safe-show-database is deprecated (and no longer does anything). See Section 5.4.3, “Startup Options for **mysqld** Concerning Security”.

  If you get Access denied errors for new users in version 4.0.2 and up, you should check whether you need some of the new grants that you didn't need before. In particular, you need REPLICATION SLAVE (instead of FILE) for new slave servers.

- **safe_mysqld** has been renamed to **mysqld_safe**. For backward compatibility, binary distributions will for some time include **safe_mysqld** as a symlink to **mysqld_safe**.

- InnoDB support is included by default in binary distributions. If you build MySQL from source, InnoDB is configured in by default. If you do not use InnoDB and want to save memory when running a server that has InnoDB support enabled, use the --skip-innodb server startup option. To compile MySQL without InnoDB support, run **configure** with the --without-innodb option.

- Values for the startup parameters myisam_max_extra_sort_file_size and myisam_max_extra_sort_file_size are given in bytes (they were given in megabytes before 4.0.3).

- **mysqld** has the option --temp-pool enabled by default because this gives better performance with some operating systems (most notably Linux).

- The **mysqld** startup options --skip-locking and --enable-locking were renamed to --skip-external-locking and --external-locking.

- External system locking of MyISAM/ISAM files is turned off by default. You can turn this on with --external-locking. (However, this is never needed for most users.)

- The following startup variables and options have been renamed:

Old Name	New Name
myisam_bulk_insert_tree_size	bulk_insert_buffer_size
query_cache_startup_type	query_cache_type
record_buffer	read_buffer_size
record_rnd_buffer	read_rnd_buffer_size
sort_buffer	sort_buffer_size
warnings	log-warnings
--err-log	--log-error (for **mysqld_safe**)

  The startup options record_buffer, sort_buffer, and warnings still work in MySQL 4.0 but are deprecated.

SQL Changes:

- The following SQL variables have been renamed:

Old Name	New Name
SQL_BIG_TABLES	BIG_TABLES
SQL_LOW_PRIORITY_UPDATES	LOW_PRIORITY_UPDATES
SQL_MAX_JOIN_SIZE	MAX_JOIN_SIZE
SQL_QUERY_CACHE_TYPE	QUERY_CACHE_TYPE

  The old names still work in MySQL 4.0 but are deprecated.

- You have to use SET GLOBAL SQL_SLAVE_SKIP_COUNTER=skip_count instead of SET SQL_SLAVE_SKIP_COUNTER=skip_count.
- SHOW MASTER STATUS returns an empty set if binary logging is not enabled.
- SHOW SLAVE STATUS returns an empty set if the slave is not initialized.
- SHOW INDEX has two more columns than it had in 3.23 (Null and Index_type).
- The format of SHOW OPEN TABLES has changed.
- ORDER BY col_name DESC sorts NULL values last, as of MySQL 4.0.11. In 3.23 and in earlier 4.0 versions, this was not always consistent.
- CHECK, LOCALTIME, and LOCALTIMESTAMP are reserved words.
- DOUBLE and FLOAT columns honor the UNSIGNED flag on storage (before, UNSIGNED was ignored for these columns).
- The result of all bitwise operators (|, &, <<, >>, and ~) is unsigned. This may cause problems if you are using them in a context where you want a signed result. See Section 12.7, “Cast Functions and Operators”.

  Note: When you use subtraction between integer values where one is of type UNSIGNED, the result

is unsigned. In other words, before upgrading to MySQL 4.0, you should check your application for cases in which you are subtracting a value from an unsigned entity and want a negative answer or subtracting an unsigned value from an integer column. You can disable this behavior by using the `--sql-mode=NO_UNSIGNED_SUBTRACTION` option when starting **mysqld**. See Section 5.2.2, “The Server SQL Mode”.

- You should use integers to store values in `BIGINT` columns (instead of using strings, as you did in MySQL 3.23). Using strings still works, but using integers is more efficient.
- In MySQL 3.23, `INSERT INTO ... SELECT` always had `IGNORE` enabled. As of 4.0.1, MySQL stops (and possibly rolls back) by default in case of an error unless you specify `IGNORE`.
- You should use `TRUNCATE TABLE` when you want to delete all rows from a table and you don't need to obtain a count of the number of rows that were deleted. (`DELETE FROM` *`tbl_name`* returns a row count in 4.0 and doesn't reset the `AUTO_INCREMENT` counter, and `TRUNCATE TABLE` is faster.)
- You get an error if you have an active transaction or `LOCK TABLES` statement when trying to execute `TRUNCATE TABLE` or `DROP DATABASE`.
- To use `MATCH ... AGAINST (... IN BOOLEAN MODE)` full-text searches with your tables, you must rebuild their indexes with `REPAIR TABLE tbl_name USE_FRM`. If you attempt a boolean full-text search without rebuilding the indexes this way, the search returns incorrect results. See Section 12.6.4, “Fine-Tuning MySQL Full-Text Search”.
- `LOCATE()` and `INSTR()` are case sensitive if one of the arguments is a binary string. Otherwise they are case insensitive.
- `STRCMP()` uses the current character set when performing comparisons. This makes the default comparison behavior not case sensitive unless one or both of the operands are binary strings.
- `HEX(`*`str`*`)` returns the characters in *`str`* converted to hexadecimal. If you want to convert a number to hexadecimal, you should ensure that you call `HEX()` with a numeric argument.
- `RAND(seed)` returns a different random number series in 4.0 than in 3.23; this was done to further differentiate `RAND(seed)` and `RAND(seed+1)`.
- The default type returned by `IFNULL(A,B)` is set to be the more ``general'' of the types of `A` and `B`. (The general-to-specific order is string, `REAL`, `INTEGER`).

C API Changes:

- The old C API functions `mysql_drop_db()`, `mysql_create_db()`, and `mysql_connect()` are no longer supported unless you compile MySQL with `CFLAGS=-DUSE_OLD_FUNCTIONS`. However, it is preferable to change client programs to use the new 4.0 API instead.
- In the `MYSQL_FIELD` structure, `length` and `max_length` have changed from `unsigned int` to `unsigned long`. This should not cause any problems, except that they may generate warning messages when used as arguments in the `printf()` class of functions.
- Multi-threaded clients should use `mysql_thread_init()` and `mysql_thread_end()`. See Section 22.2.15, “How to Make a Threaded Client”.

Other Changes:

- If you want to recompile the Perl `DBD::mysql` module, use a recent version. Version 2.9003 is recommended. Versions older than 1.2218 should not be used because they use the deprecated `mysql_drop_db()` call.

2.10.4. Upgrading from Version 3.22 to 3.23

MySQL 3.22 and 3.21 clients work without any problems with a MySQL 3.23 server.

When upgrading to MySQL 3.23 from an earlier version, note the following changes:

Table Changes:

- MySQL 3.23 supports tables of the new `MyISAM` type and the old `ISAM` type. By default, all new tables are created with type `MyISAM` unless you start **mysqld** with the `--default-table-type=isam` option. You don't have to convert your old `ISAM` tables to use them with MySQL 3.23. You can convert an `ISAM` table to `MyISAM` format with `ALTER TABLE` *`tbl_name`* `TYPE=MyISAM` or the Perl script `mysql_convert_table_format`.
- All tables that use the `tis620` character set must be fixed with **myisamchk -r** or `REPAIR TABLE`.
- If you are using the `german` character sort order for `ISAM` tables, you must repair them with **isamchk -r**, because we have made some changes in the sort order.

Client Program Changes:

- The MySQL client **mysql** is by default started with the `--no-named-commands (-g)` option. This option can be disabled with `--enable-named-commands (-G)`. This may cause incompatibility problems in some cases---for example, in SQL scripts that use named commands without a semicolon. Long format commands still work from the first line.
- If you want your **mysqldump** files to be compatible between MySQL 3.22 and 3.23, you should not use the `--opt` or `--all` option to **mysqldump**.

SQL Changes:

- If you do a `DROP DATABASE` on a symbolically linked database, both the link and the original database are deleted. This didn't happen in MySQL 3.22 because **configure** didn't detect the availability of the `readlink()` system call.
- `OPTIMIZE TABLE` works only for `MyISAM` tables. For other table types, you can use `ALTER TABLE` to optimize the table. During `OPTIMIZE TABLE`, the table is locked to prevent it from being used by other threads.
- Date functions that work on parts of dates (such as `MONTH()`) return 0 for `0000-00-00` dates. In MySQL 3.22, these functions returned `NULL`.
- The default return type of `IF()` depends on both arguments, not just the first one.
- `AUTO_INCREMENT` columns should not be used to store negative numbers. The reason for this is that negative numbers caused problems when wrapping from #1 to 0. You should not store 0 in `AUTO_INCREMENT` columns, either; `CHECK TABLE` complains about 0 values because they may change if you dump and restore the table. `AUTO_INCREMENT` for `MyISAM` tables is handled at a lower level and is much faster than before. In addition, for `MyISAM` tables, old numbers are not re-

used, even if you delete rows from the table.

- CASE, DELAYED, ELSE, END, FULLTEXT, INNER, RIGHT, THEN, and WHEN are reserved words.
- FLOAT(p) is a true floating-point type and not a value with a fixed number of decimals.
- When declaring columns using a DECIMAL(length,dec) type, the length argument no longer includes a place for the sign or the decimal point.
- A TIME string must be of one of the following formats: [[[DAYS] [H]H:]MM:]SS[.fraction] or [[[[[H]H]H]H]MM]SS[.fraction].
- LIKE compares strings using the same character comparison rules as for the = operator. If you require the old behavior, you can compile MySQL with the CXXFLAGS=-DLIKE_CMP_TOUPPER flag.
- REGEXP is case insensitive if neither of the strings is a binary string.
- When you check or repair MyISAM (.MYI) tables, you should use the CHECK TABLE statement or the **myisamchk** command. For ISAM (.ISM) tables, use the **isamchk** command.
- Check all your calls to DATE_FORMAT() to make sure that there is a '%' before each format character.
- In MySQL 3.22, the output of SELECT DISTINCT ... was almost always sorted. In MySQL 3.23, you must use GROUP BY or ORDER BY to obtain sorted output.
- SUM() returns NULL instead of 0 if there are no matching rows. This is required by standard SQL.
- An AND or OR with NULL values returns NULL instead of 0. This mostly affects queries that use NOT on an AND/OR expression as NOT NULL = NULL.
- LPAD() and RPAD() shorten the result string if it's longer than the length argument.

C API Changes:

- mysql_fetch_fields_direct() is a function instead of a macro. It returns a pointer to a MYSQL_FIELD instead of a MYSQL_FIELD.
- mysql_num_fields() cannot be used on a MYSQL* object (it's a function that takes a MYSQL_RES* value as an argument). With a MYSQL* object, you should use mysql_field_count() instead.

2.10.5. Upgrading from Version 3.21 to 3.22

Nothing that affects compatibility has changed between versions 3.21 and 3.22. The only pitfall is that new tables that are created with DATE type columns use the new way to store the date. You can't access these new columns from an old version of **mysqld**.

When upgrading to MySQL 3.23 from an earlier version, note the following changes:

- After installing MySQL Version 3.22, you should start the new server and then run the **mysql_fix_privilege_tables** script. This adds the new privileges that you need to use the GRANT command. If you forget this, you get Access denied when you try to use ALTER TABLE, CREATE INDEX, or DROP INDEX. The procedure for updating the grant tables is described in

Section 2.10.7, “Upgrading the Grant Tables”.

- The C API interface to `mysql_real_connect()` has changed. If you have an old client program that calls this function, you must pass a `0` for the new `db` argument (or recode the client to send the `db` element for faster connections). You must also call `mysql_init()` before calling `mysql_real_connect()`. This change was done to allow the new `mysql_options()` function to save options in the `MYSQL` handler structure.
- The **mysqld** variable `key_buffer` has been renamed to `key_buffer_size`, but you can still use the old name in your startup files.

2.10.6. Upgrading from Version 3.20 to 3.21

If you are running a version older than Version 3.20.28 and want to switch to Version 3.21, you need to do the following:

You can start the **mysqld** Version 3.21 server with the `--old-protocol` option to use it with clients from a Version 3.20 distribution. In this case, the server uses the old pre-3.21 `password()` checking rather than the new method. Also, the new client function `mysql_errno()` does not return any server error, only `CR_UNKNOWN_ERROR`. The function does work for client errors.

If you are *not* using the `--old-protocol` option to **mysqld**, you need to make the following changes:

- All client code must be recompiled. If you are using ODBC, you must get the MyODBC 2.x driver.
- The `scripts/add_long_password` script must be run to convert the `Password` field in the `mysql.user` table to `CHAR(16)`.
- All passwords must be reassigned in the `mysql.user` table to get 62-bit rather than 31-bit passwords.
- The table format hasn't changed, so you don't have to convert any tables.

MySQL 3.20.28 and above can handle the new `user` table format without affecting clients. If you have a MySQL version earlier than 3.20.28, passwords no longer work with it if you convert the `user` table. So to be safe, you should first upgrade to at least Version 3.20.28 and then upgrade to Version 3.21.

The new client code works with a 3.20.x **mysqld** server, so if you experience problems with 3.21.x, you can use the old 3.20.x server without having to recompile the clients again.

If you are not using the `--old-protocol` option to **mysqld**, old clients are unable to connect and should issue the following error message:

```
ERROR: Protocol mismatch. Server Version = 10 Client Version = 9
```

The Perl DBI interface also supports the old `mysqlperl` interface. The only change you have to make if you use `mysqlperl` is to change the arguments to the `connect()` function. The new arguments are: `host`, `database`, `user`, and `password` (note that the `user` and `password` arguments have changed places).

The following changes may affect queries in old applications:

- `HAVING` must be specified before any `ORDER BY` clause.

- The parameters to `LOCATE()` have been swapped.
- There are some new reserved words. The most noticeable are `DATE`, `TIME`, and `TIMESTAMP`.

2.10.7. Upgrading the Grant Tables

Some releases introduce changes to the structure of the grant tables (the tables in the `mysql` database) to add new privileges or features. To make sure that your grant tables are current when you update to a new version of MySQL, you should update your grant tables as well.

On Unix or Unix-like systems, update the grant tables by running the **mysql_fix_privilege_tables** script:

```
shell> mysql_fix_privilege_tables
```

You must run this script while the server is running. It attempts to connect to the server running on the local host as `root`. If your `root` account requires a password, indicate the password on the command line. For MySQL 4.1 and up, specify the password like this:

```
shell> mysql_fix_privilege_tables --password=root_password
```

Prior to MySQL 4.1, specify the password like this:

```
shell> mysql_fix_privilege_tables root_password
```

The **mysql_fix_privilege_tables** script performs any actions necessary to convert your grant tables to the current format. You might see some `Duplicate column name` warnings as it runs; you can ignore them.

After running the script, stop the server and restart it.

On Windows systems, there isn't an easy way to update the grant tables until MySQL 4.0.15. From version 4.0.15 on, MySQL distributions include a `mysql_fix_privilege_tables.sql` SQL script that you can run using the **mysql** client. If your MySQL installation is located at `C:\mysql`, the commands look like this:

```
C:\> C:\mysql\bin\mysql -u root -p mysql
mysql> SOURCE C:\mysql\scripts\mysql_fix_privilege_tables.sql
```

If your installation is located in some other directory, adjust the pathnames appropriately.

The **mysql** command will prompt you for the `root` password; enter it when prompted.

As with the Unix procedure, you might see some `Duplicate column name` warnings as **mysql** processes the statements in the `mysql_fix_privilege_tables.sql` script; you can ignore them.

After running the script, stop the server and restart it.

If you are upgrading to MySQL 5.0.1 or later, the grant table upgrade procedure just described adds view-related columns for the `CREATE VIEW` and `SHOW VIEW` privileges. These privileges exist at the global and database levels. Their initial values are assigned as follows:

- In MySQL 5.0.2 or later, **mysql_fix_privilege_tables** copies the `Create_priv` value in the `user`

table to the `Create_view_priv` and `Show_view_priv` columns.

- In 5.0.1, the view-related privileges are not enabled for any accounts, so you cannot immediately use `GRANT` to give them to accounts that should have them. To deal with this, first connect to the server as `root` and issue the following statements to give the privileges to the `root` accounts manually with `UPDATE`:

  ```
  mysql> UPDATE mysql.user SET Show_view_priv = 'Y', Create_view_priv = 'Y'
      -> WHERE User = 'root';
  mysql> FLUSH PRIVILEGES;
  ```

  After this, `root` can use `GRANT` to give the view privileges to other accounts. Note: You should issue the statements just shown, `GRANT ALL` does not work at the global and database levels, because `GRANT ALL` requires that you actually possess all privileges.

2.10.8. Copying MySQL Databases to Another Machine

If you are using MySQL 3.23 or later, you can copy the `.frm`, `.MYI`, and `.MYD` files for `MyISAM` tables between different architectures that support the same floating-point format. (MySQL takes care of any byte-swapping issues.) See Section 14.1, “The `MyISAM` Storage Engine”.

The MySQL `ISAM` data and index files (`.ISD` and `*.ISM`, respectively) are architecture dependent and in some cases operating system dependent. If you want to move your applications to another machine that has a different architecture or operating system than your current machine, you should not try to move a database by simply copying the files to the other machine. Use **mysqldump** instead.

By default, **mysqldump** creates a file containing SQL statements. You can then transfer the file to the other machine and feed it as input to the **mysql** client.

Try **mysqldump --help** to see what options are available. If you are moving the data to a newer version of MySQL, you should use **mysqldump --opt** to take advantage of any optimizations that result in a dump file that is smaller and can be processed faster.

The easiest (although not the fastest) way to move a database between two machines is to run the following commands on the machine on which the database is located:

```
shell> mysqladmin -h 'other_hostname' create db_name
shell> mysqldump --opt db_name | mysql -h 'other_hostname' db_name
```

If you want to copy a database from a remote machine over a slow network, you can use:

```
shell> mysqladmin create db_name
shell> mysqldump -h 'other_hostname' --opt --compress db_name | mysql db_name
```

You can also store the result in a file, then transfer the file to the target machine and load the file into the database there. For example, you can dump a database to a file on the source machine like this:

```
shell> mysqldump --quick db_name | gzip > db_name.contents.gz
```

(The file created in this example is compressed.) Transfer the file containing the database contents to the target machine and run these commands there:

```
shell> mysqladmin create db_name
shell> gunzip < db_name.contents.gz | mysql db_name
```

You can also use **mysqldump** and **mysqlimport** to transfer the database. For big tables, this is much faster than simply using **mysqldump**. In the following commands, `DUMPDIR` represents the full pathname of the directory you use to store the output from **mysqldump**.

First, create the directory for the output files and dump the database:

```
shell> mkdir DUMPDIR
shell> mysqldump --tab=DUMPDIR db_name
```

Then transfer the files in the `DUMPDIR` directory to some corresponding directory on the target machine and load the files into MySQL there:

```
shell> mysqladmin create db_name           # create database
shell> cat DUMPDIR/*.sql | mysql db_name   # create tables in database
shell> mysqlimport db_name DUMPDIR/*.txt   # load data into tables
```

Also, don't forget to copy the `mysql` database because that is where the `user`, `db`, and `host` grant tables are stored. You might have to run commands as the MySQL `root` user on the new machine until you have the `mysql` database in place.

After you import the `mysql` database on the new machine, execute **mysqladmin flush-privileges** so that the server reloads the grant table information.

2.11. Downgrading MySQL

This section describes what you should do if you are downgrading to an older MySQL version in the unlikely case that the previous version worked better than the new one.

If you are downgrading within the same release series (for example, from 4.0.20 to 4.0.19) the general rule is that you just have to install the new binaries on top of the old ones. There is no need to do anything with the databases. As always, however, it's always a good idea to make a backup.

The following items form a checklist of things you should do whenever you perform an downgrade:

- Read the upgrading section for the release series from which you are downgrading to be sure that it doesn't have any features you really need. Section 2.10, “Upgrading MySQL”.
- If there is a downgrading section for that version, please read it, too!

You can always move the MySQL format files and data files between different versions on the same architecture as long as you stay within versions for the same release series of MySQL. The current production release series is 4.1.

If you downgrade from one release series to another, there may be incompatibilities in table storage formats. In this case, you can use **mysqldump** to dump your tables before downgrading. After downgrading, reload the dump file using **mysql** or `mysqlimport` to re-create your tables. See Section 2.10.8, “Copying MySQL Databases to Another Machine” for examples.

The normal symptom of a downward-incompatible table format change when you downgrade is that you can't open tables. In that case, use the following procedure:

1. Stop the older MySQL server that you are trying to downgrade to.
2. Restart the newer MySQL server you are trying to downgrade from.

3. Dump any tables that were inaccessible to the older server by using **mysqldump** to create a dump file.

4. Stop the newer MySQL server and restart the older one.

5. Reload the dump file into the older server. Your tables should be accessible.

2.11.1. Downgrading to 4.0

The table format in 4.1 changed to include more and new character set information. Because of this, you must use `mysqldump` to dump any tables you have created with the newer MySQL server. For example, if all the tables in a particular database need to be dumped to be reverted back to MySQL 4.0 format, use this command:

```
shell> mysqldump --create-options --compatible=mysql40 db_name > dump_file
```

Then stop the newer server, restart the older server, and read in the dump file:

```
shell> mysql db_name < dump_file
```

In the special case that you're downgrading `MyISAM` tables, no special treatment is necessary if all columns in the tables contain only numeric columns or string columns (`CHAR`, `VARCHAR`, `TEXT`, and so forth) that contain only `latin1` data. Your 4.1 tables should be directly usable with a 4.0 server.

If you used the `mysql_fix_privilege_tables` script to upgrade the grant tables, you can either use the preceding method to convert them to back to MySQL 4.0 or do the following in MySQL 4.1 (or above):

```
ALTER TABLE mysql.user CONVERT TO CHARACTER SET latin1 COLLATE latin1_swedish_ci;
ALTER TABLE mysql.db CONVERT TO CHARACTER SET latin1 COLLATE latin1_swedish_ci;
ALTER TABLE mysql.host CONVERT TO CHARACTER SET latin1 COLLATE latin1_swedish_ci;
ALTER TABLE mysql.tables_priv CONVERT TO CHARACTER SET latin1 COLLATE latin1_swedi
ALTER TABLE mysql.columns_priv CONVERT TO CHARACTER SET latin1 COLLATE latin1_swed
ALTER TABLE mysql.func CONVERT TO CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

2.12. Operating System-Specific Notes

2.12.1. Linux Notes

This section discusses issues that have been found to occur on Linux. The first few subsections describe general operating system-related issues, problems that can occur when using binary or source distributions, and post-installation issues. The remaining subsections discuss problems that occur with Linux on specific platforms.

Note that most of these problems occur on older versions of Linux. If you are running a recent version, you may see none of them.

2.12.1.1. Linux Operating System Notes

MySQL needs at least Linux Version 2.0.

Warning: We have seen some strange problems with Linux 2.2.14 and MySQL on SMP systems. We also have reports from some MySQL users that they have encountered serious stability problems using

MySQL with kernel 2.2.14. If you are using this kernel, you should upgrade to 2.2.19 (or newer) or to a 2.4 kernel. If you have a multiple-CPU box, then you should seriously consider using 2.4 because it gives you a significant speed boost. Your system should be more stable.

When using LinuxThreads, you should see a minimum of three **mysqld** processes running. These are in fact threads. There is one thread for the LinuxThreads manager, one thread to handle connections, and one thread to handle alarms and signals.

2.12.1.2. Linux Binary Distribution Notes

The Linux-Intel binary and RPM releases of MySQL are configured for the highest possible speed. We are always trying to use the fastest stable compiler available.

The binary release is linked with `-static`, which means you do not normally need to worry about which version of the system libraries you have. You need not install LinuxThreads, either. A program linked with `-static` is slightly larger than a dynamically linked program, but also slightly faster (3-5%). However, one problem with a statically linked program is that you can't use user-defined functions (UDFs). If you are going to write or use UDFs (this is something for C or C++ programmers only), you must compile MySQL yourself using dynamic linking.

A known issue with binary distributions is that on older Linux systems that use `libc` (such as Red Hat 4.x or Slackware), you get some non-fatal problems with hostname resolution. If your system uses `libc` rather than `glibc2`, you probably will encounter some difficulties with hostname resolution and `getpwnam()`. This happens because `glibc` unfortunately depends on some external libraries to implement hostname resolution and `getpwent()`, even when compiled with `-static`. These problems manifest themselves in two ways:

- You may see the following error message when you run **mysql_install_db**:

  ```
  Sorry, the host 'xxxx' could not be looked up
  ```

  You can deal with this by executing **mysql_install_db --force**, which does not execute the **resolveip** test in **mysql_install_db**. The downside is that you can't use hostnames in the grant tables: Except for `localhost`, you must use IP numbers instead. If you are using an old version of MySQL that doesn't support `--force`, you must manually remove the `resolveip` test in `mysql_install` using an editor.

- You also may see the following error when you try to run **mysqld** with the `--user` option:

  ```
  getpwnam: No such file or directory
  ```

  To work around this, start **mysqld** by using the `su` command rather than by specifying the `--user` option. This causes the system itself to change the user ID of the **mysqld** process so that **mysqld** need not do so.

Another solution, which solves both problems, is to not use a binary distribution. Get a MySQL source distribution (in RPM or `tar.gz` format) and install that instead.

On some Linux 2.2 versions, you may get the error `Resource temporarily unavailable` when clients make a lot of new connections to a **mysqld** server over TCP/IP. The problem is that Linux has a delay between the time that you close a TCP/IP socket and the time that the system actually frees it. There is room for only a finite number of TCP/IP slots, so you encounter the resource-unavailable error if clients attempt too many new TCP/IP connections during a short time. For example, you may see the error when you run the MySQL `test-connect` benchmark over TCP/IP.

We have inquired about this problem a few times on different Linux mailing lists but have never been

able to find a suitable resolution. The only known ``fix'' is for the clients to use persistent connections, or, if you are running the database server and clients on the same machine, to use Unix socket file connections rather than TCP/IP connections.

2.12.1.3. Linux Source Distribution Notes

The following notes regarding `glibc` apply only to the situation when you build MySQL yourself. If you are running Linux on an x86 machine, in most cases it is much better for you to just use our binary. We link our binaries against the best patched version of `glibc` we can come up with and with the best compiler options, in an attempt to make it suitable for a high-load server. For a typical user, even for setups with a lot of concurrent connections or tables exceeding the 2GB limit, our binary is the best choice in most cases. After reading the following text, if you are in doubt about what to do, try our binary first to see whether it meets your needs. If you discover that it is not good enough, then you may want to try your own build. In that case, we would appreciate a note about it so that we can build a better binary next time.

MySQL uses LinuxThreads on Linux. If you are using an old Linux version that doesn't have `glibc2`, you must install LinuxThreads before trying to compile MySQL. You can get LinuxThreads at http://dev.mysql.com/downloads/os-linux.html.

Note that `glibc` versions before and including Version 2.1.1 have a fatal bug in `pthread_mutex_timedwait()` handling, which is used when you issue `INSERT DELAYED` statements. We recommend that you not use `INSERT DELAYED` before upgrading `glibc`.

Note that Linux kernel and the LinuxThread library can by default only have 1,024 threads. If you plan to have more than 1,000 concurrent connections, you need to make some changes to LinuxThreads:

- Increase `PTHREAD_THREADS_MAX` in `sysdeps/unix/sysv/linux/bits/local_lim.h` to 4096 and decrease `STACK_SIZE` in `linuxthreads/internals.h` to 256KB. The paths are relative to the root of `glibc`. (Note that MySQL is not stable with around 600-1000 connections if `STACK_SIZE` is the default of 2MB.)
- Recompile LinuxThreads to produce a new `libpthread.a` library, and relink MySQL against it.

The page http://www.volano.com/linuxnotes.html contains additional information about circumventing thread limits in LinuxThreads.

There is another issue that greatly hurts MySQL performance, especially on SMP systems. The mutex implementation in LinuxThreads in `glibc` 2.1 is very bad for programs with many threads that hold the mutex only for a short time. This produces a paradoxical result: If you link MySQL against an unmodified LinuxThreads, removing processors from an SMP actually improves MySQL performance in many cases. We have made a patch available for `glibc` 2.1.3 to correct this behavior (http://www.mysql.com/Downloads/Linux/linuxthreads-2.1-patch).

With `glibc` 2.2.2, MySQL 3.23.36 uses the adaptive mutex, which is much better than even the patched one in `glibc` 2.1.3. Be warned, however, that under some conditions, the current mutex code in `glibc` 2.2.2 overspins, which hurts MySQL performance. The likelihood that this condition occurs can be reduced by renicing the **mysqld** process to the highest priority. We have also been able to correct the overspin behavior with a patch, available at http://www.mysql.com/Downloads/Linux/linuxthreads-2.2.2.patch. It combines the correction of overspin, maximum number of threads, and stack spacing all in one. You need to apply it in the `linuxthreads` directory with `patch -p0 </tmp/linuxthreads-2.2.2.patch`. We hope it is included in some form in future releases of `glibc` 2.2. In any case, if you link against `glibc` 2.2.2, you still need to correct `STACK_SIZE` and `PTHREAD_THREADS_MAX`. We hope that the defaults is corrected to some more acceptable values for high-load MySQL setup in the future, so that the commands needed to produce your own build can be reduced to **./configure; make; make install**.

We recommend that you use these patches to build a special static version of `libpthread.a` and use it only for statically linking against MySQL. We know that the patches are safe for MySQL and significantly improve its performance, but we cannot say anything about other applications. If you link other applications that require LinuxThreads against the patched static version of the library, or build a patched shared version and install it on your system, you do so at your own risk.

If you experience any strange problems during the installation of MySQL, or with some common utilities hanging, it is very likely that they are either library or compiler related. If this is the case, using our binary resolves them.

If you link your own MySQL client programs, you may see the following error at runtime:

```
ld.so.1: fatal: libmysqlclient.so.#:
open failed: No such file or directory
```

This problem can be avoided by one of the following methods:

- Link clients with the `-Wl,r/full/path/to/libmysqlclient.so` flag rather than with `-Lpath`).
- Copy `libmysqclient.so` to `/usr/lib`.
- Add the pathname of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable before running your client.

If you are using the Fujitsu compiler (`fcc/FCC`), you may have some problems compiling MySQL because the Linux header files are very **gcc** oriented. The following **configure** line should work with **fcc/FCC**:

```
CC=fcc CFLAGS="-O -K fast -K lib -K omitfp -Kpreex -D_GNU_SOURCE \
    -DCONST=const -DNO_STRTOLL_PROTO" \
CXX=FCC CXXFLAGS="-O -K fast -K lib \
    -K omitfp -K preex --no_exceptions --no_rtti -D_GNU_SOURCE \
    -DCONST=const -Dalloca=__builtin_alloca -DNO_STRTOLL_PROTO \
    '-D_EXTERN_INLINE=static __inline'" \
./configure \
    --prefix=/usr/local/mysql --enable-assembler \
    --with-mysqld-ldflags=-all-static --disable-shared \
    --with-low-memory
```

2.12.1.4. Linux Post-Installation Notes

mysql.server can be found in the `support-files` directory under the MySQL installation directory or in a MySQL source tree. You can install it as `/etc/init.d/mysql` for automatic MySQL startup and shutdown. See Section 2.9.2.2, “Starting and Stopping MySQL Automatically”.

If MySQL can't open enough files or connections, it may be that you haven't configured Linux to handle enough files.

In Linux 2.2 and onward, you can check the number of allocated file handles as follows:

```
shell> cat /proc/sys/fs/file-max
shell> cat /proc/sys/fs/dquot-max
shell> cat /proc/sys/fs/super-max
```

If you have more than 16MB of memory, you should add something like the following to your init scripts (for example, `/etc/init.d/boot.local` on SuSE Linux):

```
echo 65536 > /proc/sys/fs/file-max
echo 8192 > /proc/sys/fs/dquot-max
echo 1024 > /proc/sys/fs/super-max
```

You can also run the `echo` commands from the command line as `root`, but these settings are lost the next time your computer restarts.

Alternatively, you can set these parameters on startup by using the `sysctl` tool, which is used by many Linux distributions (SuSE has added it as well, beginning with SuSE Linux 8.0). Just put the following values into a file named `/etc/sysctl.conf`:

```
# Increase some values for MySQL
fs.file-max = 65536
fs.dquot-max = 8192
fs.super-max = 1024
```

You should also add the following to `/etc/my.cnf`:

```
[mysqld_safe]
open-files-limit=8192
```

This should allow the server a limit of 8,192 for the combined number of connections and open files.

The `STACK_SIZE` constant in LinuxThreads controls the spacing of thread stacks in the address space. It needs to be large enough so that there is plenty of room for each individual thread stack, but small enough to keep the stack of some threads from running into the global **mysqld** data. Unfortunately, as we have experimentally discovered, the Linux implementation of `mmap()` successfully unmaps a mapped region if you ask it to map out an address currently in use, zeroing out the data on the entire page instead of returning an error. So, the safety of **mysqld** or any other threaded application depends on ``gentlemanly'' behavior of the code that creates threads. The user must take measures to make sure that the number of running threads at any time is sufficiently low for thread stacks to stay away from the global heap. With **mysqld**, you should enforce this behavior by setting a reasonable value for the `max_connections` variable.

If you build MySQL yourself, you can patch LinuxThreads for better stack use. See Section 2.12.1.3, “Linux Source Distribution Notes”. If you do not want to patch LinuxThreads, you should set `max_connections` to a value no higher than 500. It should be even less if you have a large key buffer, large heap tables, or some other things that make **mysqld** allocate a lot of memory, or if you are running a 2.2 kernel with a 2GB patch. If you are using our binary or RPM version 3.23.25 or later, you can safely set `max_connections` at 1500, assuming no large key buffer or heap tables with lots of data. The more you reduce `STACK_SIZE` in LinuxThreads the more threads you can safely create. We recommend values between 128KB and 256KB.

If you use a lot of concurrent connections, you may suffer from a ``feature'' in the 2.2 kernel that attempts to prevent fork bomb attacks by penalizing a process for forking or cloning a child. This causes MySQL not to scale well as you increase the number of concurrent clients. On single-CPU systems, we have seen this manifested as very slow thread creation: It may take a long time to connect to MySQL (as long as one minute), and it may take just as long to shut it down. On multiple-CPU systems, we have observed a gradual drop in query speed as the number of clients increases. In the process of trying to find a solution, we have received a kernel patch from one of our users who claimed it made a lot of difference for his site. The patch is available at http://www.mysql.com/Downloads/Patches/linux-fork.patch. We have done rather extensive testing of this patch on both development and production systems. It has significantly improved MySQL performance without causing any problems and we recommend it to our users who still run high-load servers on 2.2 kernels.

This issue has been fixed in the 2.4 kernel, so if you are not satisfied with the current performance of your system, rather than patching your 2.2 kernel, it might be easier to upgrade to 2.4. On SMP systems, upgrading also gives you a nice SMP boost in addition to fixing the fairness bug.

We have tested MySQL on the 2.4 kernel on a two-CPU machine and found MySQL scales *much* better. There was virtually no slowdown on query throughput all the way up to 1,000 clients, and the MySQL scaling factor (computed as the ratio of maximum throughput to the throughput for one client) was 180%. We have observed similar results on a four-CPU system: Virtually no slowdown as the number of clients was increased up to 1,000, and a 300% scaling factor. Based on these results, for a high-load SMP server using a 2.2 kernel, we definitely recommend upgrading to the 2.4 kernel at this point.

We have discovered that it is essential to run the **mysqld** process with the highest possible priority on the 2.4 kernel to achieve maximum performance. This can be done by adding a `renice -20 $$` command to **mysqld_safe**. In our testing on a four-CPU machine, increasing the priority resulted in a 60% throughput increase with 400 clients.

We are currently also trying to collect more information on how well MySQL performs with a 2.4 kernel on four-way and eight-way systems. If you have access such a system and have done some benchmarks, please send an email message to `<benchmarks@mysql.com>` with the results. We will review them for inclusion in the manual.

If you see a dead **mysqld** server process with **ps**, this usually means that you have found a bug in MySQL or you have a corrupted table. See Section A.4.2, “What to Do If MySQL Keeps Crashing”.

To get a core dump on Linux if **mysqld** dies with a `SIGSEGV` signal, you can start **mysqld** with the `--core-file` option. Note that you also probably need to raise the core file size by adding **ulimit -c 1000000** to **mysqld_safe** or starting **mysqld_safe** with `--core-file-size=1000000`. See Section 5.1.3, “The **mysqld_safe** Server Startup Script”.

2.12.1.5. Linux x86 Notes

MySQL requires `libc` Version 5.4.12 or newer. It's known to work with `libc` 5.4.46. `glibc` Version 2.0.6 and later should also work. There have been some problems with the `glibc` RPMs from Red Hat, so if you have problems, check whether there are any updates. The `glibc` 2.0.7-19 and 2.0.7-29 RPMs are known to work.

If you are using Red Hat 8.0 or a new `glibc` 2.2.x library, you may see **mysqld** die in `gethostbyaddr()`. This happens because the new `glibc` library requires a stack size greater than 128KB for this call. To fix the problem, start **mysqld** with the `--thread-stack=192K` option. (Use `-O thread_stack=192K` before MySQL 4.) This stack size is the default on MySQL 4.0.10 and above, so you should not see the problem.

If you are using **gcc** 3.0 and above to compile MySQL, you must install the `libstdc++v3` library before compiling MySQL; if you don't do this, you get an error about a missing `__cxa_pure_virtual` symbol during linking.

On some older Linux distributions, **configure** may produce an error like this:

```
Syntax error in sched.h. Change _P to __P in the
/usr/include/sched.h file.
See the Installation chapter in the Reference Manual.
```

Just do what the error message says. Add an extra underscore to the `_P` macro name that has only one underscore, then try again.

You may get some warnings when compiling. Those shown here can be ignored:

```
mysqld.cc -o objs-thread/mysqld.o
```

```
mysqld.cc: In function `void init_signals()':
mysqld.cc:315: warning: assignment of negative value `-1' to
`long unsigned int'
mysqld.cc: In function `void * signal_hand(void *)':
mysqld.cc:346: warning: assignment of negative value `-1' to
`long unsigned int'
```

If **mysqld** always dumps core when it starts, the problem may be that you have an old `/lib/libc.a`. Try renaming it, then remove `sql/mysqld` and do a new **make install** and try again. This problem has been reported on some Slackware installations.

If you get the following error when linking **mysqld**, it means that your `libg++.a` is not installed correctly:

```
/usr/lib/libc.a(putc.o): In function `_IO_putc':
putc.o(.text+0x0): multiple definition of `_IO_putc'
```

You can avoid using `libg++.a` by running **configure** like this:

```
shell> CXX=gcc ./configure
```

If **mysqld** crashes immediately and you are running Red Hat Version 5.0 with a version of `glibc` older than 2.0.7-5, you should make sure that you have installed all `glibc` patches. There is a lot of information about this in the MySQL mail archives, available online at http://lists.mysql.com/.

2.12.1.6. Linux SPARC Notes

In some implementations, `readdir_r()` is broken. The symptom is that the `SHOW DATABASES` statement always returns an empty set. This can be fixed by removing `HAVE_READDIR_R` from `config.h` after configuring and before compiling.

2.12.1.7. Linux Alpha Notes

MySQL 3.23.12 is the first MySQL version that is tested on Linux-Alpha. If you plan to use MySQL on Linux-Alpha, you should ensure that you have this version or newer.

We have tested MySQL on Alpha with our benchmarks and test suite, and it appears to work nicely.

We currently build the MySQL binary packages on SuSE Linux 7.0 for AXP, kernel 2.4.4-SMP, Compaq C compiler (V6.2-505) and Compaq C++ compiler (V6.3-006) on a Compaq DS20 machine with an Alpha EV6 processor.

You can find the preceding compilers at http://www.support.compaq.com/alpha-tools/. By using these compilers rather than **gcc**, we get about 9-14% better MySQL performance.

Note that until MySQL version 3.23.52 and 4.0.2, we optimized the binary for the current CPU only (by using the `-fast` compile option). This means that for older versions, you can use our Alpha binaries only if you have an Alpha EV6 processor.

For all following releases, we added the `-arch generic` flag to our compile options, which makes sure that the binary runs on all Alpha processors. We also compile statically to avoid library problems. The **configure** command looks like this:

```
CC=ccc CFLAGS="-fast -arch generic" CXX=cxx \
CXXFLAGS="-fast -arch generic -noexceptions -nortti" \
./configure --prefix=/usr/local/mysql --disable-shared \
```

```
    --with-extra-charsets=complex --enable-thread-safe-client \
    --with-mysqld-ldflags=-non_shared --with-client-ldflags=-non_shared
```

If you want to use **egcs**, the following **configure** line worked for us:

```
CFLAGS="-O3 -fomit-frame-pointer" CXX=gcc \
CXXFLAGS="-O3 -fomit-frame-pointer -felide-constructors \
    -fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql --disable-shared
```

Some known problems when running MySQL on Linux-Alpha:

- Debugging threaded applications like MySQL does not work with `gdb 4.18`. You should use **gdb** 5.1 instead.
- If you try linking **mysqld** statically when using **gcc**, the resulting image dumps core at startup time. In other words, *do not* use `--with-mysqld-ldflags=-all-static` with **gcc**.

2.12.1.8. Linux PowerPC Notes

MySQL should work on MkLinux with the newest `glibc` package (tested with `glibc` 2.0.7).

2.12.1.9. Linux MIPS Notes

To get MySQL to work on Qube2 (Linux Mips), you need the newest `glibc` libraries. `glibc-2.0.7-29C2` is known to work. You must also use the **egcs** C++ compiler (**egcs** 1.0.2-9, **gcc** 2.95.2 or newer).

2.12.1.10. Linux IA-64 Notes

To get MySQL to compile on Linux IA-64, we use the following **configure** command for building with **gcc** 2.96:

```
CC=gcc \
CFLAGS="-O3 -fno-omit-frame-pointer" \
CXX=gcc \
CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors \
    -fno-exceptions -fno-rtti" \
    ./configure --prefix=/usr/local/mysql \
    "--with-comment=Official MySQL binary" \
    --with-extra-charsets=complex
```

On IA-64, the MySQL client binaries use shared libraries. This means that if you install our binary distribution at a location other than `/usr/local/mysql`, you need to add the path of the directory where you have `libmysqlclient.so` installed either to the `/etc/ld.so.conf` file or to the value of your `LD_LIBRARY_PATH` environment variable.

See Section A.3.1, “Problems Linking to the MySQL Client Library”.

2.12.2. Mac OS X Notes

On Mac OS X, **tar** cannot handle long filenames. If you need to unpack a `.tar.gz` distribution, use **gnutar** instead.

2.12.2.1. Mac OS X 10.x (Darwin)

MySQL should work without any problems on Mac OS X 10.x (Darwin).

Our binary for Mac OS X is compiled on Darwin 6.3 with the following **configure** line:

```
CC=gcc CFLAGS="-O3 -fno-omit-frame-pointer" CXX=gcc \
CXXFLAGS="-O3 -fno-omit-frame-pointer -felide-constructors \
    -fno-exceptions -fno-rtti" \
    ./configure --prefix=/usr/local/mysql \
    --with-extra-charsets=complex --enable-thread-safe-client \
    --enable-local-infile --disable-shared
```

See Section 2.5, “Installing MySQL on Mac OS X”.

2.12.2.2. Mac OS X Server 1.2 (Rhapsody)

For current versions of Mac OS X Server, no operating system changes are necessary before compiling MySQL. Compiling for the Server platform is the same as for the client version of Mac OS X. (However, note that MySQL comes preinstalled on Mac OS X Server, so you need not build it yourself.)

For older versions (Mac OS X Server 1.2, a.k.a. Rhapsody), you must first install a pthread package before trying to configure MySQL.

See Section 2.5, “Installing MySQL on Mac OS X”.

2.12.3. Solaris Notes

On Solaris, you may run into trouble even before you get the MySQL distribution unpacked! Solaris **tar** can't handle long filenames, so you may see an error like this when you unpack MySQL:

```
x mysql-3.22.12-beta/bench/Results/ATIS-mysql_odbc-NT_4.0-cmp-db2,
informix,ms-sql,mysql,oracle,solid,sybase, 0 bytes, 0 tape blocks
tar: directory checksum error
```

In this case, you must use GNU **tar** (**gtar**) to unpack the distribution. You can find a precompiled copy for Solaris at http://dev.mysql.com/downloads/os-solaris.html.

Sun native threads work only on Solaris 2.5 and higher. For Version 2.4 and earlier, MySQL automatically uses MIT-pthreads. See Section 2.8.5, “MIT-pthreads Notes”.

If you get the following error from **configure**, it means that you have something wrong with your compiler installation:

```
checking for restartable system calls... configure: error can not
run test programs while cross compiling
```

In this case, you should upgrade your compiler to a newer version. You may also be able to solve this problem by inserting the following row into the `config.cache` file:

```
ac_cv_sys_restartable_syscalls=${ac_cv_sys_restartable_syscalls='no'}
```

If you are using Solaris on a SPARC, the recommended compiler is **gcc** 2.95.2 or 3.2. You can find this at http://gcc.gnu.org/. Note that **egcs** 1.1.1 and **gcc** 2.8.1 don't work reliably on SPARC!

The recommended **configure** line when using **gcc** 2.95.2 is:

```
CC=gcc CFLAGS="-O3" \
CXX=gcc CXXFLAGS="-O3 -felide-constructors -fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql --with-low-memory \
    --enable-assembler
```

If you have an UltraSPARC system, you can get 4% better performance by adding `-mcpu=v8 -Wa,-xarch=v8plusa` to the `CFLAGS` and `CXXFLAGS` environment variables.

If you have Sun's Forte 5.0 (or newer) compiler, you can run **configure** like this:

```
CC=cc CFLAGS="-Xa -fast -native -xstrconst -mt" \
CXX=CC CXXFLAGS="-noex -mt" \
./configure --prefix=/usr/local/mysql --enable-assembler
```

To create a 64-bit binary with Sun's Forte compiler, use the following configuration options:

```
CC=cc CFLAGS="-Xa -fast -native -xstrconst -mt -xarch=v9" \
CXX=CC CXXFLAGS="-noex -mt -xarch=v9" ASFLAGS="-xarch=v9" \
./configure --prefix=/usr/local/mysql --enable-assembler
```

To create a 64-bit Solaris binary using **gcc**, add `-m64` to `CFLAGS` and `CXXFLAGS` and remove `--enable-assembler` from the **configure** line. This works only with MySQL 4.0 and up; MySQL 3.23 does not include the required modifications to support this.

In the MySQL benchmarks, we got a 4% speedup on an UltraSPARC when using Forte 5.0 in 32-bit mode compared to using **gcc** 3.2 with the `-mcpu` flag.

If you create a 64-bit **mysqld** binary, it is 4% slower than the 32-bit binary, but can handle more threads and memory.

If you get a problem with `fdatasync` or `sched_yield`, you can fix this by adding `LIBS=-lrt` to the **configure** line

For compilers older than WorkShop 5.3, you might have to edit the **configure** script. Change this line:

```
#if !defined(__STDC__) || __STDC__ != 1
```

To this:

```
#if !defined(__STDC__)
```

If you turn on `__STDC__` with the `-Xc` option, the Sun compiler can't compile with the Solaris `pthread.h` header file. This is a Sun bug (broken compiler or broken include file).

If **mysqld** issues the following error message when you run it, you have tried to compile MySQL with the Sun compiler without enabling the `-mt` multi-thread option:

```
libc internal error: _rmutex_unlock: rmutex not held
```

Add `-mt` to `CFLAGS` and `CXXFLAGS` and recompile.

If you are using the SFW version of **gcc** (which comes with Solaris 8), you must add `/opt/sfw/lib` to the environment variable `LD_LIBRARY_PATH` before running **configure**.

If you are using the **gcc** available from `sunfreeware.com`, you may have many problems. To avoid this, you should recompile **gcc** and GNU `binutils` on the machine where you are running them.

If you get the following error when compiling MySQL with **gcc**, it means that your **gcc** is not configured for your version of Solaris:

```
shell> gcc -O3 -g -O2 -DDBUG_OFF  -o thr_alarm ...
./thr_alarm.c: In function `signal_hand':
./thr_alarm.c:556: too many arguments to function `sigwait'
```

The proper thing to do in this case is to get the newest version of **gcc** and compile it with your current **gcc** compiler. At least for Solaris 2.5, almost all binary versions of **gcc** have old, unusable include files that break all programs that use threads, and possibly other programs!

Solaris doesn't provide static versions of all system libraries (`libpthreads` and `libdl`), so you can't compile MySQL with `--static`. If you try to do so, you get one of the following errors:

```
ld: fatal: library -ldl: not found
undefined reference to `dlopen'
cannot find -lrt
```

If you link your own MySQL client programs, you may see the following error at runtime:

```
ld.so.1: fatal: libmysqlclient.so.#:
open failed: No such file or directory
```

This problem can be avoided by one of the following methods:

- Link clients with the `-Wl,r/full/path/to/libmysqlclient.so` flag rather than with `-Lpath`).
- Copy `libmysqclient.so` to `/usr/lib`.
- Add the pathname of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable before running your client.

If you have problems with **configure** trying to link with `-lz` when you don't have `zlib` installed, you have two options:

- If you want to be able to use the compressed communication protocol, you need to get and install `zlib` from `ftp.gnu.org`.
- Run **configure** with the `--with-named-z-libs=no` option when building MySQL.

If you are using **gcc** and have problems with loading user-defined functions (UDFs) into MySQL, try adding `-lgcc` to the link line for the UDF.

If you would like MySQL to start automatically, you can copy `support-files/mysql.server` to `/etc/init.d` and create a symbolic link to it named `/etc/rc3.d/S99mysql.server`.

If too many processes try to connect very rapidly to **mysqld**, you should see this error in the MySQL log:

```
Error in accept: Protocol error
```

You might try starting the server with the `--back_log=50` option as a workaround for this. (Use `-O back_log=50` before MySQL 4.)

Solaris doesn't support core files for `setuid()` applications, so you can't get a core file from **mysqld** if you are using the `--user` option.

2.12.3.1. Solaris 2.7/2.8 Notes

Normally, you can use a Solaris 2.6 binary on Solaris 2.7 and 2.8. Most of the Solaris 2.6 issues also apply for Solaris 2.7 and 2.8.

MySQL 3.23.4 and above should be able to detect new versions of Solaris automatically and enable workarounds for the following problems.

Solaris 2.7 / 2.8 has some bugs in the include files. You may see the following error when you use **gcc**:

```
/usr/include/widec.h:42: warning: `getwc' redefined
/usr/include/wchar.h:326: warning: this is the location of the previous
definition
```

If this occurs, you can fix the problem by copying `/usr/include/widec.h` to `.../lib/gcc-lib/os/gcc-version/include` and changing line 41 from this:

```
#if     !defined(lint) && !defined(__lint)
```

To this:

```
#if     !defined(lint) && !defined(__lint) && !defined(getwc)
```

Alternatively, you can edit `/usr/include/widec.h` directly. Either way, after you make the fix, you should remove `config.cache` and run **configure** again.

If you get the following errors when you run **make**, it's because **configure** didn't detect the `curses.h` file (probably because of the error in `/usr/include/widec.h`):

```
In file included from mysql.cc:50:
/usr/include/term.h:1060: syntax error before `,'
/usr/include/term.h:1081: syntax error before `;'
```

The solution to this problem is to do one of the following:

- Configure with `CFLAGS=-DHAVE_CURSES_H CXXFLAGS=-DHAVE_CURSES_H ./configure`.
- Edit `/usr/include/widec.h` as indicated in the preceding discussion and re-run **configure**.
- Remove the `#define HAVE_TERM` line from the `config.h` file and run **make** again.

If your linker can't find `-lz` when linking client programs, the problem is probably that your `libz.so` file is installed in `/usr/local/lib`. You can fix this problem by one of the following methods:

- Add `/usr/local/lib` to `LD_LIBRARY_PATH`.
- Add a link to `libz.so` from `/lib`.
- If you are using Solaris 8, you can install the optional `zlib` from your Solaris 8 CD distribution.
- Run **configure** with the `--with-named-z-libs=no` option when building MySQL.

2.12.3.2. Solaris x86 Notes

On Solaris 8 on x86, **mysqld** dumps core if you remove the debug symbols using `strip`.

If you are using **gcc** or **egcs** on Solaris x86 and you experience problems with core dumps under load, you should use the following **configure** command:

```
CC=gcc CFLAGS="-O3 -fomit-frame-pointer -DHAVE_CURSES_H" \
CXX=gcc \
CXXFLAGS="-O3 -fomit-frame-pointer -felide-constructors \
    -fno-exceptions -fno-rtti -DHAVE_CURSES_H" \
./configure --prefix=/usr/local/mysql
```

This avoids problems with the `libstdc++` library and with C++ exceptions.

If this doesn't help, you should compile a debug version and run it with a trace file or under **gdb**. See Section E.1.3, “Debugging **mysqld** under **gdb**”.

2.12.4. BSD Notes

This section provides information about using MySQL on variants of BSD Unix.

2.12.4.1. FreeBSD Notes

FreeBSD 4.x or newer is recommended for running MySQL, because the thread package is much more integrated. To get a secure and stable system, you should use only FreeBSD kernels that are marked `-RELEASE`.

The easiest (and preferred) way to install MySQL is to use the **mysql-server** and `mysql-client` ports available at http://www.freebsd.org/. Using these ports gives you the following benefits:

- A working MySQL with all optimizations enabled that are known to work on your version of FreeBSD.
- Automatic configuration and build.
- Startup scripts installed in `/usr/local/etc/rc.d`.
- The ability to use `pkg_info -L` to see which files are installed.
- The ability to use `pkg_delete` to remove MySQL if you no longer want it on your machine.

It is recommended you use MIT-pthreads on FreeBSD 2.x, and native threads on Versions 3 and up. It is possible to run with native threads on some late 2.2.x versions, but you may encounter problems shutting down **mysqld**.

Unfortunately, certain function calls on FreeBSD are not yet fully thread-safe. Most notably, this includes the `gethostbyname()` function, which is used by MySQL to convert hostnames into IP addresses. Under certain circumstances, the **mysqld** process suddenly causes 100% CPU load and is unresponsive. If you encounter this problem, try to start MySQL using the `--skip-name-resolve` option.

Alternatively, you can link MySQL on FreeBSD 4.x against the LinuxThreads library, which avoids a few of the problems that the native FreeBSD thread implementation has. For a very good comparison of LinuxThreads versus native threads, see Jeremy Zawodny's article *FreeBSD or Linux for your MySQL Server?* at http://jeremy.zawodny.com/blog/archives/000697.html.

A known problem when using LinuxThreads on FreeBSD is that the `wait_timeout` value is not honored (probably a signal handling problem in FreeBSD/LinuxThreads). This is supposed to be fixed in FreeBSD 5.0. The symptom is that persistent connections can hang for a very long time without getting closed down.

The MySQL build process requires GNU make (**gmake**) to work. If GNU **make** is not available, you must install it first before compiling MySQL.

The recommended way to compile and install MySQL on FreeBSD with **gcc** (2.95.2 and up) is:

```
CC=gcc CFLAGS="-O2 -fno-strength-reduce" \
    CXX=gcc CXXFLAGS="-O2 -fno-rtti -fno-exceptions \
    -felide-constructors -fno-strength-reduce" \
    ./configure --prefix=/usr/local/mysql --enable-assembler
gmake
gmake install
cd /usr/local/mysql
bin/mysql_install_db --user=mysql
bin/mysqld_safe &
```

If you notice that **configure** uses MIT-pthreads, you should read the MIT-pthreads notes. See Section 2.8.5, “MIT-pthreads Notes”.

If you get an error from **make install** that it can't find `/usr/include/pthreads`, **configure** didn't detect that you need MIT-pthreads. To fix this problem, remove `config.cache`, then re-run **configure** with the `--with-mit-threads` option.

Be sure that your name resolver setup is correct. Otherwise, you may experience resolver delays or failures when connecting to **mysqld**. Also make sure that the `localhost` entry in the `/etc/hosts` file is correct. The file should start with a line similar to this:

```
127.0.0.1       localhost localhost.your.domain
```

FreeBSD is known to have a very low default file handle limit. See Section A.2.17, “File Not Found”. Start the server by using the `--open-files-limit` option for **mysqld_safe**, or raise the limits for the **mysqld** user in `/etc/login.conf` and rebuild it with `cap_mkdb /etc/login.conf`. Also be sure that you set the appropriate class for this user in the password file if you are not using the default (use `chpass mysqld-user-name`). See Section 5.1.3, “The **mysqld_safe** Server Startup Script”.

If you have a lot of memory, you should consider rebuilding the kernel to allow MySQL to use more than 512MB of RAM. Take a look at `option MAXDSIZ` in the LINT config file for more information.

If you get problems with the current date in MySQL, setting the `TZ` variable should help. See Appendix F, *Environment Variables*.

2.12.4.2. NetBSD Notes

To compile on NetBSD, you need GNU **make**. Otherwise, the build process fails when **make** tries to run `lint` on C++ files.

2.12.4.3. OpenBSD 2.5 Notes

On OpenBSD Version 2.5, you can compile MySQL with native threads with the following options:

```
CFLAGS=-pthread CXXFLAGS=-pthread ./configure --with-mit-threads=no
```

2.12.4.4. OpenBSD 2.8 Notes

Our users have reported that OpenBSD 2.8 has a threading bug that causes problems with MySQL. The OpenBSD Developers have fixed the problem, but as of January 25, 2001, it's only available in the ``-current'' branch. The symptoms of this threading bug are slow response, high load, high CPU usage, and crashes.

If you get an error like `Error in accept:: Bad file descriptor` or error 9 when trying to open tables or directories, the problem is probably that you have not allocated enough file descriptors for MySQL.

In this case, try starting **mysqld_safe** as `root` with the following options:

```
mysqld_safe --user=mysql --open-files-limit=2048 &
```

2.12.4.5. BSD/OS Version 2.x Notes

If you get the following error when compiling MySQL, your **ulimit** value for virtual memory is too low:

```
item_func.h: In method
`Item_func_ge::Item_func_ge(const Item_func_ge &)':
item_func.h:28: virtual memory exhausted
make[2]: *** [item_func.o] Error 1
```

Try using **ulimit -v 80000** and run **make** again. If this doesn't work and you are using **bash**, try switching to **csh** or **sh**; some BSDI users have reported problems with **bash** and **ulimit**.

If you are using **gcc**, you may also use have to use the `--with-low-memory` flag for **configure** to be able to compile `sql_yacc.cc`.

If you get problems with the current date in MySQL, setting the `TZ` variable should help. See Appendix F, *Environment Variables*.

2.12.4.6. BSD/OS Version 3.x Notes

Upgrade to BSD/OS Version 3.1. If that is not possible, install BSDIpatch M300-038.

Use the following command when configuring MySQL:

```
env CXX=shlicc++ CC=shlicc2 \
./configure \
    --prefix=/usr/local/mysql \
    --localstatedir=/var/mysql \
    --without-perl \
    --with-unix-socket-path=/var/mysql/mysql.sock
```

The following is also known to work:

```
env CC=gcc CXX=gcc CXXFLAGS=-O3 \
./configure \
    --prefix=/usr/local/mysql \
    --with-unix-socket-path=/var/mysql/mysql.sock
```

You can change the directory locations if you wish, or just use the defaults by not specifying any locations.

If you have problems with performance under heavy load, try using the `--skip-thread-priority` option to **mysqld**. This runs all threads with the same priority. On BSDI Version 3.1, this gives better performance, at least until BSDI fixes its thread scheduler.

If you get the error `virtual memory exhausted` while compiling, you should try using **ulimit -v 80000** and running **make** again. If this doesn't work and you are using **bash**, try switching to **csh** or **sh**; some BSDI users have reported problems with **bash** and **ulimit**.

2.12.4.7. BSD/OS Version 4.x Notes

BSDI Version 4.x has some thread-related bugs. If you want to use MySQL on this, you should install all thread-related patches. At least M400-023 should be installed.

On some BSDI Version 4.x systems, you may get problems with shared libraries. The symptom is that you can't execute any client programs, for example, **mysqladmin**. In this case, you need to reconfigure not to use shared libraries with the `--disable-shared` option to configure.

Some customers have had problems on BSDI 4.0.1 that the **mysqld** binary after a while can't open tables. This is because some library/system-related bug causes **mysqld** to change current directory without having asked for that to happen.

The fix is to either upgrade MySQL to at least version 3.23.34 or, after running **configure**, remove the line `#define HAVE_REALPATH` from `config.h` before running **make**.

Note that this means that you can't symbolically link a database directories to another database directory or symbolic link a table to another database on BSDI. (Making a symbolic link to another disk is okay).

2.12.5. Other Unix Notes

2.12.5.1. HP-UX Version 10.20 Notes

There are a couple of small problems when compiling MySQL on HP-UX. We recommend that you use **gcc** instead of the HP-UX native compiler, because **gcc** produces better code.

We recommend using **gcc** 2.95 on HP-UX. Don't use high optimization flags (such as `-O6`) because they may not be safe on HP-UX.

The following **configure** line should work with **gcc** 2.95:

```
CFLAGS="-I/opt/dce/include -fpic" \
CXXFLAGS="-I/opt/dce/include -felide-constructors -fno-exceptions \
-fno-rtti" \
CXX=gcc \
./configure --with-pthread \
    --with-named-thread-libs='-ldce' \
    --prefix=/usr/local/mysql --disable-shared
```

The following **configure** line should work with **gcc** 3.1:

```
CFLAGS="-DHPUX -I/opt/dce/include -O3 -fPIC" CXX=gcc \
CXXFLAGS="-DHPUX -I/opt/dce/include -felide-constructors \
    -fno-exceptions -fno-rtti -O3 -fPIC" \
./configure --prefix=/usr/local/mysql \
    --with-extra-charsets=complex --enable-thread-safe-client \
    --enable-local-infile  --with-pthread \
    --with-named-thread-libs=-ldce --with-lib-ccflags=-fPIC
    --disable-shared
```

2.12.5.2. HP-UX Version 11.x Notes

For HP-UX Version 11.x, we recommend MySQL 3.23.15 or later.

Because of some critical bugs in the standard HP-UX libraries, you should install the following patches before trying to run MySQL on HP-UX 11.0:

```
PHKL_22840 Streams cumulative
PHNE_22397 ARPA cumulative
```

This solves the problem of getting `EWOULDBLOCK` from `recv()` and `EBADF` from `accept()` in threaded applications.

If you are using **gcc** 2.95.1 on an unpatched HP-UX 11.x system, you may get the following error:

```
In file included from /usr/include/unistd.h:11,
                 from ../include/global.h:125,
                 from mysql_priv.h:15,
                 from item.cc:19:
/usr/include/sys/unistd.h:184: declaration of C function ...
/usr/include/sys/pthread.h:440: previous declaration ...
In file included from item.h:306,
                 from mysql_priv.h:158,
                 from item.cc:19:
```

The problem is that HP-UX doesn't define `pthreads_atfork()` consistently. It has conflicting prototypes in `/usr/include/sys/unistd.h`:184 and `/usr/include/sys/pthread.h`:440.

One solution is to copy `/usr/include/sys/unistd.h` into `mysql/include` and edit `unistd.h` and change it to match the definition in `pthread.h`. Look for this line:

```
extern int pthread_atfork(void (*prepare)(), void (*parent)(),
                                          void (*child)());
```

Change it to look like this:

```
extern int pthread_atfork(void (*prepare)(void), void (*parent)(void),
                                          void (*child)(void));
```

After making the change, the following **configure** line should work:

```
CFLAGS="-fomit-frame-pointer -O3 -fpic" CXX=gcc \
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti -O3" \
./configure --prefix=/usr/local/mysql --disable-shared
```

If you are using MySQL 4.0.5 with the HP-UX compiler, you can use the following command (which has been tested with **cc** B.11.11.04):

```
CC=cc CXX=aCC CFLAGS=+DD64 CXXFLAGS=+DD64 ./configure \
    --with-extra-character-set=complex
```

You can ignore any errors of the following type:

```
aCC: warning 901: unknown option: `-3': use +help for online
documentation
```

If you get the following error from **configure**, verify that you don't have the path to the K&R compiler before the path to the HP-UX C and C++ compiler:

```
checking for cc option to accept ANSI C... no
configure: error: MySQL requires an ANSI C compiler (and a C++ compiler).
Try gcc. See the Installation chapter in the Reference Manual.
```

Another reason for not being able to compile is that you didn't define the `+DD64` flags as just described.

Another possibility for HP-UX 11 is to use MySQL binaries for HP-UX 10.20. We have received reports from some users that these binaries work fine on HP-UX 11.00. If you encounter problems, be sure to check your HP-UX patch level.

2.12.5.3. IBM-AIX notes

Automatic detection of `xlC` is missing from Autoconf, so a number of variables need to be set before running **configure**. The following example uses the IBM compiler:

```
export CC="xlc_r -ma -O3 -qstrict -qoptimize=3 -qmaxmem=8192 "
export CXX="xlC_r -ma -O3 -qstrict -qoptimize=3 -qmaxmem=8192"
export CFLAGS="-I /usr/local/include"
export LDFLAGS="-L /usr/local/lib"
export CPPFLAGS=$CFLAGS
export CXXFLAGS=$CFLAGS

./configure --prefix=/usr/local \
                --localstatedir=/var/mysql \
                --sbindir='/usr/local/bin' \
                --libexecdir='/usr/local/bin' \
                --enable-thread-safe-client \
                --enable-large-files
```

The preceding options are used to compile the MySQL distribution that can be found at http://www-frec.bull.com/.

If you change the `-O3` to `-O2` in the preceding **configure** line, you must also remove the `-qstrict` option. This is a limitation in the IBM C compiler.

If you are using **gcc** or **egcs** to compile MySQL, you *must* use the `-fno-exceptions` flag, because the exception handling in **gcc**/**egcs** is not thread-safe! (This is tested with **egcs** 1.1.) There are also some known problems with IBM's assembler that may cause it to generate bad code when used with **gcc**.

We recommend the following **configure** line with **egcs** and **gcc** 2.95 on AIX:

```
CC="gcc -pipe -mcpu=power -Wa,-many" \
CXX="gcc -pipe -mcpu=power -Wa,-many" \
```

```
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti" \
./configure --prefix=/usr/local/mysql --with-low-memory
```

The `-Wa,-many` option is necessary for the compile to be successful. IBM is aware of this problem but is in no hurry to fix it because of the workaround that is available. We don't know if the `-fno-exceptions` is required with **gcc** 2.95, but because MySQL doesn't use exceptions and the option generates faster code, we recommend that you should always use it with **egcs** / **gcc**.

If you get a problem with assembler code, try changing the `-mcpu=xxx` option to match your CPU. Typically `power2`, `power`, or `powerpc` may need to be used. Alternatively, you might need to use `604` or `604e`. We are not positive but suspect that `power` would likely be safe most of the time, even on a power2 machine.

If you don't know what your CPU is, execute a `uname -m` command. It produces a string that looks like `000514676700`, with a format of `xxyyyyyymmss` where `xx` and `ss` are always `00`, `yyyyyy` is a unique system ID and `mm` is the ID of the CPU Planar. A chart of these values can be found at http://www16.boulder.ibm.com/pseries/en_US/cmds/aixcmds5/uname.htm.

This gives you a machine type and a machine model you can use to determine what type of CPU you have.

If you have problems with signals (MySQL dies unexpectedly under high load), you may have found an OS bug with threads and signals. In this case, you can tell MySQL not to use signals by configuring as follows:

```
CFLAGS=-DDONT_USE_THR_ALARM CXX=gcc \
CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti \
-DDONT_USE_THR_ALARM" \
./configure --prefix=/usr/local/mysql --with-debug \
    --with-low-memory
```

This doesn't affect the performance of MySQL, but has the side effect that you can't kill clients that are ``sleeping'' on a connection with **mysqladmin kill** or **mysqladmin shutdown**. Instead, the client dies when it issues its next command.

On some versions of AIX, linking with `libbind.a` makes `getservbyname()` dump core. This is an AIX bug and should be reported to IBM.

For AIX 4.2.1 and **gcc**, you have to make the following changes.

After configuring, edit `config.h` and `include/my_config.h` and change the line that says this:

```
#define HAVE_SNPRINTF 1
```

to this:

```
#undef HAVE_SNPRINTF
```

And finally, in `mysqld.cc`, you need to add a prototype for `initgroups()`.

```
#ifdef _AIX41
extern "C" int initgroups(const char *,int);
#endif
```

If you need to allocate a lot of memory to the **mysqld** process, it's not enough to just use **ulimit -d unlimited**. You may also have to modify **mysqld_safe** to add a line something like this:

```
export LDR_CNTRL='MAXDATA=0x80000000'
```

You can find more information about using a lot of memory at http://publib16.boulder.ibm.com/pseries/en_US/aixprggd/genprogc/lrg_prg_support.htm.

2.12.5.4. SunOS 4 Notes

On SunOS 4, MIT-pthreads is needed to compile MySQL. This in turn means you need GNU **make**.

Some SunOS 4 systems have problems with dynamic libraries and **libtool**. You can use the following **configure** line to avoid this problem:

```
./configure --disable-shared --with-mysqld-ldflags=-all-static
```

When compiling `readline`, you may get warnings about duplicate defines. These can be ignored.

When compiling **mysqld**, there are some `implicit declaration of function` warnings. These can be ignored.

2.12.5.5. Alpha-DEC-UNIX Notes (Tru64)

If you are using **egcs** 1.1.2 on Digital Unix, you should upgrade to **gcc** 2.95.2, because **egcs** on DEC has some serious bugs!

When compiling threaded programs under Digital Unix, the documentation recommends using the `-pthread` option for **cc** and **cxx** and the `-lmach -lexc` libraries (in addition to `-lpthread`). You should run **configure** something like this:

```
CC="cc -pthread" CXX="cxx -pthread -O" \
./configure --with-named-thread-libs="-lpthread -lmach -lexc -lc"
```

When compiling **mysqld**, you may see a couple of warnings like this:

```
mysqld.cc: In function void handle_connections()':
mysqld.cc:626: passing long unsigned int *' as argument 3 of
accept(int,sockadddr *, int *)'
```

You can safely ignore these warnings. They occur because **configure** can detect only errors, not warnings.

If you start the server directly from the command line, you may have problems with it dying when you log out. (When you log out, your outstanding processes receive a `SIGHUP` signal.) If so, try starting the server like this:

```
nohup mysqld [options] &
```

`nohup` causes the command following it to ignore any `SIGHUP` signal sent from the terminal. Alternatively, start the server by running **mysqld_safe**, which invokes **mysqld** using **nohup** for you. See Section 5.1.3, “The **mysqld_safe** Server Startup Script”.

If you get a problem when compiling `mysys/get_opt.c`, just remove the `#define _NO_PROTO` line from the start of that file.

If you are using Compaq's CC compiler, the following **configure** line should work:

```
CC="cc -pthread"
CFLAGS="-O4 -ansi_alias -ansi_args -fast -inline speed all -arch host"
CXX="cxx -pthread"
CXXFLAGS="-O4 -ansi_alias -ansi_args -fast -inline speed all \
    -arch host -noexceptions -nortti"
export CC CFLAGS CXX CXXFLAGS
./configure \
    --prefix=/usr/local/mysql \
    --with-low-memory \
    --enable-large-files \
    --enable-shared=yes \
    --with-named-thread-libs="-lpthread -lmach -lexc -lc"
gnumake
```

If you get a problem with **libtool** when compiling with shared libraries as just shown, when linking **mysql**, you should be able to get around this by issuing these commands:

```
cd mysql
/bin/sh ../libtool --mode=link cxx -pthread  -O3 -DDBUG_OFF \
    -O4 -ansi_alias -ansi_args -fast -inline speed \
    -speculate all \ -arch host  -DUNDEF_HAVE_GETHOSTBYNAME_R \
    -o mysql  mysql.o readline.o sql_string.o completion_hash.o \
    ../readline/libreadline.a -lcurses \
    ../libmysql/.libs/libmysqlclient.so  -lm
cd ..
gnumake
gnumake install
scripts/mysql_install_db
```

2.12.5.6. Alpha-DEC-OSF/1 Notes

If you have problems compiling and have DEC **CC** and **gcc** installed, try running **configure** like this:

```
CC=cc CFLAGS=-O CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql
```

If you get problems with the `c_asm.h` file, you can create and use a 'dummy' `c_asm.h` file with:

```
touch include/c_asm.h
CC=gcc CFLAGS=-I./include \
CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql
```

Note that the following problems with the **ld** program can be fixed by downloading the latest DEC (Compaq) patch kit from: http://ftp.support.compaq.com/public/unix/.

On OSF/1 V4.0D and compiler "DEC C V5.6-071 on Digital Unix V4.0 (Rev. 878)," the compiler had some strange behavior (undefined `asm` symbols). `/bin/ld` also appears to be broken (problems with `_exit undefined` errors occurring while linking **mysqld**). On this system, we have managed to compile MySQL with the following **configure** line, after replacing `/bin/ld` with the version from OSF 4.0C:

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
```

With the Digital compiler "C++ V6.1-029," the following should work:

```
CC=cc -pthread
```

```
CFLAGS=-O4 -ansi_alias -ansi_args -fast -inline speed \
       -speculate all -arch host
CXX=cxx -pthread
CXXFLAGS=-O4 -ansi_alias -ansi_args -fast -inline speed \
         -speculate all -arch host -noexceptions -nortti
export CC CFLAGS CXX CXXFLAGS
./configure --prefix=/usr/mysql/mysql \
            --with-mysqld-ldflags=-all-static --disable-shared \
            --with-named-thread-libs="-lmach -lexc -lc"
```

In some versions of OSF/1, the `alloca()` function is broken. Fix this by removing the line in `config.h` that defines `'HAVE_ALLOCA'`.

The `alloca()` function also may have an incorrect prototype in `/usr/include/alloca.h`. This warning resulting from this can be ignored.

configure uses the following thread libraries automatically: `--with-named-thread-libs="-lpthread -lmach -lexc -lc"`.

When using **gcc**, you can also try running **configure** like this:

```
CFLAGS=-D_PTHREAD_USE_D4 CXX=gcc CXXFLAGS=-O3 ./configure ...
```

If you have problems with signals (MySQL dies unexpectedly under high load), you may have found an OS bug with threads and signals. In this case, you can tell MySQL not to use signals by configuring with:

```
CFLAGS=-DDONT_USE_THR_ALARM \
CXXFLAGS=-DDONT_USE_THR_ALARM \
./configure ...
```

This doesn't affect the performance of MySQL, but has the side effect that you can't kill clients that are ``sleeping'' on a connection with **mysqladmin kill** or **mysqladmin shutdown**. Instead, the client dies when it issues its next command.

With **gcc** 2.95.2, you may encounter the following compile error:

```
sql_acl.cc:1456: Internal compiler error in `scan_region',
at except.c:2566
Please submit a full bug report.
```

To fix this, you should change to the `sql` directory and do a cut-and-paste of the last **gcc** line, but change `-O3` to `-O0` (or add `-O0` immediately after **gcc** if you don't have any `-O` option on your compile line). After this is done, you can just change back to the top-level directory and run **make** again.

2.12.5.7. SGI Irix Notes

If you are using Irix Version 6.5.3 or newer, **mysqld** is able to create threads only if you run it as a user that has `CAP_SCHED_MGT` privileges (such as `root`) or give the **mysqld** server this privilege with the following shell command:

```
chcap "CAP_SCHED_MGT+epi" /opt/mysql/libexec/mysqld
```

You may have to undefine some symbols in `config.h` after running **configure** and before compiling.

In some Irix implementations, the `alloca()` function is broken. If the **mysqld** server dies on some

`SELECT` statements, remove the lines from `config.h` that define `HAVE_ALLOC` and `HAVE_ALLOCA_H`. If **mysqladmin create** doesn't work, remove the line from `config.h` that defines `HAVE_READDIR_R`. You may have to remove the `HAVE_TERM_H` line as well.

SGI recommends that you install all the patches on this page as a set: http://support.sgi.com/surfzone/patches/patchset/6.2_indigo.rps.html

At the very minimum, you should install the latest kernel rollup, the latest `rld` rollup, and the latest `libc` rollup.

You definitely need all the POSIX patches on this page, for pthreads support:

http://support.sgi.com/surfzone/patches/patchset/6.2_posix.rps.html

If you get the something like the following error when compiling `mysql.cc`:

```
"/usr/include/curses.h", line 82: error(1084):
invalid combination of type
```

Type the following in the top-level directory of your MySQL source tree:

```
extra/replace bool curses_bool < /usr/include/curses.h > include/curses.h
make
```

There have also been reports of scheduling problems. If only one thread is running, performance is slow. Avoid this by starting another client. This may lead to a two-to-tenfold increase in execution speed thereafter for the other thread. This is a poorly understood problem with Irix threads; you may have to improvise to find solutions until this can be fixed.

If you are compiling with **gcc**, you can use the following **configure** command:

```
CC=gcc CXX=gcc CXXFLAGS=-O3 \
./configure --prefix=/usr/local/mysql --enable-thread-safe-client \
    --with-named-thread-libs=-lpthread
```

On Irix 6.5.11 with native Irix C and C++ compilers ver. 7.3.1.2, the following is reported to work

```
CC=cc CXX=CC CFLAGS='-O3 -n32 -TARG:platform=IP22 -I/usr/local/include \
-L/usr/local/lib' CXXFLAGS='-O3 -n32 -TARG:platform=IP22 \
-I/usr/local/include -L/usr/local/lib' \
./configure --prefix=/usr/local/mysql --with-innodb --with-berkeley-db \
    --with-libwrap=/usr/local \
    --with-named-curses-libs=/usr/local/lib/libncurses.a
```

2.12.5.8. SCO Notes

The current port is tested only on ``sco3.2v5.0.5," ``sco3.2v5.0.6," and ``sco3.2v5.0.7" systems. There has also been a lot of progress on a port to ``sco 3.2v4.2." Open Server 5.0.8(Legend) has native threads and allows files greater than 2GB. The current maximum file size is 2GB.

We have been able to compile MySQL with the following **configure** command on OpenServer with **gcc** 2.95.3.

```
CC=gcc CXX=gcc ./configure --prefix=/usr/local/mysql \
    --enable-thread-safe-client --with-innodb \
    --with-openssl --with-vio --with-extra-charsets=complex
```

gcc is available at ftp://ftp.sco.com/pub/openserver5/opensrc/gnutools-5.0.7Kj.

This development system requires the OpenServer Execution Environment Supplement oss646B on OpenServer 5.0.6 and oss656B and The OpenSource libraries found in gwxlibs. All OpenSource tools are in the `opensrc` directory. They are available at ftp://ftp.sco.com/pub/openserver5/opensrc/.

We recommend using the latest production release of MySQL.

SCO provides operating system patches at ftp://ftp.sco.com/pub/openserver5 for OpenServer 5.0.[0-6] and ftp://ftp.sco.com/pub/openserverv5/507 for OpenServer 5.0.7.

SCO provides information about security fixes at ftp://ftp.sco.com/pub/security/OpenServer for OpenServer 5.0.x.

The maximum file size on an OpenSever 5.0.x system is 2GB.

The total memory which could be allocated for streams buffers, clists and lock records cannot exceed 60MB on OpenServer 5.0.x.

Streams buffers are allocated in units of 4096 byte pages, clists are 70 bytes each, and lock records are 64 bytes each, so:

```
(NSTRPAGES * 4096) + (NCLIST * 70) + (MAX_FLCKREC * 64) <= 62914560
```

Follow this procedure to configure the Database Services option. If you are unsure whether an application requires this, see the documentation provided with the application.

1. Log in as `root`.
2. Enable the SUDS driver by editing the `/etc/conf/sdevice.d/suds` file. Change the `N` in the second field to a `Y`.
3. Use `mkdev aio` or the Hardware/Kernel Manager to enable support for asynchronous I/O and relink the kernel. To allow users to lock down memory for use with this type of I/O, update the aiomemlock(F) file. This file should be updated to include the names of users that can use AIO and the maximum amounts of memory they can lock down.
4. Many applications use setuid binaries so that you need to specify only a single user. See the documentation provided with the application to see if this is the case for your application.

After you complete this process, reboot the system to create a new kernel incorporating these changes.

By default, the entries in `/etc/conf/cf.d/mtune` are set as follows:

Value	Default	Min	Max
NBUF	0	24	450000
NHBUF	0	32	524288
NMPBUF	0	12	512
MAX_INODE	0	100	64000
MAX_FILE	0	100	64000
CTBUFSIZE	128	0	256
MAX_PROC	0	50	16000
MAX_REGION	0	500	160000
NCLIST	170	120	16640
MAXUP	100	15	16000
NOFILES	110	60	11000
NHINODE	128	64	8192

```
NAUTOUP         10              0               60
NGROUPS         8               0               128
BDFLUSHR        30              1               300
MAX_FLCKREC     0               50              16000
PUTBUFSZ        8000            2000            20000
MAXSLICE        100             25              100
ULIMIT          4194303         2048            4194303
* Streams Parameters
NSTREAM         64              1               32768
NSTRPUSH        9               9               9
NMUXLINK        192             1               4096
STRMSGSZ        16384           4096            524288
STRCTLSZ        1024            1024            1024
STRMAXBLK       524288          4096            524288
NSTRPAGES       500             0               8000
STRSPLITFRAC    80              50              100
NLOG            3               3               3
NUMSP           64              1               256
NUMTIM          16              1               8192
NUMTRW          16              1               8192
* Semaphore Parameters
SEMMAP          10              10              8192
SEMMNI          10              10              8192
SEMMNS          60              60              8192
SEMMNU          30              10              8192
SEMMSL          25              25              150
SEMOPM          10              10              1024
SEMUME          10              10              25
SEMVMX          32767           32767           32767
SEMAEM          16384           16384           16384
* Shared Memory Parameters
SHMMAX          524288          131072          2147483647
SHMMIN          1               1               1
SHMMNI          100             100             2000
FILE            0               100             64000
NMOUNT          0               4               256
NPROC           0               50              16000
NREGION         0               500             160000
```

We recommend setting these values as follows:

`NOFILES` should be 4096 or 2048.

`MAXUP` should be 2048.

To make changes to the kernel, `cd` to `/etc/conf/bin` and use **./idtune** *`name parameter`* to make the changes. For example, to change `SEMMS` to `200`, execute these commands as `root`:

```
# cd /etc/conf/bin
# ./idtune SEMMNS 200
```

We recommend tuning the system, but the proper parameter values to use depend on the number of users accessing the application or database and size the of the database (that is, the used buffer pool). The following affects the kernel parameters defined in `/etc/conf/cf.d/stune`:

`SHMMAX` (recommended setting: 128MB) and `SHMSEG` (recommended setting: 15). These parameters have influence on the MySQL database engine to create user buffer pools.

`NOFILES` and `MAXUP` should be at to at least 2048.

`MAXPROC` should be set to at least 3000/4000 (depends on number of users) or more.

Also is recommended to use following formula to count value for `SEMMSL`, `SEMMNS` and `SEMMNU`:

```
SEMMSL = 13
```

The 13 is what has been found to be the best for both Progress and MySQL.

`SEMMNS` = `SEMMSL` * number of db servers to be run on the system.

Set `SEMMNS` to the value of `SEMMSL` multiplied by the number of db servers (maximum) that you are running on the system at one time.

```
SEMMNU = SEMMNS
```

Set the value of `SEMMNU` to equal the value of `SEMMNS`. You could probably set this to 75% of `SEMMNS`, but this is a conservative estimate.

You need to at least install the "SCO OpenServer Linker and Application Development Libraries" or the OpenServer Development System to use **gcc**. You cannot just use the GCC Dev system without installing one of these.

You should get the FSU Pthreads package and install it first. This can be found at http://moss.csc.ncsu.edu/~mueller/ftp/pub/PART/pthreads.tar.gz. You can also get a precompiled package from ftp://ftp.zenez.com/pub/zenez/prgms/FSU-threads-3.14.tar.gz.

FSU Pthreads can be compiled with SCO Unix 4.2 with tcpip, or using OpenServer 3.0 or Open Desktop 3.0 (OS 3.0 ODT 3.0) with the SCO Development System installed using a good port of GCC 2.5.x. For ODT or OS 3.0, you need a good port of GCC 2.5.x. There are a lot of problems without a good port. The port for this product requires the SCO Unix Development system. Without it, you are missing the libraries and the linker that is needed. You also need `SCO-3.2v4.2-includes.tar.gz`. This file contains the changes to the SCO Development include files that are needed to get MySQL to build. You need to replace the existing system include files with these modified header files. They can be obtained from ftp://ftp.zenez.com/pub/zenez/prgms/SCO-3.2v4.2-includes.tar.gz.

To build FSU Pthreads on your system, all you should need to do is run GNU **make**. The `Makefile` in FSU-threads-3.14.tar.gz is set up to make FSU-threads.

You can run **./configure** in the `threads/src` directory and select the SCO OpenServer option. This command copies `Makefile.SCO5` to `Makefile`. Then run **make**.

To install in the default `/usr/include` directory, log in as `root`, then `cd` to the `thread/src` directory and run **make install**.

Remember that you must use GNU **make** when making MySQL.

Note: If you don't start **mysqld_safe** as `root`, you should get only the default 110 open files per process. **mysqld** writes a note about this in the log file.

With SCO 3.2V4.2, you should use FSU Pthreads version 3.14 or newer. The following **configure** command should work:

```
CFLAGS="-D_XOPEN_XPG4" CXX=gcc CXXFLAGS="-D_XOPEN_XPG4" \
./configure \
    --prefix=/usr/local/mysql \
    --with-named-thread-libs="-lgthreads -lsocket -lgen -lgthreads" \
    --with-named-curses-libs="-lcurses"
```

You may get some problems with some include files. In this case, you can find new SCO-specific in-

clude files at ftp://ftp.zenez.com/pub/zenez/prgms/SCO-3.2v4.2-includes.tar.gz.

You should unpack this file in the `include` directory of your MySQL source tree.

SCO development notes:

- MySQL should automatically detect FSU Pthreads and link **mysqld** with `-lgthreads -lsocket -lgthreads`.
- The SCO development libraries are re-entrant in FSU Pthreads. SCO claims that its library functions are re-entrant, so they must be re-entrant with FSU Pthreads. FSU Pthreads on OpenServer tries to use the SCO scheme to make re-entrant libraries.
- FSU Pthreads (at least the version at ftp::/ftp.zenez.com) comes linked with GNU `malloc`. If you encounter problems with memory usage, make sure that `gmalloc.o` is included in `libgthreads.a` and `libgthreads.so`.
- In FSU Pthreads, the following system calls are pthreads-aware: `read()`, `write()`, `getmsg()`, `connect()`, `accept(), select()`, and `wait()`.
- The CSSA-2001-SCO.35.2 (the patch is listed in custom as erg711905-dscr_remap security patch (version 2.0.0)) breaks FSU threads and makes **mysqld** unstable. You have to remove this one if you want to run **mysqld** on an OpenServer 5.0.6 machine.
- SCO provides operating system patches at ftp://ftp.sco.com/pub/openserver5 for OpenServer 5.0.x.
- SCO provides security fixes and `libsocket.so.2` at ftp://ftp.sco.com/pub/security/OpenServer and ftp://ftp.sco.com/pub/security/sse for OpenServer 5.0.x.
- Pre-OSR506 security fixes. Also, the `telnetd` fix at ftp://stage.caldera.com/pub/security/openserver/ or ftp://stage.caldera.com/pub/security/openserver/CSSA-2001-SCO.10/ as both `libsocket.so.2` and `libresolv.so.1` with instructions for installing on pre-OSR506 systems.

  It's probably a good idea to install these patches before trying to compile/use MySQL.

Beginning with Legend, OpenServer has native threads and no 2GB file size limit.

2.12.5.9. SCO UnixWare Version 7.1.x Notes

We recommend using the latest production release of MySQL. Currently this is MySQL 4.0.x. Should you choose to use an older release of MySQL on UnixWare 7.1.x, you must use a version of MySQL at least as recent as 3.22.13 to get fixes for some portability and OS problems.

We have been able to compile MySQL with the following **configure** command on UnixWare Version 7.1.x:

```
CC="cc" CFLAGS="-I/usr/local/include" \
CXX="CC" CXXFLAGS="-I/usr/local/include" \
./configure --prefix=/usr/local/mysql \
    --enable-thread-safe-client --with-berkeley-db=./bdb \
    --with-innodb --with-openssl --with-extra-charsets=complex
```

If you want to use **gcc**, you must use **gcc** 2.95.3 or newer.

```
CC=gcc CXX=g++ ./configure --prefix=/usr/local/mysql
```

SCO provides operating system patches at ftp://ftp.sco.com/pub/unixware7 for UnixWare 7.1.1, ftp://ftp.sco.com/pub/unixware7/713/ for UnixWare 7.1.3, ftp://ftp.sco.com/pub/unixware7/714/ for UnixWare 7.1.4, and ftp://ftp.sco.com/pub/openunix8 for OpenUNIX 8.0.0.

SCO provides information about security fixes at ftp://ftp.sco.com/pub/security/OpenUNIX for OpenUNIX and ftp://ftp.sco.com/pub/security/UnixWare for UnixWare.

By default, the maximum file size on a UnixWare 7 system is 1GB. Many OS utilities have a limitation of 2GB. The maximum possible file size on UnixWare 7 is 1TB with VXFS.

To enable large file support on UnixWare 7.1.x, run `fsadm`.

```
# fsadm -Fvxfs -o largefiles /
# fsadm /         * Note
# ulimit unlimited
# cd /etc/conf/bin
# ./idtune SFSZLIM 0x7FFFFFFF     ** Note
# ./idtune HFSZLIM 0x7FFFFFFF     ** Note
# ./idbuild -B

* This should report "largefiles".
** 0x7FFFFFFF represents infinity for these values.
```

Reboot the system using `shutdown`.

By default, the entries in `/etc/conf/cf.d/mtune` are set to:

```
Value           Default         Min             Max
-----           -------         ---             ---
SVMMLIM         0x9000000       0x1000000       0x7FFFFFFF
HVMMLIM         0x9000000       0x1000000       0x7FFFFFFF
SSTKLIM         0x1000000       0x2000          0x7FFFFFFF
HSTKLIM         0x1000000       0x2000          0x7FFFFFFF
```

We recommend setting these values as follows:

```
SDATLIM 0x7FFFFFFF
HDATLIM 0x7FFFFFFF
SSTKLIM 0x7FFFFFFF
HSTKLIM 0x7FFFFFFF
SVMMLIM 0x7FFFFFFF
HVMMLIM 0x7FFFFFFF
SFNOLIM 2048
HFNOLIM 2048
```

We recommend tuning the system, but the proper parameter values to use depend on the number of users accessing the application or database and size the of the database (that is, the used buffer pool). The following affects the kernel parameters defined in `/etc/conf/cf.d/stune`:

`SHMMAX` (recommended setting: 128MB) and `SHMSEG` (recommended setting: 15). These parameters have influence on the MySQL database engine to create user buffer pools.

`SFNOLIM` and `HFNOLIM` should be at maximum 2048.

`NPROC` should be set to at least 3000/4000 (depends on number of users).

Also is recommended to use following formula to count value for `SEMMSL`, `SEMMNS`, and `SEMMNU`:

```
SEMMSL = 13
```

13 is what has been found to be the best for both Progress and MySQL.

SEMMNS = SEMMSL * number of db servers to be run on the system.

Set SEMMNS to the value of SEMMSL multiplied by the number of db servers (maximum) that you are running on the system at one time.

SEMMNU = SEMMNS

Set the value of SEMMNU to equal the value of SEMMNS. You could probably set this to 75% of SEMMNS, but this is a conservative estimate.

2.12.6. OS/2 Notes

MySQL uses quite a few open files. Because of this, you should add something like the following to your CONFIG.SYS file:

```
SET EMXOPT=-c -n -h1024
```

If you don't do this, you may encounter the following error:

```
File 'xxxx' not found (Errcode: 24)
```

When using MySQL with OS/2 Warp 3, FixPack 29 or above is required. With OS/2 Warp 4, FixPack 4 or above is required. This is a requirement of the Pthreads library. MySQL must be installed on a partition with a type that supports long filenames, such as HPFS, FAT32, and so on.

The INSTALL.CMD script must be run from OS/2's own CMD.EXE and may not work with replacement shells such as 4OS2.EXE.

The scripts/mysql-install-db script has been renamed. It is called install.cmd and is a REXX script, which sets up the default MySQL security settings and creates the WorkPlace Shell icons for MySQL.

Dynamic module support is compiled in but not fully tested. Dynamic modules should be compiled using the Pthreads runtime library.

```
gcc -Zdll -Zmt -Zcrtdll=pthrdrtl -I../include -I../regex -I.. \
    -o example udf_example.cc -L../lib -lmysqlclient udf_example.def
mv example.dll example.udf
```

Note: Due to limitations in OS/2, UDF module name stems must not exceed eight characters. Modules are stored in the /mysql2/udf directory; the safe-mysqld.cmd script puts this directory in the BEGINLIBPATH environment variable. When using UDF modules, specified extensions are ignored--it is assumed to be .udf. For example, in Unix, the shared module might be named example.so and you would load a function from it like this:

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'example.so';
```

In OS/2, the module would be named example.udf, but you would not specify the module extension:

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'example';
```

2.12.7. BeOS Notes

We have in the past talked with some BeOS developers who have said that MySQL is 80% ported to BeOS, but we haven't heard from them in a while.

2.13. Perl Installation Notes

Perl support for MySQL is provided by means of the `DBI`/`DBD` client interface. The interface requires Perl Version 5.6.0 or later. It *does not work* if you have an older version of Perl.

If you want to use transactions with Perl DBI, you need to have `DBD::mysql` version 1.2216 or newer. Version 2.9003 or newer is recommended.

If you are using the MySQL 4.1 client library, you must use `DBD::mysql` 2.9003 or newer.

As of MySQL 3.22.8, Perl support is no longer included with MySQL distributions. You can obtain the necessary modules from http://search.cpan.org for Unix, or by using the ActiveState `ppm` program on Windows. The following sections describe how to do this.

Perl support for MySQL must be installed if you want to run the MySQL benchmark scripts. See Section 7.1.4, “The MySQL Benchmark Suite”.

2.13.1. Installing Perl on Unix

MySQL Perl support requires that you've installed MySQL client programming support (libraries and header files). Most installation methods install the necessary files. However, if you installed MySQL from RPM files on Linux, be sure that you've installed the developer RPM. The client programs are in the client RPM, but client programming support is in the developer RPM.

If you want to install Perl support, the files you need can be obtained from the CPAN (Comprehensive Perl Archive Network) at http://search.cpan.org.

The easiest way to install Perl modules on Unix is to use the `CPAN` module. For example:

```
shell> perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD::mysql
```

The `DBD::mysql` installation runs a number of tests. These tests require being able to connect to the local MySQL server as the anonymous user with no password. If you have removed anonymous accounts or assigned them passwords, the tests fail. You can use `force install DBD::mysql` to ignore the failed tests.

`DBI` requires the `Data::Dumper` module. It may be installed; if not, you should install it before installing `DBI`.

It is also possible to download the module distributions in the form of compressed **tar** archives and build the modules manually. For example, to unpack and build a DBI distribution, use a procedure such as this:

1. Unpack the distribution into the current directory:

   ```
   shell> gunzip < DBI-VERSION.tar.gz | tar xvf -
   ```

   This command creates a directory named `DBI-`*`VERSION`*.

2. Change location into the top-level directory of the unpacked distribution:

```
shell> cd DBI-VERSION
```

3. Build the distribution and compile everything:

```
shell> perl Makefile.PL
shell> make
shell> make test
shell> make install
```

The **make test** command is important because it verifies that the module is working. Note that when you run that command during the `DBD::mysql` installation to exercise the interface code, the MySQL server must be running or the test fails.

It is a good idea to rebuild and reinstall the `DBD::mysql` distribution whenever you install a new release of MySQL, particularly if you notice symptoms such as that all your `DBI` scripts fail after you upgrade MySQL.

If you don't have access rights to install Perl modules in the system directory or if you want to install local Perl modules, the following reference may be useful: http://servers.digitaldaze.com/extensions/perl/modules.html#modules

Look under the heading ``Installing New Modules that Require Locally Installed Modules.''

2.13.2. Installing ActiveState Perl on Windows

On Windows, you should do the following to install the MySQL `DBD` module with ActiveState Perl:

- Get ActiveState Perl from http://www.activestate.com/Products/ActivePerl/ and install it.
- Open a console window (a ``DOS window'').
- If required, set the `HTTP_proxy` variable. For example, you might try:

  ```
  set HTTP_proxy=my.proxy.com:3128
  ```

- Start the PPM program:

  ```
  C:\> C:\perl\bin\ppm.pl
  ```

- If you have not previously done so, install `DBI`:

  ```
  ppm> install DBI
  ```

- If this succeeds, run the following command:

  ```
  install \
  ftp://ftp.de.uu.net/pub/CPAN/authors/id/JWIED/DBD-mysql-1.2212.x86.ppd
  ```

This procedure should work at least with ActiveState Perl Version 5.6.

If you can't get the procedure to work, you should instead install the MyODBC driver and connect to the MySQL server through ODBC:

```
use DBI;
$dbh= DBI->connect("DBI:ODBC:$dsn",$user,$password) ||
  die "Got error $DBI::errstr when connecting to $dsn\n";
```

2.13.3. Problems Using the Perl DBI/DBD Interface

If Perl reports that it can't find the `../mysql/mysql.so` module, then the problem is probably that Perl can't locate the shared library `libmysqlclient.so`.

You should be able to fix this by one of the following methods:

- Compile the `DBD::mysql` distribution with `perl Makefile.PL -static -config` rather than `perl Makefile.PL`.
- Copy `libmysqlclient.so` to the directory where your other shared libraries are located (probably `/usr/lib` or `/lib`).
- Modify the `-L` options used to compile `DBD::mysql` to reflect the actual location of `libmysqlclient.so`.
- On Linux, you can add the pathname of the directory where `libmysqlclient.so` is located to the `/etc/ld.so.conf` file.
- Add the pathname of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable. Some systems use `LD_LIBRARY_PATH` instead.

Note that you may also need to modify the `-L` options if there are other libraries that the linker fails to find. For example, if the linker cannot find `libc` because it is in `/lib` and the link command specifies `-L/usr/lib`, change the `-L` option to `-L/lib` or add `-L/lib` to the existing link command.

If you get the following errors from `DBD::mysql`, you are probably using **gcc** (or using an old binary compiled with **gcc**):

```
/usr/bin/perl: can't resolve symbol '__moddi3'
/usr/bin/perl: can't resolve symbol '__divdi3'
```

Add `-L/usr/lib/gcc-lib/... -lgcc` to the link command when the `mysql.so` library gets built (check the output from **make** for `mysql.so` when you compile the Perl client). The `-L` option should specify the pathname of the directory where `libgcc.a` is located on your system.

Another cause of this problem may be that Perl and MySQL aren't both compiled with **gcc**. In this case, you can solve the mismatch by compiling both with **gcc**.

You may see the following error from `DBD::mysql` when you run the tests:

```
t/00base............install_driver(mysql) failed:
Can't load '../blib/arch/auto/DBD/mysql/mysql.so' for module DBD::mysql:
../blib/arch/auto/DBD/mysql/mysql.so: undefined symbol:
uncompress at /usr/lib/perl5/5.00503/i586-linux/DynaLoader.pm line 169.
```

This means that you need to include the `-lz` compression library on the link line. That can be done by changing the following line in the file `lib/DBD/mysql/Install.pm`:

```
$sysliblist .= " -lm";
```

Change that line to:

```
$sysliblist .= " -lm -lz";
```

After this, you *must* run **make realclean** and then proceed with the installation from the beginning.

If you want to install DBI on SCO, you have to edit the `Makefile` in DBI-*xxx* and each subdirectory. Note that the following assumes **gcc** 2.95.2 or newer:

```
OLD:                                    NEW:
CC = cc                                 CC = gcc
CCCDLFLAGS = -KPIC -W1,-Bexport         CCCDLFLAGS = -fpic
CCDLFLAGS = -wl,-Bexport                CCDLFLAGS =

LD = ld                                 LD = gcc -G -fpic
LDDLFLAGS = -G -L/usr/local/lib         LDDLFLAGS = -L/usr/local/lib
LDFLAGS = -belf -L/usr/local/lib        LDFLAGS = -L/usr/local/lib

LD = ld                                 LD = gcc -G -fpic
OPTIMISE = -Od                          OPTIMISE = -O1

OLD:
CCCFLAGS = -belf -dy -w0 -U M_XENIX -DPERL_SCO5 -I/usr/local/include

NEW:
CCFLAGS = -U M_XENIX -DPERL_SCO5 -I/usr/local/include
```

These changes are necessary because the Perl dynaloader does not load the `DBI` modules if they were compiled with **icc** or **cc**.

If you want to use the Perl module on a system that doesn't support dynamic linking (such as SCO), you can generate a static version of Perl that includes `DBI` and `DBD::mysql`. The way this works is that you generate a version of Perl with the `DBI` code linked in and install it on top of your current Perl. Then you use that to build a version of Perl that additionally has the `DBD` code linked in, and install that.

On SCO, you must have the following environment variables set:

```
LD_LIBRARY_PATH=/lib:/usr/lib:/usr/local/lib:/usr/progressive/lib
```

Or:

```
LD_LIBRARY_PATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:\
    /usr/progressive/lib:/usr/skunk/lib
LIBPATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:\
    /usr/progressive/lib:/usr/skunk/lib
MANPATH=scohelp:/usr/man:/usr/local1/man:/usr/local/man:\
    /usr/skunk/man:
```

First, create a Perl that includes a statically linked `DBI` module by running these commands in the directory where your `DBI` distribution is located:

```
shell> perl Makefile.PL -static -config
shell> make
shell> make install
shell> make perl
```

Then you must install the new Perl. The output of **make perl** indicates the exact **make** command you need to execute to perform the installation. On SCO, this is **make -f Makefile.aperl inst_perl MAP_TARGET=perl**.

Next, use the just-created Perl to create another Perl that also includes a statically linked `DBD::mysql` by running these commands in the directory where your `DBD::mysql` distribution is located:

```
shell> perl Makefile.PL -static -config
shell> make
shell> make install
shell> make perl
```

Finally, you should install this new Perl. Again, the output of **make perl** indicates the command to use.

Chapter 3. MySQL Tutorial

This chapter provides a tutorial introduction to MySQL by showing how to use the **mysql** client program to create and use a simple database. **mysql** (sometimes referred to as the ``terminal monitor'' or just ``monitor'') is an interactive program that allows you to connect to a MySQL server, run queries, and view the results. **mysql** may also be used in batch mode: you place your queries in a file beforehand, then tell **mysql** to execute the contents of the file. Both ways of using **mysql** are covered here.

To see a list of options provided by **mysql**, invoke it with the `--help` option:

```
shell> mysql --help
```

This chapter assumes that **mysql** is installed on your machine and that a MySQL server is available to which you can connect. If this is not true, contact your MySQL administrator. (If **you** are the administrator, you need to consult other sections of this manual.)

This chapter describes the entire process of setting up and using a database. If you are interested only in accessing an existing database, you may want to skip over the sections that describe how to create the database and the tables it contains.

Because this chapter is tutorial in nature, many details are necessarily omitted. Consult the relevant sections of the manual for more information on the topics covered here.

3.1. Connecting to and Disconnecting from the Server

To connect to the server, you'll usually need to provide a MySQL username when you invoke **mysql** and, most likely, a password. If the server runs on a machine other than the one where you log in, you'll also need to specify a hostname. Contact your administrator to find out what connection parameters you should use to connect (that is, what host, username, and password to use). Once you know the proper parameters, you should be able to connect like this:

```
shell> mysql -h host -u user -p
Enter password: ********
```

`host` and `user` represent the hostname where your MySQL server is running and the username of your MySQL account. Substitute appropriate values for your setup. The `********` represents your password; enter it when **mysql** displays the `Enter password:` prompt.

If that works, you should see some introductory information followed by a `mysql>` prompt:

```
shell> mysql -h host -u user -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25338 to server version: 4.0.14-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

The prompt tells you that **mysql** is ready for you to enter commands.

Some MySQL installations allow users to connect as the anonymous (unnamed) user to the server run-

ning on the local host. If this is the case on your machine, you should be able to connect to that server by invoking **mysql** without any options:

```
shell> mysql
```

After you have connected successfully, you can disconnect any time by typing `QUIT` (or `\q`) at the `mysql>` prompt:

```
mysql> QUIT
Bye
```

On Unix, you can also disconnect by pressing Control-D.

Most examples in the following sections assume that you are connected to the server. They indicate this by the `mysql>` prompt.

3.2. Entering Queries

Make sure that you are connected to the server, as discussed in the previous section. Doing so does not in itself select any database to work with, but that's okay. At this point, it's more important to find out a little about how to issue queries than to jump right in creating tables, loading data into them, and retrieving data from them. This section describes the basic principles of entering commands, using several queries you can try out to familiarize yourself with how **mysql** works.

Here's a simple command that asks the server to tell you its version number and the current date. Type it in as shown here following the `mysql>` prompt and press Enter:

```
mysql> SELECT VERSION(), CURRENT_DATE;
+--------------+--------------+
| VERSION()    | CURRENT_DATE |
+--------------+--------------+
| 3.22.20a-log | 1999-03-19   |
+--------------+--------------+
1 row in set (0.01 sec)
mysql>
```

This query illustrates several things about **mysql**:

- A command normally consists of an SQL statement followed by a semicolon. (There are some exceptions where a semicolon may be omitted. `QUIT`, mentioned earlier, is one of them. We'll get to others later.)
- When you issue a command, **mysql** sends it to the server for execution and displays the results, then prints another `mysql>` prompt to indicate that it is ready for another command.
- **mysql** displays query output in tabular form (rows and columns). The first row contains labels for the columns. The rows following are the query results. Normally, column labels are the names of the columns you fetch from database tables. If you're retrieving the value of an expression rather than a table column (as in the example just shown), **mysql** labels the column using the expression itself.
- **mysql** shows how many rows were returned and how long the query took to execute, which gives you a rough idea of server performance. These values are imprecise because they represent wall clock time (not CPU or machine time), and because they are affected by factors such as server load and network latency. (For brevity, the ``rows in set'' line is not shown in the remaining examples in this chapter.)

Keywords may be entered in any lettercase. The following queries are equivalent:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

Here's another query. It demonstrates that you can use **mysql** as a simple calculator:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+-------------+---------+
| SIN(PI()/4) | (4+1)*5 |
+-------------+---------+
|    0.707107 |      25 |
+-------------+---------+
```

The queries shown thus far have been relatively short, single-line statements. You can even enter multiple statements on a single line. Just end each one with a semicolon:

```
mysql> SELECT VERSION(); SELECT NOW();
+--------------+
| VERSION()    |
+--------------+
| 3.22.20a-log |
+--------------+

+---------------------+
| NOW()               |
+---------------------+
| 1999-03-19 00:15:33 |
+---------------------+
```

A command need not be given all on a single line, so lengthy commands that require several lines are not a problem. **mysql** determines where your statement ends by looking for the terminating semicolon, not by looking for the end of the input line. (In other words, **mysql** accepts free-format input: it collects input lines but does not execute them until it sees the semicolon.)

Here's a simple multiple-line statement:

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
+--------------------+--------------+
| USER()             | CURRENT_DATE |
+--------------------+--------------+
| joesmith@localhost | 1999-03-18   |
+--------------------+--------------+
```

In this example, notice how the prompt changes from `mysql>` to `->` after you enter the first line of a multiple-line query. This is how **mysql** indicates that it hasn't seen a complete statement and is waiting for the rest. The prompt is your friend, because it provides valuable feedback. If you use that feedback, you can always be aware of what **mysql** is waiting for.

If you decide you don't want to execute a command that you are in the process of entering, cancel it by typing `\c`:

```
mysql> SELECT
    -> USER()
```

```
    -> \c
mysql>
```

Here, too, notice the prompt. It switches back to `mysql>` after you type `\c`, providing feedback to indicate that **mysql** is ready for a new command.

The following table shows each of the prompts you may see and summarizes what they mean about the state that **mysql** is in:

Prompt	Meaning
`mysql>`	Ready for new command.
`->`	Waiting for next line of multiple-line command.
`'>`	Waiting for next line, collecting a string that begins with a single quote (`''`).
`">`	Waiting for next line, collecting a string that begins with a double quote (`'"'`).
`` `> ``	Waiting for next line, collecting an identifier that begins with a backtick (`` '`' ``).

Multiple-line statements commonly occur by accident when you intend to issue a command on a single line, but forget the terminating semicolon. In this case, **mysql** waits for more input:

```
mysql> SELECT USER()
    ->
```

If this happens to you (you think you've entered a statement but the only response is a `->` prompt), most likely **mysql** is waiting for the semicolon. If you don't notice what the prompt is telling you, you might sit there for a while before realizing what you need to do. Enter a semicolon to complete the statement, and **mysql** executes it:

```
mysql> SELECT USER()
    -> ;
+--------------------+
| USER()             |
+--------------------+
| joesmith@localhost |
+--------------------+
```

The `'>` and `">` prompts occur during string collection. In MySQL, you can write strings surrounded by either `''` or `'"'` characters (for example, `'hello'` or `"goodbye"`), and **mysql** lets you enter strings that span multiple lines. When you see a `'>` or `">` prompt, it means that you've entered a line containing a string that begins with a `''` or `'"'` quote character, but have not yet entered the matching quote that terminates the string. That's fine if you really are entering a multiple-line string, but how likely is that? Not very. More often, the `'>` and `">` prompts indicate that you've inadvertently left out a quote character. For example:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
    '>
```

If you enter this `SELECT` statement, then press Enter and wait for the result, nothing happens. Instead of wondering why this query takes so long, notice the clue provided by the `'>` prompt. It tells you that **mysql** expects to see the rest of an unterminated string. (Do you see the error in the statement? The string `'Smith` is missing the second quote.)

At this point, what do you do? The simplest thing is to cancel the command. However, you cannot just type `\c` in this case, because **mysql** interprets it as part of the string that it is collecting! Instead, enter

the closing quote character (so **mysql** knows you've finished the string), then type `\c`:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
    '> '\c
mysql>
```

The prompt changes back to `mysql>`, indicating that **mysql** is ready for a new command.

The `` `> `` prompt is similar to th `'>` and `">` prompts, but indicates that you have begun but not completed a backtick-quoted identifier.

It's important to know what the `'>`, `">`, and `` `> `` prompts signify, because if you mistakenly enter an unterminated string, any further lines you type appear to be ignored by **mysql**---including a line containing `QUIT`. This can be quite confusing, especially if you don't know that you need to supply the terminating quote before you can cancel the current command.

3.3. Creating and Using a Database

Once you know how to enter commands, it's time to access a database.

Suppose that you have several pets in your home (your menagerie) and you'd like to keep track of various types of information about them. You can do so by creating tables to hold your data and loading them with the desired information. Then you can answer different sorts of questions about your animals by retrieving data from the tables. This section shows you how to:

- Create a database
- Create a table
- Load data into the table
- Retrieve data from the table in various ways
- Use multiple tables

The menagerie database is simple (deliberately), but it is not difficult to think of real-world situations in which a similar type of database might be used. For example, a database like this could be used by a farmer to keep track of livestock, or by a veterinarian to keep track of patient records. A menagerie distribution containing some of the queries and sample data used in the following sections can be obtained from the MySQL Web site. It's available in either compressed **tar** format (http://www.mysql.com/Downloads/Contrib/Examples/menagerie.tar.gz) or Zip format (http://www.mysql.com/Downloads/Contrib/Examples/menagerie.zip).

Use the `SHOW` statement to find out what databases currently exist on the server:

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
| tmp      |
+----------+
```

The list of databases is probably different on your machine, but the `mysql` and `test` databases are likely to be among them. The `mysql` database is required because it describes user access privileges.

The `test` database is often provided as a workspace for users to try things out.

Note that you may not see all databases if you don't have the `SHOW DATABASES` privilege. See Section 13.5.1.3, “`GRANT` and `REVOKE` Syntax”.

If the `test` database exists, try to access it:

```
mysql> USE test
Database changed
```

Note that `USE`, like `QUIT`, does not require a semicolon. (You can terminate such statements with a semicolon if you like; it does no harm.) The `USE` statement is special in another way, too: it must be given on a single line.

You can use the `test` database (if you have access to it) for the examples that follow, but anything you create in that database can be removed by anyone else with access to it. For this reason, you should probably ask your MySQL administrator for permission to use a database of your own. Suppose that you want to call yours `menagerie`. The administrator needs to execute a command like this:

```
mysql> GRANT ALL ON menagerie.* TO 'your_mysql_name'@'your_client_host';
```

where `your_mysql_name` is the MySQL username assigned to you and `your_client_host` is the host from which you connect to the server.

3.3.1. Creating and Selecting a Database

If the administrator creates your database for you when setting up your permissions, you can begin using it. Otherwise, you need to create it yourself:

```
mysql> CREATE DATABASE menagerie;
```

Under Unix, database names are case sensitive (unlike SQL keywords), so you must always refer to your database as `menagerie`, not as `Menagerie`, `MENAGERIE`, or some other variant. This is also true for table names. (Under Windows, this restriction does not apply, although you must refer to databases and tables using the same lettercase throughout a given query.)

Creating a database does not select it for use; you must do that explicitly. To make `menagerie` the current database, use this command:

```
mysql> USE menagerie
Database changed
```

Your database needs to be created only once, but you must select it for use each time you begin a **mysql** session. You can do this by issuing a `USE` statement as shown in the example. Alternatively, you can select the database on the command line when you invoke **mysql**. Just specify its name after any connection parameters that you might need to provide. For example:

```
shell> mysql -h host -u user -p menagerie
Enter password: ********
```

Note that `menagerie` is not your password on the command just shown. If you want to supply your password on the command line after the `-p` option, you must do so with no intervening space (for example, as `-pmypassword`, not as `-p mypassword`). However, putting your password on the command line is not recommended, because doing so exposes it to snooping by other users logged in on your machine.

3.3.2. Creating a Table

Creating the database is the easy part, but at this point it's empty, as `SHOW TABLES` tells you:

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

The harder part is deciding what the structure of your database should be: what tables you need and what columns should be in each of them.

You'll want a table that contains a record for each of your pets. This can be called the `pet` table, and it should contain, as a bare minimum, each animal's name. Because the name by itself is not very interesting, the table should contain other information. For example, if more than one person in your family keeps pets, you might want to list each animal's owner. You might also want to record some basic descriptive information such as species and sex.

How about age? That might be of interest, but it's not a good thing to store in a database. Age changes as time passes, which means you'd have to update your records often. Instead, it's better to store a fixed value such as date of birth. Then, whenever you need age, you can calculate it as the difference between the current date and the birth date. MySQL provides functions for doing date arithmetic, so this is not difficult. Storing birth date rather than age has other advantages, too:

- You can use the database for tasks such as generating reminders for upcoming pet birthdays. (If you think this type of query is somewhat silly, note that it is the same question you might ask in the context of a business database to identify clients to whom you'll soon need to send out birthday greetings, for that computer-assisted personal touch.)
- You can calculate age in relation to dates other than the current date. For example, if you store death date in the database, you can easily calculate how old a pet was when it died.

You can probably think of other types of information that would be useful in the `pet` table, but the ones identified so far are sufficient: name, owner, species, sex, birth, and death.

Use a `CREATE TABLE` statement to specify the layout of your table:

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
    -> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

`VARCHAR` is a good choice for the `name`, `owner`, and `species` columns because the column values vary in length. The lengths of those columns need not all be the same, and need not be `20`. You can pick any length from `1` to `255`, whatever seems most reasonable to you. (If you make a poor choice and it turns out later that you need a longer field, MySQL provides an `ALTER TABLE` statement.)

Several types of values can be chosen to represent sex in animal records, such as `'m'` and `'f'`, or perhaps `'male'` and `'female'`. It's simplest to use the single characters `'m'` and `'f'`.

The use of the `DATE` data type for the `birth` and `death` columns is a fairly obvious choice.

Once you have created a table, `SHOW TABLES` should produce some output:

```
mysql> SHOW TABLES;
+---------------------+
| Tables in menagerie |
+---------------------+
| pet                 |
+---------------------+
```

To verify that your table was created the way you expected, use a `DESCRIBE` statement:

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

You can use `DESCRIBE` any time, for example, if you forget the names of the columns in your table or what types they have.

3.3.3. Loading Data into a Table

After creating your table, you need to populate it. The `LOAD DATA` and `INSERT` statements are useful for this.

Suppose that your pet records can be described as shown here. (Observe that MySQL expects dates in `'YYYY-MM-DD'` format; this may be different from what you are used to.)

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	

Because you are beginning with an empty table, an easy way to populate it is to create a text file containing a row for each of your animals, then load the contents of the file into the table with a single statement.

You could create a text file `pet.txt` containing one record per line, with values separated by tabs, and given in the order in which the columns were listed in the `CREATE TABLE` statement. For missing values (such as unknown sexes or death dates for animals that are still living), you can use `NULL` values. To represent these in your text file, use `\N` (backslash, capital-N). For example, the record for Whistler the bird would look like this (where the whitespace between values is a single tab character):

name	owner	species	sex	birth	death
`Whistler`	`Gwen`	`bird`	`\N`	`1997-12-09`	`\N`

To load the text file `pet.txt` into the `pet` table, use this command:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet;
```

Note that if you created the file on Windows with an editor that uses `\r\n` as a line terminator, you should use:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet
    -> LINES TERMINATED BY '\r\n';
```

You can specify the column value separator and end of line marker explicitly in the `LOAD DATA` statement if you wish, but the defaults are tab and linefeed. These are sufficient for the statement to read the file `pet.txt` properly.

If the statement fails, it is likely that your MySQL installation does not have local file capability enabled by default. See Section 5.4.4, “Security Issues with `LOAD DATA LOCAL`” for information on how to change this.

When you want to add new records one at a time, the `INSERT` statement is useful. In its simplest form, you supply values for each column, in the order in which the columns were listed in the `CREATE TABLE` statement. Suppose that Diane gets a new hamster named Puffball. You could add a new record using an `INSERT` statement like this:

```
mysql> INSERT INTO pet
    -> VALUES ('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

Note that string and date values are specified as quoted strings here. Also, with `INSERT`, you can insert `NULL` directly to represent a missing value. You do not use `\N` like you do with `LOAD DATA`.

From this example, you should be able to see that there would be a lot more typing involved to load your records initially using several `INSERT` statements rather than a single `LOAD DATA` statement.

3.3.4. Retrieving Information from a Table

The `SELECT` statement is used to pull information from a table. The general form of the statement is:

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy;
```

`what_to_select` indicates what you want to see. This can be a list of columns, or `*` to indicate ``all columns.'' `which_table` indicates the table from which you want to retrieve data. The `WHERE` clause is optional. If it's present, `conditions_to_satisfy` specifies conditions that rows must satisfy to qualify for retrieval.

3.3.4.1. Selecting All Data

The simplest form of `SELECT` retrieves everything from a table:

```
mysql> SELECT * FROM pet;
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       |
| Bowser   | Diane  | dog     | m    | 1979-08-31 | 1995-07-29 |
```

```
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       |
| Slim     | Benny  | snake   | m    | 1996-04-29 | NULL       |
| Puffball | Diane  | hamster | f    | 1999-03-30 | NULL       |
+----------+--------+---------+------+------------+------------+
```

This form of `SELECT` is useful if you want to review your entire table, for example, after you've just loaded it with your initial dataset. For example, you may happen to think that the birth date for Bowser doesn't seem quite right. Consulting your original pedigree papers, you find that the correct birth year should be 1989, not 1979.

There are least a couple of ways to fix this:

- Edit the file `pet.txt` to correct the error, then empty the table and reload it using `DELETE` and `LOAD DATA`:

  ```
  mysql> DELETE FROM pet;
  mysql> LOAD DATA LOCAL INFILE 'pet.txt' INTO TABLE pet;
  ```

  However, if you do this, you must also re-enter the record for Puffball.

- Fix only the erroneous record with an `UPDATE` statement:

  ```
  mysql> UPDATE pet SET birth = '1989-08-31' WHERE name = 'Bowser';
  ```

  The `UPDATE` changes only the record in question and does not require you to reload the table.

3.3.4.2. Selecting Particular Rows

As shown in the preceding section, it is easy to retrieve an entire table. Just omit the `WHERE` clause from the `SELECT` statement. But typically you don't want to see the entire table, particularly when it becomes large. Instead, you're usually more interested in answering a particular question, in which case you specify some constraints on the information you want. Let's look at some selection queries in terms of questions about your pets that they answer.

You can select only particular rows from your table. For example, if you want to verify the change that you made to Bowser's birth date, select Bowser's record like this:

```
mysql> SELECT * FROM pet WHERE name = 'Bowser';
+--------+-------+---------+------+------------+------------+
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
```

The output confirms that the year is correctly recorded as 1989, not 1979.

String comparisons normally are case-insensitive, so you can specify the name as `'bowser'`, `'BOWSER'`, etc. The query result is the same.

You can specify conditions on any column, not just `name`. For example, if you want to know which animals were born after 1998, test the `birth` column:

```
mysql> SELECT * FROM pet WHERE birth >= '1998-1-1';
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Puffball | Diane | hamster | f    | 1999-03-30 | NULL  |
+----------+-------+---------+------+------------+-------+
```

You can combine conditions, for example, to locate female dogs:

```
mysql> SELECT * FROM pet WHERE species = 'dog' AND sex = 'f';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

The preceding query uses the `AND` logical operator. There is also an `OR` operator:

```
mysql> SELECT * FROM pet WHERE species = 'snake' OR species = 'bird';
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL  |
| Slim     | Benny | snake   | m    | 1996-04-29 | NULL  |
+----------+-------+---------+------+------------+-------+
```

`AND` and `OR` may be intermixed, although `AND` has higher precedence than `OR`. If you use both operators, it's a good idea to use parentheses to indicate explicitly how conditions should be grouped:

```
mysql> SELECT * FROM pet WHERE (species = 'cat' AND sex = 'm')
    -> OR (species = 'dog' AND sex = 'f');
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

3.3.4.3. Selecting Particular Columns

If you don't want to see entire rows from your table, just name the columns in which you're interested, separated by commas. For example, if you want to know when your animals were born, select the `name` and `birth` columns:

```
mysql> SELECT name, birth FROM pet;
+----------+------------+
| name     | birth      |
+----------+------------+
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Buffy    | 1989-05-13 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
```

```
| Puffball | 1999-03-30 |
+----------+------------+
```

To find out who owns pets, use this query:

```
mysql> SELECT owner FROM pet;
+--------+
| owner  |
+--------+
| Harold |
| Gwen   |
| Harold |
| Benny  |
| Diane  |
| Gwen   |
| Gwen   |
| Benny  |
| Diane  |
+--------+
```

However, notice that the query simply retrieves the `owner` field from each record, and some of them appear more than once. To minimize the output, retrieve each unique output record just once by adding the keyword `DISTINCT`:

```
mysql> SELECT DISTINCT owner FROM pet;
+--------+
| owner  |
+--------+
| Benny  |
| Diane  |
| Gwen   |
| Harold |
+--------+
```

You can use a `WHERE` clause to combine row selection with column selection. For example, to get birth dates for dogs and cats only, use this query:

```
mysql> SELECT name, species, birth FROM pet
    -> WHERE species = 'dog' OR species = 'cat';
+--------+---------+------------+
| name   | species | birth      |
+--------+---------+------------+
| Fluffy | cat     | 1993-02-04 |
| Claws  | cat     | 1994-03-17 |
| Buffy  | dog     | 1989-05-13 |
| Fang   | dog     | 1990-08-27 |
| Bowser | dog     | 1989-08-31 |
+--------+---------+------------+
```

3.3.4.4. Sorting Rows

You may have noticed in the preceding examples that the result rows are displayed in no particular order. It's often easier to examine query output when the rows are sorted in some meaningful way. To sort a result, use an `ORDER BY` clause.

Here are animal birthdays, sorted by date:

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
```

```
+----------+------------+
| name     | birth      |
+----------+------------+
| Buffy    | 1989-05-13 |
| Bowser   | 1989-08-31 |
| Fang     | 1990-08-27 |
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Slim     | 1996-04-29 |
| Whistler | 1997-12-09 |
| Chirpy   | 1998-09-11 |
| Puffball | 1999-03-30 |
+----------+------------+
```

On character type columns, sorting---like all other comparison operations---is normally performed in a case-insensitive fashion. This means that the order is undefined for columns that are identical except for their case. You can force a case-sensitive sort for a column by using the `BINARY` cast: `ORDER BY BINARY col_name`.

The default sort order is ascending, with smallest values first. To sort in reverse (descending) order, add the `DESC` keyword to the name of the column you are sorting by:

```
mysql> SELECT name, birth FROM pet ORDER BY birth DESC;
+----------+------------+
| name     | birth      |
+----------+------------+
| Puffball | 1999-03-30 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Claws    | 1994-03-17 |
| Fluffy   | 1993-02-04 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Buffy    | 1989-05-13 |
+----------+------------+
```

You can sort on multiple columns, and you can sort columns in different directions. For example, to sort by type of animal in ascending order, then by birth date within animal type in descending order (youngest animals first), use the following query:

```
mysql> SELECT name, species, birth FROM pet
    -> ORDER BY species, birth DESC;
+----------+---------+------------+
| name     | species | birth      |
+----------+---------+------------+
| Chirpy   | bird    | 1998-09-11 |
| Whistler | bird    | 1997-12-09 |
| Claws    | cat     | 1994-03-17 |
| Fluffy   | cat     | 1993-02-04 |
| Fang     | dog     | 1990-08-27 |
| Bowser   | dog     | 1989-08-31 |
| Buffy    | dog     | 1989-05-13 |
| Puffball | hamster | 1999-03-30 |
| Slim     | snake   | 1996-04-29 |
+----------+---------+------------+
```

Note that the `DESC` keyword applies only to the column name immediately preceding it (`birth`); it does not affect the `species` column sort order.

3.3.4.5. Date Calculations

MySQL provides several functions that you can use to perform calculations on dates, for example, to calculate ages or extract parts of dates.

To determine how many years old each of your pets is, compute the difference in the year part of the current date and the birth date, then subtract one if the current date occurs earlier in the calendar year than the birth date. The following query shows, for each pet, the birth date, the current date, and the age in years.

```
mysql> SELECT name, birth, CURDATE(),
    -> (YEAR(CURDATE())-YEAR(birth))
    -> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
    -> AS age
    -> FROM pet;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
+----------+------------+------------+------+
```

Here, `YEAR()` pulls out the year part of a date and `RIGHT()` pulls off the rightmost five characters that represent the `MM-DD` (calendar year) part of the date. The part of the expression that compares the `MM-DD` values evaluates to 1 or 0, which adjusts the year difference down a year if `CURDATE()` occurs earlier in the year than `birth`. The full expression is somewhat ungainly, so an alias (`age`) is used to make the output column label more meaningful.

The query works, but the result could be scanned more easily if the rows were presented in some order. This can be done by adding an `ORDER BY name` clause to sort the output by name:

```
mysql> SELECT name, birth, CURDATE(),
    -> (YEAR(CURDATE())-YEAR(birth))
    -> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
    -> AS age
    -> FROM pet ORDER BY name;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
+----------+------------+------------+------+
```

To sort the output by `age` rather than `name`, just use a different `ORDER BY` clause:

```
mysql> SELECT name, birth, CURDATE(),
```

```
    -> (YEAR(CURDATE())-YEAR(birth))
    -> - (RIGHT(CURDATE(),5)<RIGHT(birth,5))
    -> AS age
    -> FROM pet ORDER BY age;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
+----------+------------+------------+------+
```

A similar query can be used to determine age at death for animals that have died. You determine which animals these are by checking whether the `death` value is `NULL`. Then, for those with non-`NULL` values, compute the difference between the `death` and `birth` values:

```
mysql> SELECT name, birth, death,
    -> (YEAR(death)-YEAR(birth)) - (RIGHT(death,5)<RIGHT(birth,5))
    -> AS age
    -> FROM pet WHERE death IS NOT NULL ORDER BY age;
+--------+------------+------------+------+
| name   | birth      | death      | age  |
+--------+------------+------------+------+
| Bowser | 1989-08-31 | 1995-07-29 |    5 |
+--------+------------+------------+------+
```

The query uses `death IS NOT NULL` rather than `death <> NULL` because `NULL` is a special value that cannot be compared using the usual comparison operators. This is discussed later. See Section 3.3.4.6, “Working with `NULL` Values”.

What if you want to know which animals have birthdays next month? For this type of calculation, year and day are irrelevant; you simply want to extract the month part of the `birth` column. MySQL provides several date-part extraction functions, such as `YEAR()`, `MONTH()`, and `DAYOFMONTH()`. `MONTH()` is the appropriate function here. To see how it works, run a simple query that displays the value of both `birth` and `MONTH(birth)`:

```
mysql> SELECT name, birth, MONTH(birth) FROM pet;
+----------+------------+--------------+
| name     | birth      | MONTH(birth) |
+----------+------------+--------------+
| Fluffy   | 1993-02-04 |            2 |
| Claws    | 1994-03-17 |            3 |
| Buffy    | 1989-05-13 |            5 |
| Fang     | 1990-08-27 |            8 |
| Bowser   | 1989-08-31 |            8 |
| Chirpy   | 1998-09-11 |            9 |
| Whistler | 1997-12-09 |           12 |
| Slim     | 1996-04-29 |            4 |
| Puffball | 1999-03-30 |            3 |
+----------+------------+--------------+
```

Finding animals with birthdays in the upcoming month is easy, too. Suppose that the current month is April. Then the month value is `4` and you look for animals born in May (month `5`) like this:

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
+-------+------------+
| name  | birth      |
+-------+------------+
| Buffy | 1989-05-13 |
+-------+------------+
```

There is a small complication if the current month is December. You don't just add one to the month number (12) and look for animals born in month 13, because there is no such month. Instead, you look for animals born in January (month 1).

You can even write the query so that it works no matter what the current month is. That way you don't have to use a particular month number in the query. DATE_ADD() allows you to add a time interval to a given date. If you add a month to the value of CURDATE(), then extract the month part with MONTH(), the result produces the month in which to look for birthdays:

```
mysql> SELECT name, birth FROM pet
    -> WHERE MONTH(birth) = MONTH(DATE_ADD(CURDATE(),INTERVAL 1 MONTH));
```

A different way to accomplish the same task is to add 1 to get the next month after the current one (after using the modulo function (MOD) to wrap around the month value to 0 if it is currently 12):

```
mysql> SELECT name, birth FROM pet
    -> WHERE MONTH(birth) = MOD(MONTH(CURDATE()), 12) + 1;
```

Note that MONTH returns a number between 1 and 12. And MOD(something,12) returns a number between 0 and 11. So the addition has to be after the MOD(), otherwise we would go from November (11) to January (1).

3.3.4.6. Working with NULL Values

The NULL value can be surprising until you get used to it. Conceptually, NULL means missing value or unknown value and it is treated somewhat differently than other values. To test for NULL, you cannot use the arithmetic comparison operators such as =, <, or <>. To demonstrate this for yourself, try the following query:

```
mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
+----------+-----------+----------+----------+
| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |
+----------+-----------+----------+----------+
|     NULL |      NULL |     NULL |     NULL |
+----------+-----------+----------+----------+
```

Clearly you get no meaningful results from these comparisons. Use the IS NULL and IS NOT NULL operators instead:

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----------+---------------+
| 1 IS NULL | 1 IS NOT NULL |
+-----------+---------------+
|         0 |             1 |
+-----------+---------------+
```

Note that in MySQL, 0 or NULL means false and anything else means true. The default truth value from a boolean operation is 1.

This special treatment of NULL is why, in the previous section, it was necessary to determine which animals are no longer alive using death IS NOT NULL instead of death <> NULL.

Two NULL values are regarded as equal in a GROUP BY.

When doing an ORDER BY, NULL values are presented first if you do ORDER BY ... ASC and last if you do ORDER BY ... DESC.

Note that MySQL 4.0.2 to 4.0.10 incorrectly always sorts NULL values first regardless of the sort direction.

3.3.4.7. Pattern Matching

MySQL provides standard SQL pattern matching as well as a form of pattern matching based on extended regular expressions similar to those used by Unix utilities such as vi, grep, and sed.

SQL pattern matching allows you to use '_' to match any single character and '%' to match an arbitrary number of characters (including zero characters). In MySQL, SQL patterns are case-insensitive by default. Some examples are shown here. Note that you do not use = or <> when you use SQL patterns; use the LIKE or NOT LIKE comparison operators instead.

To find names beginning with 'b':

```
mysql> SELECT * FROM pet WHERE name LIKE 'b%';
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

To find names ending with 'fy':

```
mysql> SELECT * FROM pet WHERE name LIKE '%fy';
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

To find names containing a 'w':

```
mysql> SELECT * FROM pet WHERE name LIKE '%w%';
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

To find names containing exactly five characters, use five instances of the '_' pattern character:

```
mysql> SELECT * FROM pet WHERE name LIKE '_____';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
```

```
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

The other type of pattern matching provided by MySQL uses extended regular expressions. When you test for a match for this type of pattern, use the `REGEXP` and `NOT REGEXP` operators (or `RLIKE` and `NOT RLIKE`, which are synonyms).

Some characteristics of extended regular expressions are:

- '`.`' matches any single character.
- A character class '`[...]`' matches any character within the brackets. For example, '`[abc]`' matches 'a', 'b', or 'c'. To name a range of characters, use a dash. '`[a-z]`' matches any letter, whereas '`[0-9]`' matches any digit.
- '`*`' matches zero or more instances of the thing preceding it. For example, '`x*`' matches any number of 'x' characters, '`[0-9]*`' matches any number of digits, and '`.*`' matches any number of anything.
- A `REGEXP` pattern match succeed if the pattern matches anywhere in the value being tested. (This differs from a `LIKE` pattern match, which succeeds only if the pattern matches the entire value.)
- To anchor a pattern so that it must match the beginning or end of the value being tested, use '`^`' at the beginning or '`$`' at the end of the pattern.

To demonstrate how extended regular expressions work, the `LIKE` queries shown previously are rewritten here to use `REGEXP`.

To find names beginning with 'b', use '`^`' to match the beginning of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^b';
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

Prior to MySQL Version 3.23.4, `REGEXP` is case sensitive, and the previous query will return no rows. In this case, to match either lowercase or uppercase 'b', use this query instead:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^[bB]';
```

From MySQL 3.23.4 on, if you really want to force a `REGEXP` comparison to be case sensitive, use the `BINARY` keyword to make one of the strings a binary string. This query matches only lowercase 'b' at the beginning of a name:

```
mysql> SELECT * FROM pet WHERE name REGEXP BINARY '^b';
```

To find names ending with 'fy', use '`$`' to match the end of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP 'fy$';
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
```

```
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

To find names containing a 'w', use this query:

```
mysql> SELECT * FROM pet WHERE name REGEXP 'w';
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

Because a regular expression pattern matches if it occurs anywhere in the value, it is not necessary in the previous query to put a wildcard on either side of the pattern to get it to match the entire value like it would be if you used an SQL pattern.

To find names containing exactly five characters, use '^' and '$' to match the beginning and end of the name, and five instances of '.' in between:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^.....$';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

You could also write the previous query using the '{n}' ``repeat-n-times" operator:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^.{5}$';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

3.3.4.8. Counting Rows

Databases are often used to answer the question, ``How often does a certain type of data occur in a table?" For example, you might want to know how many pets you have, or how many pets each owner has, or you might want to perform various kinds of census operations on your animals.

Counting the total number of animals you have is the same question as ``How many rows are in the `pet` table?" because there is one record per pet. `COUNT(*)` counts the number of rows, so the query to count your animals looks like this:

```
mysql> SELECT COUNT(*) FROM pet;
+----------+
| COUNT(*) |
+----------+
|        9 |
+----------+
```

Earlier, you retrieved the names of the people who owned pets. You can use `COUNT()` if you want to find out how many pets each owner has:

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Benny  |        2 |
| Diane  |        2 |
| Gwen   |        3 |
| Harold |        2 |
+--------+----------+
```

Note the use of `GROUP BY` to group together all records for each `owner`. Without it, all you get is an error message:

```
mysql> SELECT owner, COUNT(*) FROM pet;
ERROR 1140: Mixing of GROUP columns (MIN(),MAX(),COUNT()...)
with no GROUP columns is illegal if there is no GROUP BY clause
```

`COUNT()` and `GROUP BY` are useful for characterizing your data in various ways. The following examples show different ways to perform animal census operations.

Number of animals per species:

```
mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
+---------+----------+
| species | COUNT(*) |
+---------+----------+
| bird    |        2 |
| cat     |        2 |
| dog     |        3 |
| hamster |        1 |
| snake   |        1 |
+---------+----------+
```

Number of animals per sex:

```
mysql> SELECT sex, COUNT(*) FROM pet GROUP BY sex;
+------+----------+
| sex  | COUNT(*) |
+------+----------+
| NULL |        1 |
| f    |        4 |
| m    |        4 |
+------+----------+
```

(In this output, `NULL` indicates that the sex is unknown.)

Number of animals per combination of species and sex:

```
mysql> SELECT species, sex, COUNT(*) FROM pet GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| bird    | NULL |        1 |
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
```

```
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

You need not retrieve an entire table when you use `COUNT()`. For example, the previous query, when performed just on dogs and cats, looks like this:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
    -> WHERE species = 'dog' OR species = 'cat'
    -> GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
+---------+------+----------+
```

Or, if you wanted the number of animals per sex only for known-sex animals:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
    -> WHERE sex IS NOT NULL
    -> GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

3.3.4.9. Using More Than one Table

The `pet` table keeps track of which pets you have. If you want to record other information about them, such as events in their lives like visits to the vet or when litters are born, you need another table. What should this table look like? It needs:

- To contain the pet name so you know which animal each event pertains to.
- A date so you know when the event occurred.
- A field to describe the event.
- An event type field, if you want to be able to categorize events.

Given these considerations, the `CREATE TABLE` statement for the `event` table might look like this:

```
mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
    -> type VARCHAR(15), remark VARCHAR(255));
```

As with the `pet` table, it's easiest to load the initial records by creating a tab-delimited text file containing the information:

name	date	type	remark
Fluffy	1995-05-15	litter	4 kittens, 3 female, 1 male
Buffy	1993-06-23	litter	5 puppies, 2 female, 3 male
Buffy	1994-06-19	litter	3 puppies, 3 female
Chirpy	1999-03-21	vet	needed beak straightened
Slim	1997-08-03	vet	broken rib
Bowser	1991-10-12	kennel	
Fang	1991-10-12	kennel	
Fang	1998-08-28	birthday	Gave him a new chew toy
Claws	1998-03-17	birthday	Gave him a new flea collar
Whistler	1998-12-09	birthday	First birthday

Load the records like this:

```
mysql> LOAD DATA LOCAL INFILE 'event.txt' INTO TABLE event;
```

Based on what you've learned from the queries you've run on the `pet` table, you should be able to perform retrievals on the records in the `event` table; the principles are the same. But when is the `event` table by itself insufficient to answer questions you might ask?

Suppose that you want to find out the ages at which each pet had its litters. We saw earlier how to calculate ages from two dates. The litter date of the mother is in the `event` table, but to calculate her age on that date you need her birth date, which is stored in the `pet` table. This means the query requires both tables:

```
mysql> SELECT pet.name,
    -> (YEAR(date)-YEAR(birth)) - (RIGHT(date,5)<RIGHT(birth,5)) AS age,
    -> remark
    -> FROM pet, event
    -> WHERE pet.name = event.name AND type = 'litter';
+--------+------+----------------------------+
| name   | age  | remark                     |
+--------+------+----------------------------+
| Fluffy |    2 | 4 kittens, 3 female, 1 male |
| Buffy  |    4 | 5 puppies, 2 female, 3 male |
| Buffy  |    5 | 3 puppies, 3 female        |
+--------+------+----------------------------+
```

There are several things to note about this query:

- The `FROM` clause lists two tables because the query needs to pull information from both of them.
- When combining (joining) information from multiple tables, you need to specify how records in one table can be matched to records in the other. This is easy because they both have a `name` column. The query uses `WHERE` clause to match up records in the two tables based on the `name` values.
- Because the `name` column occurs in both tables, you must be specific about which table you mean when referring to the column. This is done by prepending the table name to the column name.

You need not have two different tables to perform a join. Sometimes it is useful to join a table to itself, if you want to compare records in a table to other records in that same table. For example, to find breeding pairs among your pets, you can join the `pet` table with itself to produce candidate pairs of males and females of like species:

```
mysql> SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species
    -> FROM pet AS p1, pet AS p2
    -> WHERE p1.species = p2.species AND p1.sex = 'f' AND p2.sex = 'm';
+--------+------+--------+------+---------+
| name   | sex  | name   | sex  | species |
+--------+------+--------+------+---------+
| Fluffy | f    | Claws  | m    | cat     |
| Buffy  | f    | Fang   | m    | dog     |
| Buffy  | f    | Bowser | m    | dog     |
+--------+------+--------+------+---------+
```

In this query, we specify aliases for the table name in order to refer to the columns and keep straight which instance of the table each column reference is associated with.

3.4. Getting Information About Databases and Tables

What if you forget the name of a database or table, or what the structure of a given table is (for example, what its columns are called)? MySQL addresses this problem through several statements that provide information about the databases and tables it supports.

You have previously seen `SHOW DATABASES`, which lists the databases managed by the server. To find out which database is currently selected, use the `DATABASE()` function:

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| menagerie  |
+------------+
```

If you haven't selected any database yet, the result is `NULL` (or the empty string before MySQL 4.1.1).

To find out what tables the current database contains (for example, when you're not sure about the name of a table), use this command:

```
mysql> SHOW TABLES;
+---------------------+
| Tables in menagerie |
+---------------------+
| event               |
| pet                 |
+---------------------+
```

If you want to find out about the structure of a table, the `DESCRIBE` command is useful; it displays information about each of a table's columns:

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
```

```
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

`Field` indicates the column name, `Type` is the data type for the column, `NULL` indicates whether the column can contain `NULL` values, `Key` indicates whether the column is indexed, and `Default` specifies the column's default value.

If you have indexes on a table, `SHOW INDEX FROM` *`tbl_name`* produces information about them.

3.5. Using mysql in Batch Mode

In the previous sections, you used **mysql** interactively to enter queries and view the results. You can also run **mysql** in batch mode. To do this, put the commands you want to run in a file, then tell **mysql** to read its input from the file:

```
shell> mysql < batch-file
```

If you are running **mysql** under Windows and have some special characters in the file that cause problems, you can do this:

```
C:\> mysql -e "source batch-file"
```

If you need to specify connection parameters on the command line, the command might look like this:

```
shell> mysql -h host -u user -p < batch-file
Enter password: ********
```

When you use **mysql** this way, you are creating a script file, then executing the script.

If you want the script to continue even if some of the statements in it produce errors, you should use the `--force` command-line option.

Why use a script? Here are a few reasons:

- If you run a query repeatedly (say, every day or every week), making it a script allows you to avoid retyping it each time you execute it.
- You can generate new queries from existing ones that are similar by copying and editing script files.
- Batch mode can also be useful while you're developing a query, particularly for multiple-line commands or multiple-statement sequences of commands. If you make a mistake, you don't have to retype everything. Just edit your script to correct the error, then tell **mysql** to execute it again.
- If you have a query that produces a lot of output, you can run the output through a pager rather than watching it scroll off the top of your screen:

  ```
  shell> mysql < batch-file | more
  ```

- You can catch the output in a file for further processing:

```
shell> mysql < batch-file > mysql.out
```

- You can distribute your script to other people so they can run the commands, too.
- Some situations do not allow for interactive use, for example, when you run a query from a **cron** job. In this case, you must use batch mode.

The default output format is different (more concise) when you run **mysql** in batch mode than when you use it interactively. For example, the output of `SELECT DISTINCT species FROM pet` looks like this when **mysql** is run interactively:

```
+---------+
| species |
+---------+
| bird    |
| cat     |
| dog     |
| hamster |
| snake   |
+---------+
```

In batch mode, the output looks like this instead:

```
species
bird
cat
dog
hamster
snake
```

If you want to get the interactive output format in batch mode, use `mysql -t`. To echo to the output the commands that are executed, use `mysql -vvv`.

You can also use scripts from the **mysql** prompt by using the `source` or `\.` command:

```
mysql> source filename;
mysql> \. filename
```

3.6. Examples of Common Queries

Here are examples of how to solve some common problems with MySQL.

Some of the examples use the table `shop` to hold the price of each article (item number) for certain traders (dealers). Supposing that each trader has a single fixed price per article, then (`article`, `dealer`) is a primary key for the records.

Start the command-line tool **mysql** and select a database:

```
shell> mysql your-database-name
```

(In most MySQL installations, you can use the database name `test`).

You can create and populate the example table with these statements:

```
mysql> CREATE TABLE shop (
    -> article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
    -> dealer  CHAR(20)                 DEFAULT ''     NOT NULL,
    -> price   DOUBLE(16,2)             DEFAULT '0.00' NOT NULL,
    -> PRIMARY KEY(article, dealer));
mysql> INSERT INTO shop VALUES
    -> (1,'A',3.45),(1,'B',3.99),(2,'A',10.99),(3,'B',1.45),
    -> (3,'C',1.69),(3,'D',1.25),(4,'D',19.95);
```

After issuing the statements, the table should have the following contents:

```
mysql> SELECT * FROM shop;
+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0001 | A      |  3.45 |
|    0001 | B      |  3.99 |
|    0002 | A      | 10.99 |
|    0003 | B      |  1.45 |
|    0003 | C      |  1.69 |
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

3.6.1. The Maximum Value for a Column

``What's the highest item number?''

```
SELECT MAX(article) AS article FROM shop;

+---------+
| article |
+---------+
|       4 |
+---------+
```

3.6.2. The Row Holding the Maximum of a Certain Column

``Find number, dealer, and price of the most expensive article.''

In standard SQL (and as of MySQL 4.1), this is easily done with a subquery:

```
SELECT article, dealer, price
FROM   shop
WHERE  price=(SELECT MAX(price) FROM shop);
```

In MySQL versions prior to 4.1, just do it in two steps:

1. Get the maximum price value from the table with a `SELECT` statement.

   ```
   mysql> SELECT MAX(price) FROM shop;
   +------------+
   | MAX(price) |
   +------------+
   ```

```
|      19.95 |
+------------+
```

2. Using the value 19.95 shown by the previous query to be the maximum article price, write a query to locate and display the corresponding record:

```
mysql> SELECT article, dealer, price
    -> FROM   shop
    -> WHERE  price=19.95;
+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

Another solution is to sort all rows descending by price and only get the first row using the MySQL-specific `LIMIT` clause:

```
SELECT article, dealer, price
FROM   shop
ORDER BY price DESC
LIMIT 1;
```

Note: If there were several most expensive articles, each with a price of 19.95, the `LIMIT` solution would show only one of them!

3.6.3. Maximum of Column per Group

``What's the highest price per article?''

```
SELECT article, MAX(price) AS price
FROM   shop
GROUP BY article

+---------+-------+
| article | price |
+---------+-------+
|    0001 |  3.99 |
|    0002 | 10.99 |
|    0003 |  1.69 |
|    0004 | 19.95 |
+---------+-------+
```

3.6.4. The Rows Holding the Group-wise Maximum of a Certain Field

``For each article, find the dealer or dealers with the most expensive price.''

In standard SQL (and as of MySQL 4.1), the problem can be solved with a subquery like this:

```
SELECT article, dealer, price
FROM   shop s1
WHERE  price=(SELECT MAX(s2.price)
              FROM shop s2
```

```
              WHERE s1.article = s2.article);
```

In MySQL versions prior to 4.1, it's best do it in several steps:

1. Get the list of (article,maxprice) pairs.
2. For each article, get the corresponding rows that have the stored maximum price.

This can easily be done with a temporary table and a join:

```
CREATE TEMPORARY TABLE tmp (
        article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
        price   DOUBLE(16,2)             DEFAULT '0.00' NOT NULL);

LOCK TABLES shop READ;

INSERT INTO tmp SELECT article, MAX(price) FROM shop GROUP BY article;

SELECT shop.article, dealer, shop.price FROM shop, tmp
WHERE shop.article=tmp.article AND shop.price=tmp.price;

UNLOCK TABLES;

DROP TABLE tmp;
```

If you don't use a `TEMPORARY` table, you must also lock the `tmp` table.

``Can it be done with a single query?''

Yes, but only by using a quite inefficient trick called the ``MAX-CONCAT trick'':

```
SELECT article,
       SUBSTRING( MAX( CONCAT(LPAD(price,6,'0'),dealer) ), 7) AS dealer,
  0.00+LEFT(      MAX( CONCAT(LPAD(price,6,'0'),dealer) ), 6) AS price
FROM   shop
GROUP BY article;

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0001 | B      |  3.99 |
|    0002 | A      | 10.99 |
|    0003 | C      |  1.69 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

The last example can be made a bit more efficient by doing the splitting of the concatenated column in the client.

3.6.5. Using User Variables

You can use MySQL user variables to remember results without having to store them in temporary variables in the client. See Section 9.3, “User Variables”.

For example, to find the articles with the highest and lowest price you can do this:

```
mysql> SELECT @min_price:=MIN(price),@max_price:=MAX(price) FROM shop;
mysql> SELECT * FROM shop WHERE price=@min_price OR price=@max_price;
+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

3.6.6. Using Foreign Keys

In MySQL 3.23.44 and up, `InnoDB` tables support checking of foreign key constraints. See Chapter 15, *The `InnoDB` Storage Engine*. See also Section 1.5.5.5, “Foreign Keys”.

You don't actually need foreign keys to join two tables. For table types other than `InnoDB`, the only things MySQL currently doesn't do are 1) `CHECK` to make sure that the keys you use really exist in the table or tables you're referencing and 2) automatically delete rows from a table with a foreign key definition. Using your keys to join tables works just fine:

```
CREATE TABLE person (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    name CHAR(60) NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE shirt (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    style ENUM('t-shirt', 'polo', 'dress') NOT NULL,
    color ENUM('red', 'blue', 'orange', 'white', 'black') NOT NULL,
    owner SMALLINT UNSIGNED NOT NULL REFERENCES person(id),
    PRIMARY KEY (id)
);

INSERT INTO person VALUES (NULL, 'Antonio Paz');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'polo', 'blue', @last),
(NULL, 'dress', 'white', @last),
(NULL, 't-shirt', 'blue', @last);

INSERT INTO person VALUES (NULL, 'Lilliana Angelovska');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'dress', 'orange', @last),
(NULL, 'polo', 'red', @last),
(NULL, 'dress', 'blue', @last),
(NULL, 't-shirt', 'white', @last);

SELECT * FROM person;
+----+---------------------+
| id | name                |
+----+---------------------+
|  1 | Antonio Paz         |
|  2 | Lilliana Angelovska |
+----+---------------------+

SELECT * FROM shirt;
```

```
+----+---------+--------+-------+
| id | style   | color  | owner |
+----+---------+--------+-------+
|  1 | polo    | blue   |     1 |
|  2 | dress   | white  |     1 |
|  3 | t-shirt | blue   |     1 |
|  4 | dress   | orange |     2 |
|  5 | polo    | red    |     2 |
|  6 | dress   | blue   |     2 |
|  7 | t-shirt | white  |     2 |
+----+---------+--------+-------+
```

```
SELECT s.* FROM person p, shirt s
 WHERE p.name LIKE 'Lilliana%'
   AND s.owner = p.id
   AND s.color <> 'white';

+----+-------+--------+-------+
| id | style | color  | owner |
+----+-------+--------+-------+
|  4 | dress | orange |     2 |
|  5 | polo  | red    |     2 |
|  6 | dress | blue   |     2 |
+----+-------+--------+-------+
```

3.6.7. Searching on Two Keys

An `OR` using a single key is well optimized, as is the handling of `AND`.

The one tricky case is that of searching on two different keys combined with `OR`:

```
SELECT field1_index, field2_index FROM test_table
WHERE field1_index = '1' OR  field2_index = '1'
```

This case is optimized from MySQL 5.0.0. See Section 7.2.6, “Index Merge Optimization”.

In MySQL 4.0 and up, you can also solve the problem efficiently by using a `UNION` that combines the output of two separate `SELECT` statements. See Section 13.1.7.2, “`UNION` Syntax”.

Each `SELECT` searches only one key and can be optimized:

```
SELECT field1_index, field2_index
    FROM test_table WHERE field1_index = '1'
UNION
SELECT field1_index, field2_index
    FROM test_table WHERE field2_index = '1';
```

Prior to MySQL 4.0, you can achieve the same effect by using a `TEMPORARY` table and separate `SELECT` statements. This type of optimization is also very good if you are using very complicated queries where the SQL server does the optimizations in the wrong order.

```
CREATE TEMPORARY TABLE tmp
SELECT field1_index, field2_index
    FROM test_table WHERE field1_index = '1';
INSERT INTO tmp
SELECT field1_index, field2_index
    FROM test_table WHERE field2_index = '1';
SELECT * from tmp;
```

```
DROP TABLE tmp;
```

This method of solving the problem is in effect a `UNION` of two queries.

3.6.8. Calculating Visits Per Day

The following example shows how you can use the bit group functions to calculate the number of days per month a user has visited a Web page.

```
CREATE TABLE t1 (year YEAR(4), month INT(2) UNSIGNED ZEROFILL,
             day INT(2) UNSIGNED ZEROFILL);
INSERT INTO t1 VALUES(2000,1,1),(2000,1,20),(2000,1,30),(2000,2,2),
            (2000,2,23),(2000,2,23);
```

The example table contains year-month-day values representing visits by users to the page. To determine how many different days in each month these visits occur, use this query:

```
SELECT year,month,BIT_COUNT(BIT_OR(1<<day)) AS days FROM t1
       GROUP BY year,month;
```

Which returns:

```
+------+-------+------+
| year | month | days |
+------+-------+------+
| 2000 |    01 |    3 |
| 2000 |    02 |    2 |
+------+-------+------+
```

The query calculates how many different days appear in the table for each year/month combination, with automatic removal of duplicate entries.

3.6.9. Using `AUTO_INCREMENT`

The `AUTO_INCREMENT` attribute can be used to generate a unique identity for new rows:

```
CREATE TABLE animals (
             id MEDIUMINT NOT NULL AUTO_INCREMENT,
             name CHAR(30) NOT NULL,
             PRIMARY KEY (id)
             );
INSERT INTO animals (name) VALUES ('dog'),('cat'),('penguin'),
                                  ('lax'),('whale'),('ostrich');
SELECT * FROM animals;
```

Which returns:

```
+----+---------+
| id | name    |
+----+---------+
|  1 | dog     |
|  2 | cat     |
|  3 | penguin |
|  4 | lax     |
|  5 | whale   |
|  6 | ostrich |
```

```
+----+---------+
```

You can retrieve the most recent AUTO_INCREMENT value with the LAST_INSERT_ID() SQL function or the mysql_insert_id() C API function. These functions are connection-specific, so their return value is not affected by another connection also doing inserts.

Note: For a multiple-row insert, LAST_INSERT_ID()/mysql_insert_id() actually returns the AUTO_INCREMENT key from the **first** of the inserted rows. This allows multiple-row inserts to be reproduced correctly on other servers in a replication setup.

For MyISAM and BDB tables you can specify AUTO_INCREMENT on a secondary column in a multiple-column index. In this case, the generated value for the AUTO_INCREMENT column is calculated as MAX(auto_increment_column)+1 WHERE prefix=given-prefix. This is useful when you want to put data into ordered groups.

```
CREATE TABLE animals (
             grp ENUM('fish','mammal','bird') NOT NULL,
             id MEDIUMINT NOT NULL AUTO_INCREMENT,
             name CHAR(30) NOT NULL,
             PRIMARY KEY (grp,id)
             );
INSERT INTO animals (grp,name) VALUES('mammal','dog'),('mammal','cat'),
                  ('bird','penguin'),('fish','lax'),('mammal','whale'),
                  ('bird','ostrich');
SELECT * FROM animals ORDER BY grp,id;
```

Which returns:

```
+--------+----+---------+
| grp    | id | name    |
+--------+----+---------+
| fish   |  1 | lax     |
| mammal |  1 | dog     |
| mammal |  2 | cat     |
| mammal |  3 | whale   |
| bird   |  1 | penguin |
| bird   |  2 | ostrich |
+--------+----+---------+
```

Note that in this case (when the AUTO_INCREMENT column is part of a multiple-column index), AUTO_INCREMENT values are reused if you delete the row with the biggest AUTO_INCREMENT value in any group. This happens even for MyISAM tables, for which AUTO_INCREMENT values normally are not reused.)

If the AUTO_INCREMENT column is part of multiple indexes, MySQL will generate sequence values using the index that begins with the AUTO_INCREMENT column, if there is one. For example, if the animals table contained indexes PRIMARY KEY (grp, id) and INDEX (id), MySQL would ignore the PRIMARY KEY for generating sequence values. As a result, the table would contain a single sequence, not a sequence per grp value.

3.7. Queries from the Twin Project

At Analytikerna and Lentus, we have been doing the systems and field work for a big research project. This project is a collaboration between the Institute of Environmental Medicine at Karolinska Institutet Stockholm and the Section on Clinical Research in Aging and Psychology at the University of Southern California.

The project involves a screening part where all twins in Sweden older than 65 years are interviewed by telephone. Twins who meet certain criteria are passed on to the next stage. In this latter stage, twins who want to participate are visited by a doctor/nurse team. Some of the examinations include physical and neuropsychological examination, laboratory testing, neuroimaging, psychological status assessment, and family history collection. In addition, data are collected on medical and environmental risk factors.

More information about Twin studies can be found at: http://www.mep.ki.se/twinreg/index_en.html

The latter part of the project is administered with a Web interface written using Perl and MySQL.

Each night all data from the interviews is moved into a MySQL database.

3.7.1. Find All Non-distributed Twins

The following query is used to determine who goes into the second part of the project:

```
SELECT
    CONCAT(p1.id, p1.tvab) + 0 AS tvid,
    CONCAT(p1.christian_name, ' ', p1.surname) AS Name,
    p1.postal_code AS Code,
    p1.city AS City,
    pg.abrev AS Area,
    IF(td.participation = 'Aborted', 'A', ' ') AS A,
    p1.dead AS dead1,
    l.event AS event1,
    td.suspect AS tsuspect1,
    id.suspect AS isuspect1,
    td.severe AS tsevere1,
    id.severe AS isevere1,
    p2.dead AS dead2,
    l2.event AS event2,
    h2.nurse AS nurse2,
    h2.doctor AS doctor2,
    td2.suspect AS tsuspect2,
    id2.suspect AS isuspect2,
    td2.severe AS tsevere2,
    id2.severe AS isevere2,
    l.finish_date
FROM
    twin_project AS tp
    /* For Twin 1 */
    LEFT JOIN twin_data AS td ON tp.id = td.id
              AND tp.tvab = td.tvab
    LEFT JOIN informant_data AS id ON tp.id = id.id
              AND tp.tvab = id.tvab
    LEFT JOIN harmony AS h ON tp.id = h.id
              AND tp.tvab = h.tvab
    LEFT JOIN lentus AS l ON tp.id = l.id
              AND tp.tvab = l.tvab
    /* For Twin 2 */
    LEFT JOIN twin_data AS td2 ON p2.id = td2.id
              AND p2.tvab = td2.tvab
    LEFT JOIN informant_data AS id2 ON p2.id = id2.id
              AND p2.tvab = id2.tvab
    LEFT JOIN harmony AS h2 ON p2.id = h2.id
              AND p2.tvab = h2.tvab
    LEFT JOIN lentus AS l2 ON p2.id = l2.id
              AND p2.tvab = l2.tvab,
    person_data AS p1,
    person_data AS p2,
    postal_groups AS pg
WHERE
```

```
    /* p1 gets main twin and p2 gets his/her twin. */
    /* ptvab is a field inverted from tvab */
    p1.id = tp.id AND p1.tvab = tp.tvab AND
    p2.id = p1.id AND p2.ptvab = p1.tvab AND
    /* Just the screening survey */
    tp.survey_no = 5 AND
    /* Skip if partner died before 65 but allow emigration (dead=9) */
    (p2.dead = 0 OR p2.dead = 9 OR
     (p2.dead = 1 AND
      (p2.death_date = 0 OR
       (((TO_DAYS(p2.death_date) - TO_DAYS(p2.birthday)) / 365)
        >= 65))))
    AND
    (
    /* Twin is suspect */
    (td.future_contact = 'Yes' AND td.suspect = 2) OR
    /* Twin is suspect - Informant is Blessed */
    (td.future_contact = 'Yes' AND td.suspect = 1
                               AND id.suspect = 1) OR
    /* No twin - Informant is Blessed */
    (ISNULL(td.suspect) AND id.suspect = 1
                        AND id.future_contact = 'Yes') OR
    /* Twin broken off - Informant is Blessed */
    (td.participation = 'Aborted'
     AND id.suspect = 1 AND id.future_contact = 'Yes') OR
    /* Twin broken off - No inform - Have partner */
    (td.participation = 'Aborted' AND ISNULL(id.suspect)
                                  AND p2.dead = 0))
    AND
    l.event = 'Finished'
    /* Get at area code */
    AND SUBSTRING(p1.postal_code, 1, 2) = pg.code
    /* Not already distributed */
    AND (h.nurse IS NULL OR h.nurse=00 OR h.doctor=00)
    /* Has not refused or been aborted */
    AND NOT (h.status = 'Refused' OR h.status = 'Aborted'
    OR h.status = 'Died' OR h.status = 'Other')
ORDER BY
    tvid;
```

Some explanations:

- `CONCAT(p1.id, p1.tvab) + 0 AS tvid`

  We want to sort on the concatenated `id` and `tvab` in numerical order. Adding `0` to the result causes MySQL to treat the result as a number.

- column `id`

  This identifies a pair of twins. It is a key in all tables.

- column `tvab`

  This identifies a twin in a pair. It has a value of `1` or `2`.

- column `ptvab`

  This is an inverse of `tvab`. When `tvab` is `1` this is `2`, and vice versa. It exists to save typing and to make it easier for MySQL to optimize the query.

This query demonstrates, among other things, how to do lookups on a table from the same table with a join (`p1` and `p2`). In the example, this is used to check whether a twin's partner died before the age of 65. If so, the row is not returned.

All of the above exist in all tables with twin-related information. We have a key on both `id,tvab` (all tables), and `id,ptvab` (`person_data`) to make queries faster.

On our production machine (A 200MHz UltraSPARC), this query returns about 150-200 rows and takes less than one second.

The current number of records in the tables used in the query:

Table	Rows
`person_data`	71074
`lentus`	5291
`twin_project`	5286
`twin_data`	2012
`informant_data`	663
`harmony`	381
`postal_groups`	100

3.7.2. Show a Table of Twin Pair Status

Each interview ends with a status code called `event`. The query shown here is used to display a table over all twin pairs combined by event. This indicates in how many pairs both twins are finished, in how many pairs one twin is finished and the other refused, and so on.

```
SELECT
        t1.event,
        t2.event,
        COUNT(*)
FROM
        lentus AS t1,
        lentus AS t2,
        twin_project AS tp
WHERE
        /* We are looking at one pair at a time */
        t1.id = tp.id
        AND t1.tvab=tp.tvab
        AND t1.id = t2.id
        /* Just the screening survey */
        AND tp.survey_no = 5
        /* This makes each pair only appear once */
        AND t1.tvab='1' AND t2.tvab='2'
GROUP BY
        t1.event, t2.event;
```

3.8. Using MySQL with Apache

There are programs that let you authenticate your users from a MySQL database and also let you write your log files into a MySQL table.

You can change the Apache logging format to be easily readable by MySQL by putting the following in-

to the Apache configuration file:

```
LogFormat \
        "\"%h\",%{%Y%m%d%H%M%S}t,%>s,\"%b\",\"%{Content-Type}o\",  \
        \"%U\",\"%{Referer}i\",\"%{User-Agent}i\""
```

To load a log file in that format into MySQL, you can use a statement something like this:

```
LOAD DATA INFILE '/local/access_log' INTO TABLE tbl_name
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
```

The named table should be created to have columns that correspond to those that the `LogFormat` line writes to the log file.

Chapter 4. Using MySQL Programs

This chapter provides a brief overview of the programs provided by MySQL AB and discusses how to specify options when you run these programs. Most programs have options that are specific to their own operation, but the syntax for specifying options is similar for all of them. Later chapters provide more detailed descriptions of individual programs, including which options they recognize.

4.1. Overview of MySQL Programs

MySQL AB provides several types of programs:

- The MYSQL server and server startup scripts:
  - **mysqld** is the MySQL server
  - **mysqld_safe**, **mysql.server**, and **mysqld_multi** are server startup scripts
  - **mysql_install_db** initializes the data directory and the initial databases

  These programs are discussed further in Chapter 5, *Database Administration*.
- Client programs that access the server:
  - **mysql** is a command-line client for executing SQL statements interactively or in batch mode
  - **mysqlcc** (MySQL Control Center) is an interactive graphical tool for executing SQL statements and administration
  - **mysqladmin** is an administrative client
  - **mysqlcheck** performs table maintenance operations
  - **mysqldump** and **mysqlhotcopy** make database backups
  - **mysqlimport** imports data files
  - **mysqlshow** displays information about databases and tables

  These programs are discussed further in Chapter 8, *MySQL Client and Utility Programs*.
- Utility programs that operate independently of the server:
  - **myisamchk** performs table maintenance operations
  - **myisampack** produces compressed, read-only tables
  - **mysqlbinlog** is a tool for processing binary log files
  - **perror** displays error code meanings

myisamchk is discussed further in Chapter 5, *Database Administration*. The other programs are further in Chapter 8, *MySQL Client and Utility Programs*.

Most MySQL distributions include all of these programs, except for those programs that are platform-specific. (For example, the server startup scripts are not used on Windows.) The exception is that RPM distributions are more specialized. There is one RPM for the server, another for the client programs, and so forth. If you appear to be missing one or more programs, see Chapter 2, *Installing MySQL* for information on types of distributions and what they contain. It may be that you need to install something else.

4.2. Invoking MySQL Programs

To invoke a MySQL program at the command line (that is, from your shell or command prompt), enter the program name followed by any options or other arguments needed to instruct the program what you want it to do. The following commands show some sample program invocations. ``shell>'' represents the prompt for your command interpreter; it is not part of what you type. The particular prompt you see depends on your command interpreter. Typical prompts are `$` for **sh** or **bash**, `%` for **csh** or **tcsh**, and `C:\>` for Windows **command.com** or **cmd.exe**.

```
shell> mysql test
shell> mysqladmin extended-status variables
shell> mysqlshow --help
shell> mysqldump --user=root personnel
```

Arguments that begin with a dash are option arguments. They typically specify the type of connection a program should make to the server or affect its operational mode. Options have a syntax that is described in Section 4.3, “Specifying Program Options”.

Non-option arguments (arguments with no leading dash) provide additional information to the program. For example, the **mysql** program interprets the first non-option argument as a database name, so the command `mysql test` indicates that you want to use the `test` database.

Later sections that describe individual programs indicate which options a program understands and describe the meaning of any additional non-option arguments.

Some options are common to a number of programs. The most common of these are the `--host`, `--user`, and `--password` options that specify connection parameters. They indicate the host where the MySQL server is running, and the username and password of your MySQL account. All MySQL client programs understand these options; they allow you to specify which server to connect to and the account to use on that server.

You may find it necessary to invoke MySQL programs using the pathname to the `bin` directory in which they are installed. This is likely to be the case if you get a ``program not found'' error whenever you attempt to run a MySQL program from any directory other than the `bin` directory. To make it more convenient to use MySQL, you can add the pathname of the `bin` directory to your `PATH` environment variable setting. Then to run a program you need only type its name, not its entire pathname.

Consult the documentation for your command interpreter for instructions on setting your `PATH`. The syntax for setting environment variables is interpreter-specific.

4.3. Specifying Program Options

You can provide options for MySQL programs in several ways:

- On the command line following the program name. This is most common for options that apply to a specific invocation of the program.
- In an option file that the program reads when it starts. This is common for options that you want the program to use each time it runs.
- In environment variables. These are useful for options that you want to apply each time the program runs, although in practice option files are used more commonly for this purpose. (Section 5.10.2, “Running Multiple Servers on Unix” discusses one situation in which environment variables can be very helpful. It describes a handy technique that uses such variables to specify the TCP/IP port number and Unix socket file for both the server and client programs.)

MySQL programs determine which options are given first by examining environment variables, then option files, and then the command line. If an option is specified multiple times, the last occurrence takes precedence. This means that environment variables have the lowest precedence and command-line options the highest.

You can take advantage of the way that MySQL programs process options by specifying the default values for a program's options in an option file. Then you need not type them each time you run the program, but can override the defaults if necessary by using command-line options.

4.3.1. Using Options on the Command Line

Program options specified on the command line follow these rules:

- Options are given after the command name.
- An option argument begins with one dash or two dashes, depending on whether it has a short name or a long name. Many options have both forms. For example, `-?` and `--help` are the short and long forms of the option that instructs a MySQL program to display a help message.
- Option names are case sensitive. `-v` and `-V` are both legal and have different meanings. (They are the corresponding short forms of the `--verbose` and `--version` options.)
- Some options take a value following the option name. For example, `-h localhost` or `--host=localhost` indicate the MySQL server host to a client program. The option value tells the program the name of the host where the MySQL server is running.
- For a long option that takes a value, separate the option name and the value by an '`=`' sign. For a short option that takes a value, the option value can immediately follow the option letter, or there can be a space between. (`-hlocalhost` and `-h localhost` are equivalent.) An exception to this rule is the option for specifying your MySQL password. This option can be given in long form as `--password=pass_val` or as `--password`. In the latter case (with no password value given), the program prompts you for the password. The password option also may be given in short form as `-ppass_val` or as `-p`. However, for the short form, if the password value is given, it must follow the option letter with *no intervening space*. The reason for this is that if a space follows the option letter, the program has no way to tell whether a following argument is supposed to be the password value or some other kind of argument. Consequently, the following two commands have two completely different meanings:

  ```
  shell> mysql -ptest
  shell> mysql -p test
  ```

  The first command instructs **mysql** to use a password value of `test`, but specifies no default database. The second instructs **mysql** to prompt for the password value and to use `test` as the default

database.

MySQL 4.0 introduced some additional flexibility in the way you specify options. These changes were made in MySQL 4.0.2. Some of them relate to the way you specify options that have ``enabled'' and ``disabled'' states, and to the use of options that might be present in one version of MySQL but not another. Those capabilities are discussed in this section. Another change pertains to the way you use options to set program variables. Section 4.3.4, “Using Options to Set Program Variables” discusses that topic further.

Some options control behavior that can be turned on or off. For example, the **mysql** client supports a `--column-names` option that determines whether or not to display a row of column names at the beginning of query results. By default, this option is enabled. However, you may want to disable it in some instances, such as when sending the output of **mysql** into another program that expects to see only data and not an initial header line.

To disable column names, you can specify the option using any of these forms:

```
--disable-column-names
--skip-column-names
--column-names=0
```

The `--disable` and `--skip` prefixes and the `=0` suffix all have the same effect: They turn the option off.

The ``enabled'' form of the option may be specified in any of these ways:

```
--column-names
--enable-column-names
--column-names=1
```

Another change to option processing introduced in MySQL 4.0 is that you can use the `--loose` prefix for command-line options. If an option is prefixed by `--loose`, the program does not exit with an error if it does not recognize the option, but instead issues only a warning:

```
shell> mysql --loose-no-such-option
mysql: WARNING: unknown option '--no-such-option'
```

The `--loose` prefix can be useful when you run programs from multiple installations of MySQL on the same machine, at least if all the versions are as recent as 4.0.2. This prefix is particularly useful when you list options in an option file. An option that may not be recognized by all versions of a program can be given using the `--loose` prefix (or `loose` in an option file). Versions of the program that do not recognize the option issue a warning and ignore it. This strategy requires that versions involved be 4.0.2 or later, because earlier versions know nothing of the `--loose` convention.

4.3.2. Using Option Files

MySQL programs can read startup options from option files (also sometimes called configuration files). Option files provide a convenient way to specify commonly used options so that they need not be entered on the command line each time you run a program. Option file capability is available from MySQL 3.22 on.

The following programs support option files: **myisamchk**, **myisampack**, **mysql**, **mysql.server**, **mysqladmin**, **mysqlbinlog**, **mysqlcc**, **mysqlcheck**, **mysqld_safe**, **mysqldump**, **mysqld**, **mysqlhotcopy**, **mysqlimport**, and **mysqlshow**.

On Windows, MySQL programs read startup options from the following files:

Filename	Purpose
WINDIR\my.ini	Global options
C:\my.cnf	Global options
INSTALLDIR\my.ini	Global Options

WINDIR represents the location of your Windows directory. This is commonly C:\Windows or C:\WinNT. You can determine its exact location from the value of the WINDIR environment variable using the following command:

```
C:\> echo %WINDIR%
```

INSTALLDIR represents the installation directory of MySQL. This is typically the case with MySQL 4.1.5 and higher, when installed using the installation and configuration wizards. See Section 2.3.5.14, “The Location of the my.ini File”.

On Unix, MySQL programs read startup options from the following files:

Filename	Purpose
/etc/my.cnf	Global options
$MYSQL_HOME/my.cnf	Server-specific options
defaults-extra-file	The file specified with --defaults-extra-file=*path*, if any
~/.my.cnf	User-specific options

MYSQL_HOME is an environment variable containing the path to the directory in which the server-specific my.cnf file resides. This used to be *DATADIR* prior to MySQL version 5.0.3.

If *MYSQL_HOME* is not set and there is a my.cnf file in *DATADIR* and there is no my.cnf file in *BASEDIR*, mysqld_safe sets *MYSQL_HOME* to *DATADIR*. Otherwise, if *MYSQL_HOME* is not set and there is no my.cnf in *DATADIR*, then mysqld_safe sets *MYSQL_HOME* to *BASEDIR*.

Typically this is /usr/local/mysql/data for a binary installation or /usr/local/var for a source installation. Note that this is the data directory location that was specified at configuration time, not the one specified with --datadir when **mysqld** starts. Use of --datadir at runtime has no effect on where the server looks for option files, because it looks for them before processing any command-line arguments.

MySQL looks for option files in the order just described and reads any that exist. If an option file that you want to use does not exist, create it with a plain text editor. If multiple option files exist, an option specified in a file read later takes precedence over the same option specified in a file read earlier.

Any long option that may be given on the command line when running a MySQL program can be given in an option file as well. To get the list of available options for a program, run it with the --help option.

The syntax for specifying options in an option file is similar to command-line syntax, except that you omit the leading two dashes. For example, --quick or --host=localhost on the command line should be specified as quick or host=localhost in an option file. To specify an option of the form --loose-opt_name in an option file, write it as loose-opt_name.

Empty lines in option files are ignored. Non-empty lines can take any of the following forms:

- *#comment* , *;comment*

  Comment lines start with '#' or ';'. As of MySQL 4.0.14, a '#'-comment can start in the middle of a line as well.

- [*group*]

  group is the name of the program or group for which you want to set options. After a group line, any *opt_name* or set-variable lines apply to the named group until the end of the option file or another group line is given.

- *opt_name*

  This is equivalent to --*opt_name* on the command line.

- *opt_name=value*

  This is equivalent to --*opt_name=value* on the command line. In an option file, you can have spaces around the '=' character, something that is not true on the command line. As of MySQL 4.0.16, you can quote the value with double quotes or single quotes. This is useful if the value contains a '#' comment character or whitespace.

- set-variable = *var_name=value*

  Set the program variable *var_name* to the given value. This is equivalent to --set-variable=*var_name=value* on the command line. Spaces are allowed around the first '=' character but not around the second. This syntax is deprecated as of MySQL 4.0. See Section 4.3.4, “Using Options to Set Program Variables” for more information on setting program variables.

Leading and trailing blanks are automatically deleted from option names and values. You may use the escape sequences '\b', '\t', '\n', '\r', '\\', and '\s' in option values to represent the backspace, tab, newline, carriage return, and space characters.

On Windows, if an option value represents a pathname, you should specify the value using '/' rather than '\' as the pathname separator. If you use '\', you must double it as '\\', because '\' is the escape character in MySQL.

If an option group name is the same as a program name, options in the group apply specifically to that program.

The [client] option group is read by all client programs (but not by **mysqld**). This allows you to specify options that apply to every client. For example, [client] is the perfect group to use to specify the password that you use to connect to the server. (But make sure that the option file is readable and writable only by yourself, so that other people cannot find out your password.) Be sure not to put an option in the [client] group unless it is recognized by *all* client programs that you use. Programs that do not understand the option quit after displaying an error message if you try to run them.

As of MySQL 4.0.14, if you want to create option groups that should be read only by one specific **mysqld** server release series, you can do this by using groups with names of [mysqld-4.0], [mysqld-4.1], and so forth. The following group indicates that the --new option should be used only by MySQL servers with 4.0.x version numbers:

```
[mysqld-4.0]
new
```

Here is a typical global option file:

```
[client]
port=3306
socket=/tmp/mysql.sock

[mysqld]
port=3306
socket=/tmp/mysql.sock
key_buffer_size=16M
max_allowed_packet=8M

[mysqldump]
quick
```

The preceding option file uses *var_name=value* syntax for the lines that set the `key_buffer_size` and `max_allowed_packet` variables. Prior to MySQL 4.0.2, you would need to use `set-variable` syntax instead (described earlier in this section).

Here is a typical user option file:

```
[client]
# The following password will be sent to all standard MySQL clients
password="my_password"

[mysql]
no-auto-rehash
set-variable = connect_timeout=2

[mysqlhotcopy]
interactive-timeout
```

This option file uses `set-variable` syntax to set the `connect_timeout` variable. For MySQL 4.0.2 and up, you can also set the variable using just `connect_timeout=2`.

If you have a source distribution, you can find sample option files named `my-xxxx.cnf` in the `support-files` directory. If you have a binary distribution, look in the `support-files` directory under your MySQL installation directory (typically `C:\mysql` on Windows or `/usr/local/mysql` on Unix). On Windows the sample option files may also be located in the MySQL installation directory. Currently there are sample option files for small, medium, large, and very large systems. To experiment with one of these files, copy it to `C:\my.cnf` on Windows or to `.my.cnf` in your home directory on Unix.

Note: On Windows, the `.cnf` option file extension might not be displayed.

All MySQL programs that support option files handle the following command-line options:

- `--no-defaults`

  Don't read any option files.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

- `--defaults-file=`*path_name*

  Use only the given option file. *path_name* is the full pathname to the file.

- `--defaults-extra-file=path_name`

  Read this option file after the global option file but before the user option file. *`path_name`* is the full pathname to the file.

To work properly, each of these options must immediately follow the command name on the command line, with the exception that `--print-defaults` may be used immediately after `--defaults-file` or `--defaults-extra-file`.

In shell scripts, you can use the **my_print_defaults** program to parse option files. The following example shows the output that **my_print_defaults** might produce when asked to show the options found in the `[client]` and `[mysql]` groups:

```
shell> my_print_defaults client mysql
--port=3306
--socket=/tmp/mysql.sock
--no-auto-rehash
```

Note for developers: Option file handling is implemented in the C client library simply by processing all matching options (that is, options in the appropriate group) before any command-line arguments. This works nicely for programs that use the last instance of an option that is specified multiple times. If you have a C or C++ program that handles multiply specified options this way but doesn't read option files, you need add only two lines to give it that capability. Check the source code of any of the standard MySQL clients to see how to do this.

Several other language interfaces to MySQL are based on the C client library, and some of them provide a way to access option file contents. These include Perl and Python. See the documentation for your preferred interface for details.

4.3.3. Using Environment Variables to Specify Options

To specify an option using an environment variable, set the variable using the syntax appropriate for your comment processor. For example, on Windows or NetWare, you can set the `USER` variable to specify your MySQL account name. To do so, use this syntax:

```
SET USER=your_name
```

The syntax on Unix depends on your shell. Suppose that you want to specify the TCP/IP port number using the `MYSQL_TCP_PORT` variable. Typical syntax (such as for **sh**, `bash`, **zsh**, etc.) is as follows:

```
MYSQL_TCP_PORT=3306
export MYSQL_TCP_PORT
```

The first command sets the variable, and the `export` command exports the variable to the shell environment so that its value becomes accessible to MySQL and other processes.

For **csh** and **tcsh** there are similar issues. When running these shells, use **setenv** to make the shell variable available to the environment:

```
setenv MYSQL_TCP_PORT 3306
```

The commands to set environment variables can be executed at your command prompt to take effect immediately. These settings persist until you log out. To have the settings take effect each time you log in, place the appropriate command or commands in a startup file that your command interpreter reads each

time it starts. Typical startup files are `AUTOEXEC.BAT` for Windows, `.bash_profile` for **bash**, or `.tcshrc` for **tcsh**. Consult the documentation for your command interpreter for specific details.

Appendix F, *Environment Variables* lists all environment variables that affect MySQL program operation.

4.3.4. Using Options to Set Program Variables

Many MySQL programs have internal variables that can be set at runtime. As of MySQL 4.0.2, program variables are set the same way as any other long option that takes a value. For example, **mysql** has a `max_allowed_packet` variable that controls the maximum size of its communication buffer. To set the `max_allowed_packet` variable for **mysql** to a value of 16MB, use either of the following commands:

```
shell> mysql --max_allowed_packet=16777216
shell> mysql --max_allowed_packet=16M
```

The first command specifies the value in bytes. The second specifies the value in megabytes. Variable values can have a suffix of `K`, `M`, or `G` (either uppercase or lowercase) to indicate units of kilobytes, megabytes, or gigabytes.

In an option file, the variable setting is given without the leading dashes:

```
[mysql]
max_allowed_packet=16777216
```

Or:

```
[mysql]
max_allowed_packet=16M
```

If you like, underscores in a variable name can be specified as dashes.

Prior to MySQL 4.0.2, program variable names are not recognized as option names. Instead, use the `--set-variable` option to assign a value to a variable:

```
shell> mysql --set-variable=max_allowed_packet=16777216
shell> mysql --set-variable=max_allowed_packet=16M
```

In an option file, omit the leading dashes:

```
[mysql]
set-variable = max_allowed_packet=16777216
```

Or:

```
[mysql]
set-variable = max_allowed_packet=16M
```

With `--set-variable`, underscores in variable names cannot be given as dashes for versions of MySQL older than 4.0.2.

The `--set-variable` option is still recognized in MySQL 4.0.2 and up, but is deprecated.

Some server variables can be set at runtime. For details, see Section 5.2.3.1, “Dynamic System Variables”.

Chapter 5. Database Administration

This chapter covers topics that deal with administering a MySQL installation, such as configuring the server, managing user accounts, and performing backups.

5.1. The MySQL Server and Server Startup Scripts

The MySQL server, **mysqld**, is the main program that does most of the work in a MySQL installation. The server is accompanied by several related scripts that perform setup operations when you install MySQL or that are helper programs to assist you in starting and stopping the server.

This section provides an overview of the server and related programs, and information about server startup scripts. Information about configuring the server itself is given in Section 5.2, “Configuring the MySQL Server”.

5.1.1. Overview of the Server-Side Scripts and Utilities

All MySQL programs take many different options. However, every MySQL program provides a `--help` option that you can use to get a description of the program's options. For example, try **mysqld --help**.

You can override default options for all standard programs by specifying options on the command line or in an option file. Section 4.3, “Specifying Program Options”.

The following list briefly describes the MySQL server and server-related programs:

- `mysqld`

  The SQL daemon (that is, the MySQL server). To use client programs, this program must be running, because clients gain access to databases by connecting to the server. See Section 5.2, “Configuring the MySQL Server”.

- `mysqld-max`

  A version of the server that includes additional features. See Section 5.1.2, “The **mysqld-max** Extended MySQL Server”.

- `mysqld_safe`

  A server startup script. **mysqld_safe** attempts to start **mysqld-max** if it exists, and **mysqld** otherwise. See Section 5.1.3, “The **mysqld_safe** Server Startup Script”.

- `mysql.server`

  A server startup script. This script is used on systems that use run directories containing scripts that start system services for particular run levels. It invokes **mysqld_safe** to start the MySQL server. See Section 5.1.4, “The **mysql.server** Server Startup Script”.

- `mysqld_multi`

A server startup script that can start or stop multiple servers installed on the system. See Section 5.1.5, “The **mysqld_multi** Program for Managing Multiple MySQL Servers”.

- `mysql_install_db`

  This script creates the MySQL grant tables with default privileges. It is usually executed only once, when first installing MySQL on a system.

- `mysql_fix_privilege_tables`

  This script is used after an upgrade install operation, to update the grant tables with any changes that have been made in newer versions of MySQL.

There are several other programs that also are run on the server host:

- `myisamchk`

  A utility to describe, check, optimize, and repair `MyISAM` tables. **myisamchk** is described in Section 5.7.3, “Table Maintenance and Crash Recovery”.

- `make_binary_distribution`

  This program makes a binary release of a compiled MySQL. This could be sent by FTP to `/pub/mysql/upload/` on `ftp.mysql.com` for the convenience of other MySQL users.

- `mysqlbug`

  The MySQL bug reporting script. It can be used to send a bug report to the MySQL mailing list. (You can also visit http://bugs.mysql.com/ to file a bug report online.)

5.1.2. The mysqld-max Extended MySQL Server

A MySQL-Max server is a version of the **mysqld** MySQL server that has been built to include additional features.

The distribution to use depends on your platform:

- For Windows, MySQL binary distributions include both the standard server (`mysqld.exe`) and the MySQL-Max server (**mysqld-max.exe**), so you need not get a special distribution. Just use a regular Windows distribution, available at http://dev.mysql.com/downloads/. See Section 2.3, “Installing MySQL on Windows”.

- For Linux, if you install MySQL using RPM distributions, use the regular `MySQL-server` RPM first to install a standard server named **mysqld**. Then use the `MySQL-Max` RPM to install a server named **mysqld-max**. The `MySQL-Max` RPM presupposes that you have installed the regular server RPM. See Section 2.4, “Installing MySQL on Linux” for more information on the Linux RPM packages.

- All other MySQL-Max distributions contain a single server that is named **mysqld** but that has the additional features included.

You can find the MySQL-Max binaries on the MySQL AB Web site at http://dev.mysql.com/downloads/mysql-4.0.html.

MySQL AB builds the MySQL-Max servers by using the following **configure** options:

- `--with-server-suffix=-max`

  This option adds a `-max` suffix to the **mysqld** version string.

- `--with-innodb`

  This option enables support for the InnoDB storage engine. MySQL-Max servers always include InnoDB support, but this option actually is needed only for MySQL 3.23. From MySQL 4 onwards, InnoDB is included by default in binary distributions, so you do not need a MySQL-Max server merely to obtain InnoDB support.

- `--with-bdb`

  This option enables support for the Berkeley DB (BDB) storage engine.

- `USE_SYMDIR`

  This define is enabled to turn on database symbolic link support for Windows. (This applies only before MySQL 4.0. As of MySQL 4.0, symbolic link support is available for all Windows servers, so a Max server is not needed to take advantage of this feature.)

- `--with-ndb-cluster`

  This option enables support for the NDB Cluster storage engine in MySQL 4.1.2 and newer. Note that as of MySQL 5.0.3, CLuster is supported on Linux, Solaris, and Mac OS X only.

MySQL-Max binary distributions are a convenience for those who wish to install precompiled programs. If you build MySQL using a source distribution, you can build your own Max-like server by enabling the same features at configuration time that the MySQL-Max binary distributions are built with.

MySQL-Max servers include the BerkeleyDB (BDB) storage engine whenever possible, but not all platforms support BDB.

MySQL-Max servers for Solaris, Mac OS X, and Linux (on most platforms) include support for the NDB Cluster storage engine. Note that the server must be started with the `ndbcluster` option in order to run the server as part of a MySQL Cluster. (For details, see Section 16.5, “MySQL Cluster Configuration”.)

The following table shows on which platforms allow MySQL-Max binaries include support for BDB and/or NDB Cluster:

System	BDB Support	NDB Support
AIX 4.3	N	N
HP-UX 11.0	N	N
Linux-Alpha	N	Y
Linux-IA-64	N	N
Linux-Intel	Y	Y
Mac OS X	N	N
NetWare	N	N

SCO OSR5	Y	N
Solaris-SPARC	Y	Y
Solaris-Intel	N	Y
UnixWare	Y	N
Windows NT/2000/XP	Y	N

To find out which storage engines your server supports, issue the following statement:

```
mysql> SHOW ENGINES;
+------------+---------+------------------------------------------------------------
| Engine     | Support | Comment
+------------+---------+------------------------------------------------------------
| MyISAM     | DEFAULT | Default engine as of MySQL 3.23 with great performance
| HEAP       | YES     | Alias for MEMORY
| MEMORY     | YES     | Hash based, stored in memory, useful for temporary tables
| MERGE      | YES     | Collection of identical MyISAM tables
| MRG_MYISAM | YES     | Alias for MERGE
| ISAM       | NO      | Obsolete storage engine, now replaced by MyISAM
| MRG_ISAM   | NO      | Obsolete storage engine, now replaced by MERGE
| InnoDB     | YES     | Supports transactions, row-level locking, and foreign key
| INNOBASE   | YES     | Alias for INNODB
| BDB        | YES     | Supports transactions and page-level locking
| BERKELEYDB | YES     | Alias for BDB
| NDBCLUSTER | NO      | Clustered, fault-tolerant, memory-based tables
| NDB        | NO      | Alias for NDBCLUSTER
| EXAMPLE    | NO      | Example storage engine
| ARCHIVE    | NO      | Archive storage engine
| CSV        | NO      | CSV storage engine
+------------+---------+------------------------------------------------------------
16 rows in set (0.02 sec)
```

(See also Section 13.5.4.8, “`SHOW ENGINES` Syntax”.)

Before MySQL 4.1.2, `SHOW ENGINES` is unavailable. Use the following statement instead and check the value of the variable for the storage engine in which you are interested:

```
mysql> SHOW VARIABLES LIKE 'have_%';
+---------------------+-------+
| Variable_name       | Value |
+---------------------+-------+
| have_archive        | NO    |
| have_bdb            | YES   |
| have_compress       | YES   |
| have_crypt          | NO    |
| have_csv            | NO    |
| have_example_engine | NO    |
| have_geometry       | YES   |
| have_innodb         | YES   |
| have_isam           | NO    |
| have_ndbcluster     | NO    |
| have_openssl        | NO    |
| have_query_cache    | YES   |
| have_raid           | NO    |
| have_rtree_keys     | YES   |
| have_symlink        | YES   |
+---------------------+-------+
15 rows in set (0.15 sec)
```

The precise output from these SHOW commands will vary according to the MySQL version used (and the features which are enabled). The values in the second column indicate the server's level of support for each feature, as shown here:

Value	Meaning
YES	The feature is supported and is active.
NO	The feature is not supported.
DISABLED	The feature is supported but has been disabled.

A value of NO means that the server was compiled without support for the feature, so it cannot be activated at runtime.

A value of DISABLED occurs either because the server was started with an option that disables the feature, or because not all options required to enable it were given. In the latter case, the *host_name*.err error log file should contain a reason indicating why the option is disabled.

One situation in which you might see DISABLED occurs with MySQL 3.23 when the InnoDB storage engine is compiled in. In MySQL 3.23, you must supply at least the innodb_data_file_path option at runtime to set up the InnoDB tablespace. Without this option, InnoDB disables itself. See Section 15.3, "InnoDB in MySQL 3.23". You can specify configuration options for the BDB storage engine, too, but BDB does not disable itself if you do not provide them. See Section 14.4.3, "BDB Startup Options".

You might also see DISABLED for the InnoDB, BDB, or ISAM storage engines if the server was compiled to support them, but was started with the --skip-innodb, --skip-bdb, or --skip-isam options at runtime.

As of Version 3.23, all MySQL servers support MyISAM tables, because MyISAM is the default storage engine.

5.1.3. The mysqld_safe Server Startup Script

mysqld_safe is the recommended way to start a **mysqld** server on Unix and NetWare. **mysqld_safe** adds some safety features such as restarting the server when an error occurs and logging runtime information to an error log file. NetWare-specific behaviors are listed later in this section.

Note: Before MySQL 4.0, **mysqld_safe** is named **safe_mysqld**. To preserve backward compatibility, MySQL binary distributions for some time will include **safe_mysqld** as a symbolic link to **mysqld_safe**.

By default, **mysqld_safe** tries to start an executable named **mysqld-max** if it exists, or **mysqld** otherwise. Be aware of the implications of this behavior:

- On Linux, the MySQL-Max RPM relies on this **mysqld_safe** behavior. The RPM installs an executable named **mysqld-max**, which causes **mysqld_safe** to automatically use that executable from that point on.
- If you install a MySQL-Max distribution that includes a server named **mysqld-max**, then upgrade later to a non-Max version of MySQL, **mysqld_safe** still attempts to run the old **mysqld-max** server. If you perform such an upgrade, you should manually remove the old **mysqld-max** server to ensure that **mysqld_safe** runs the new **mysqld** server.

To override the default behavior and specify explicitly which server you want to run, specify a --mysqld or --mysqld-version option to **mysqld_safe**.

Many of the options to **mysqld_safe** are the same as the options to **mysqld**. See Section 5.2.1, “**mysqld** Command-Line Options”.

All options specified to **mysqld_safe** on the command line are passed to **mysqld**. If you want to use any options that are specific to **mysqld_safe** and that **mysqld** doesn't support, do not specify them on the command line. Instead, list them in the `[mysqld_safe]` group of an option file. See Section 4.3.2, “Using Option Files”.

mysqld_safe reads all options from the `[mysqld]`, `[server]`, and `[mysqld_safe]` sections in option files. For backward compatibility, it also reads `[safe_mysqld]` sections, although you should rename such sections to `[mysqld_safe]` when you begin using MySQL 4.0 or later.

mysqld_safe supports the following options:

- `--help`

  Display a help message and exit. (New in 5.0.3)

- `--basedir=`*`path`*

  The path to the MySQL installation directory.

- `--core-file-size=`*`size`*

  The size of the core file **mysqld** should be able to create. The option value is passed to **ulimit -c**.

- `--datadir=`*`path`*

  The path to the data directory.

- `--defaults-extra-file=`*`path`*

  The name of an option file to be read in addition to the usual option files.

- `--defaults-file=`*`path`*

  The name of an option file to be read instead of the usual option files.

- `--err-log=`*`path`*

  The old form of the `--log-error` option, to be used before MySQL 4.0.

- `--ledir=`*`path`*

  The path to the directory containing the **mysqld** program. Use this option to explicitly indicate the location of the server.

- `--log-error=`*`path`*

  Write the error log to the given file. See Section 5.9.1, “The Error Log”.

- `--mysqld=`*`prog_name`*

  The name of the server program (in the `ledir` directory) that you want to start. This option is needed if you use the MySQL binary distribution but have the data directory outside of the binary distribution.

- `--mysqld-version=`*`suffix`*

This option is similar to the `--mysqld` option, but you specify only the suffix for the server program name. The basename is assumed to be **mysqld**. For example, if you use `--mysqld-version=max`, **mysqld_safe** starts the **mysqld-max** program in the `ledir` directory. If the argument to `--mysqld-version` is empty, **mysqld_safe** uses **mysqld** in the `ledir` directory.

- `--nice=priority`

  Use the `nice` program to set the server's scheduling priority to the given value. This option was added in MySQL 4.0.14.

- `--no-defaults`

  Do not read any option files.

- `--open-files-limit=count`

  The number of files **mysqld** should be able to open. The option value is passed to **ulimit -n**. Note that you need to start **mysqld_safe** as `root` for this to work properly!

- `--pid-file=path`

  The path to the process ID file.

- `--port=port_num`

  The port number to use when listening for TCP/IP connections.

- `--socket=path`

  The Unix socket file to use for local connections.

- `--timezone=zone`

  Set the `TZ` time zone environment variable to the given option value. Consult your operating system documentation for legal time zone specification formats.

- `--user={user_name | user_id}`

  Run the **mysqld** server as the user having the name `user_name` or the numeric user ID `user_id`. (``User'' in this context refers to a system login account, not a MySQL user listed in the grant tables.)

The **mysqld_safe** script is written so that it normally can start a server that was installed from either a source or a binary distribution of MySQL, even though these types of distributions typically install the server in slightly different locations. (See Section 2.1.5, “Installation Layouts”.) **mysqld_safe** expects one of the following conditions to be true:

- The server and databases can be found relative to the directory from which **mysqld_safe** is invoked. For binary distributions, **mysqld_safe** looks under its working directory for `bin` and `data` directories. For source distributions, it looks for `libexec` and `var` directories. This condition should be met if you execute **mysqld_safe** from your MySQL installation directory (for example, `/usr/local/mysql` for a binary distribution).

- If the server and databases cannot be found relative to the working directory, **mysqld_safe** attempts to locate them by absolute pathnames. Typical locations are `/usr/local/libexec` and `/usr/local/var`. The actual locations are determined from the values configured into the distri-

bution at the time it was built. They should be correct if MySQL is installed in the location specified at configuration time.

Because **mysqld_safe** tries to find the server and databases relative to its own working directory, you can install a binary distribution of MySQL anywhere, as long as you run **mysqld_safe** from the MySQL installation directory:

```
shell> cd mysql_installation_directory
shell> bin/mysqld_safe &
```

If **mysqld_safe** fails, even when invoked from the MySQL installation directory, you can specify the `--ledir` and `--datadir` options to indicate the directories in which the server and databases are located on your system.

Normally, you should not edit the **mysqld_safe** script. Instead, configure **mysqld_safe** by using command-line options or options in the `[mysqld_safe]` section of a `my.cnf` option file. In rare cases, it might be necessary to edit **mysqld_safe** to get it to start the server properly. However, if you do this, your modified version of **mysqld_safe** might be overwritten if you upgrade MySQL in the future, so you should make a copy of your edited version that you can reinstall.

On NetWare, **mysqld_safe** is a NetWare Loadable Module (NLM) that is ported from the original Unix shell script. It does the following:

1. Runs a number of system and option checks.
2. Runs a check on `MyISAM` and `ISAM` tables.
3. Provides a screen presence for the MySQL server.
4. Starts **mysqld**, monitors it, and restarts it if it terminates in error.
5. Sends error messages from **mysqld** to the *`host_name`*`.err` file in the data directory.
6. Sends **mysqld_safe** screen output to the *`host_name`*`.safe` file in the data directory.

5.1.4. The mysql.server Server Startup Script

MySQL distributions on Unix include a script named **mysql.server**. It can be used on systems such as Linux and Solaris that use System V-style run directories to start and stop system services. It is also used by the Mac OS X Startup Item for MySQL.

mysql.server can be found in the `support-files` directory under your MySQL installation directory or in a MySQL source tree.

If you use the Linux server RPM package (`MySQL-server-`*`VERSION`*`.rpm`), the **mysql.server** script will be installed in the `/etc/init.d` directory with the name `mysql`. You need not install it manually. See Section 2.4, “Installing MySQL on Linux” for more information on the Linux RPM packages.

Some vendors provide RPM packages that install a startup script under a different name such as **mysqld**.

If you install MySQL from a source distribution or using a binary distribution format that does not install **mysql.server** automatically, you can install it manually. Instructions are provided in Section 2.9.2.2, “Starting and Stopping MySQL Automatically”.

mysql.server reads options from the `[mysql.server]` and `[mysqld]` sections of option files. (For

backward compatibility, it also reads `[mysql_server]` sections, although you should rename such sections to `[mysql.server]` when you begin using MySQL 4.0 or later.)

5.1.5. The mysqld_multi Program for Managing Multiple MySQL Servers

mysqld_multi is meant for managing several **mysqld** processes that listen for connections on different Unix socket files and TCP/IP ports. It can start or stop servers, or report their current status.

The program searches for groups named `[mysqld#]` in `my.cnf` (or in the file named by the `--config-file` option). # can be any positive integer. This number is referred to in the following discussion as the option group number, or *GNR*. Group numbers distinguish option groups from one another and are used as arguments to **mysqld_multi** to specify which servers you want to start, stop, or obtain a status report for. Options listed in these groups are the same that you would use in the `[mysqld]` group used for starting **mysqld**. (See, for example, Section 2.9.2.2, "Starting and Stopping MySQL Automatically".) However, when using multiple servers it is necessary that each one use its own value for options such as the Unix socket file and TCP/IP port number. For more information on which options must be unique per server in a multiple-server environment, see Section 5.10, "Running Multiple MySQL Servers on the Same Machine".

To invoke **mysqld_multi**, use the following syntax:

```
shell> mysqld_multi [options] {start|stop|report} [GNR[,GNR]...]
```

`start`, `stop`, and `report` indicate which operation you want to perform. You can perform the designated operation on a single server or multiple servers, depending on the *GNR* list that follows the option name. If there is no list, **mysqld_multi** performs the operation for all servers in the option file.

Each *GNR* value represents an option group number or range of group numbers. The value should be the number at the end of the group name in the option file. For example, the *GNR* for a group named `[mysqld17]` is `17`. To specify a range of numbers, separate the first and last numbers by a dash. The *GNR* value `10-13` represents groups `[mysqld10]` through `[mysqld13]`. Multiple groups or group ranges can be specified on the command line, separated by commas. There must be no whitespace characters (spaces or tabs) in the *GNR* list; anything after a whitespace character is ignored.

This command starts a single server using option group `[mysqld17]`:

```
shell> mysqld_multi start 17
```

This command stops several servers, using option groups `[mysql8]` and `[mysqld10]` through `[mysqld13]`:

```
shell> mysqld_multi stop 8,10-13
```

For an example of how you might set up an option file, use this command:

```
shell> mysqld_multi --example
```

mysqld_multi supports the following options:

- `--config-file=name`

  Specify the name of an alternative option file. This affects where **mysqld_multi** looks for `[mysqld#]` option groups. Without this option, all options are read from the usual `my.cnf` file.

The option does not affect where **mysqld_multi** reads its own options, which are always taken from the `[mysqld_multi]` group in the usual `my.cnf` file.

- `--example`

  Display a sample option file.

- `--help`

  Display a help message and exit.

- `--log=name`

  Specify the name of the log file. If the file exists, log output is appended to it.

- `--mysqladmin=prog_name`

  The **mysqladmin** binary to be used to stop servers.

- `--mysqld=prog_name`

  The **mysqld** binary to be used. Note that you can specify **mysqld_safe** as the value for this option also. The options are passed to **mysqld**. Just make sure that you have the directory where **mysqld** is located in your `PATH` environment variable setting or fix **mysqld_safe**.

- `--no-log`

  Print log information to stdout rather than to the log file. By default, output goes to the log file.

- `--password=password`

  The password of the MySQL account to use when invoking **mysqladmin**. Note that the password value is not optional for this option, unlike for other MySQL programs.

- `--silent`

  Disable warnings. This option was added in MySQL 4.1.6.

- `--tcp-ip`

  Connect to each MySQL server via the TCP/IP port instead of the Unix socket file. (If a socket file is missing, the server might still be running, but accessible only via the TCP/IP port.) By default, connections are made using the Unix socket file. This option affects `stop` and `report` operations.

- `--user=user_name`

  The username of the MySQL account to use when invoking **mysqladmin**.

- `--verbose`

Be more verbose. This option was added in MySQL 4.1.6.

- `--version`

  Display version information and exit.

Some notes about **mysqld_multi**:

- Make sure that the MySQL account used for stopping the **mysqld** servers (with the **mysqladmin** program) has the same username and password for each server. Also, make sure that the account has the `SHUTDOWN` privilege. If the servers that you want to manage have many different usernames or passwords for the administrative accounts, you might want to create an account on each server that has the same username and password. For example, you might set up a common `multi_admin` account by executing the following commands for each server:

  ```
  shell> mysql -u root -S /tmp/mysql.sock -proot_password
  mysql> GRANT SHUTDOWN ON *.*
      -> TO 'multi_admin'@'localhost' IDENTIFIED BY 'multipass';
  ```

  See Section 5.5.2, “How the Privilege System Works”. You have to do this for each **mysqld** server. Change the connection parameters appropriately when connecting to each one. Note that the host part of the account name must allow you to connect as `multi_admin` from the host where you want to run **mysqld_multi**.

- The `--pid-file` option is very important if you are using **mysqld_safe** to start **mysqld** (for example, `--mysqld=mysqld_safe`) Every **mysqld** should have its own process ID file. The advantage of using **mysqld_safe** instead of **mysqld** is that **mysqld_safe** ``guards'' its **mysqld** process and restarts it if the process terminates due to a signal sent using `kill -9` or for other reasons, such as a segmentation fault. Please note that the **mysqld_safe** script might require that you start it from a certain place. This means that you might have to change location to a certain directory before running **mysqld_multi**. If you have problems starting, please see the **mysqld_safe** script. Check especially the lines:

  ```
  ----------------------------------------------------------------
  MY_PWD=`pwd`
  # Check if we are starting this relative (for the binary release)
  if test -d $MY_PWD/data/mysql -a -f ./share/mysql/english/errmsg.sys -a \
   -x ./bin/mysqld
  ----------------------------------------------------------------
  ```

  See Section 5.1.3, “The **mysqld_safe** Server Startup Script”. The test performed by these lines should be successful, or you might encounter problems.

- The Unix socket file and the TCP/IP port number must be different for every **mysqld**.

- You might want to use the `--user` option for **mysqld**, but in order to do this you need to run the **mysqld_multi** script as the Unix `root` user. Having the option in the option file doesn't matter; you just get a warning if you are not the superuser and the **mysqld** processes are started under your own Unix account.

- **Important**: Make sure that the data directory is fully accessible to the Unix account that the specific **mysqld** process is started as. *Do not* use the Unix root account for this, unless you *know* what you are doing.

- **Most important**: Before using **mysqld_multi** be sure that you understand the meanings of the options that are passed to the **mysqld** servers and *why* you would want to have separate **mysqld** processes. Beware of the dangers of using multiple **mysqld** servers with the same data directory. Use separate data directories, unless you *know* what you are doing. Starting multiple servers with the same data directory *does not* give you extra performance in a threaded system. See Section 5.10, “Running Multiple MySQL Servers on the Same Machine”.

The following example shows how you might set up an option file for use with **mysqld_multi**. The first and fifth `[mysqld#]` group were intentionally left out from the example to illustrate that you can have ``gaps'' in the option file. This gives you more flexibility. The order in which the **mysqld** programs are started or stopped depends on the order in which they appear in the option file.

```
# This file should probably be in your home dir (~/.my.cnf)
# or /etc/my.cnf
# Version 2.1 by Jani Tolonen

[mysqld_multi]
mysqld     = /usr/local/bin/mysqld_safe
mysqladmin = /usr/local/bin/mysqladmin
user       = multi_admin
password   = multipass

[mysqld2]
socket     = /tmp/mysql.sock2
port       = 3307
pid-file   = /usr/local/mysql/var2/hostname.pid2
datadir    = /usr/local/mysql/var2
language   = /usr/local/share/mysql/english
user       = john

[mysqld3]
socket     = /tmp/mysql.sock3
port       = 3308
pid-file   = /usr/local/mysql/var3/hostname.pid3
datadir    = /usr/local/mysql/var3
language   = /usr/local/share/mysql/swedish
user       = monty

[mysqld4]
socket     = /tmp/mysql.sock4
port       = 3309
pid-file   = /usr/local/mysql/var4/hostname.pid4
datadir    = /usr/local/mysql/var4
language   = /usr/local/share/mysql/estonia
user       = tonu

[mysqld6]
socket     = /tmp/mysql.sock6
port       = 3311
pid-file   = /usr/local/mysql/var6/hostname.pid6
datadir    = /usr/local/mysql/var6
language   = /usr/local/share/mysql/japanese
user       = jani
```

See Section 4.3.2, “Using Option Files”.

5.2. Configuring the MySQL Server

This section discusses MySQL server configuration topics:

- Startup options that the server supports
- How to set the server SQL mode
- Server system variables
- Server status variables

5.2.1. mysqld Command-Line Options

When you start the **mysqld** server, you can specify program options using any of the methods described in Section 4.3, “Specifying Program Options”. The most common methods are to provide options in an option file or on the command line. However, in most cases it is desirable to make sure that the server uses the same options each time it runs. The best way to ensure this is to list them in an option file. See Section 4.3.2, “Using Option Files”.

mysqld reads options from the `[mysqld]` and `[server]` groups. **mysqld_safe** reads options from the `[mysqld]`, `[server]`, `[mysqld_safe]`, and `[safe_mysqld]` groups. **mysql.server** reads options from the `[mysqld]` and `[mysql.server]` groups. An embedded MySQL server usually reads options from the `[server]`, `[embedded]`, and `[`*xxxxx*`_SERVER]` groups, where *xxxxx* is the name of the application into which the server is embedded.

mysqld accepts many command-line options. For a list, execute **mysqld --help**. Before MySQL 4.1.1, `--help` prints the full help message. As of 4.1.1, it prints a brief message; to see the full list, use **mysqld --verbose --help**.

The following list shows some of the most common server options. Additional options are described elsewhere:

- Options that affect security: See Section 5.4.3, “Startup Options for **mysqld** Concerning Security”.
- SSL-related options: See Section 5.6.7.5, “SSL Command-Line Options”.
- Binary log control options: See Section 5.9.4, “The Binary Log”.
- Replication-related options: See Section 6.8, “Replication Startup Options”.
- Options specific to particular storage engines: See Section 14.1.1, “`MyISAM` Startup Options”, Section 14.4.3, “`BDB` Startup Options”, Section 15.5, “`InnoDB` Startup Options”.

You can also set the value of a server system variable by using the variable name as an option, as described later in this section.

- `--help, -?`

  Display a short help message and exit. Before MySQL 4.1.1, `--help` displays the full help message. As of 4.1.1, it displays an abbreviated message only. Use both the `--verbose` and `--help` options to see the full message.

- `--allow-suspicious-udfs`

  This option controls whether user-defined functions that have only an `xxx` symbol for the main function can be loaded. By default, the option is off and only UDFs that have at least one auxiliary symbol can be loaded. This prevents attempts at loading functions from shared object files other than those containing legitimate UDFs. This option was added in MySQL 4.0.24, 4.1.10a, and 5.0.3. See

Section 25.2.3.6, “User-defined Function Security Precautions”.

- `--ansi`

  Use standard SQL syntax instead of MySQL syntax. See Section 1.5.3, “Running MySQL in ANSI Mode”. For more precise control over the server SQL mode, use the `--sql-mode` option instead.

- `--basedir=path, -b path`

  The path to the MySQL installation directory. All paths are usually resolved relative to this.

- `--big-tables`

  Allow large result sets by saving all temporary sets in files. This option prevents most ``table full'' errors, but also slows down queries for which in-memory tables would suffice. Since MySQL 3.23.2, the server is able to handle large result sets automatically by using memory for small temporary tables and switching to disk tables where necessary.

- `--bind-address=IP`

  The IP address to bind to.

- `--console`

  Write the error log messages to stderr/stdout even if `--log-error` is specified. On Windows, **mysqld** does not close the console screen if this option is used.

- `--character-sets-dir=path`

  The directory where character sets are installed. See Section 5.8.1, “The Character Set Used for Data and Sorting”.

- `--chroot=path`

  Put the **mysqld** server in a closed environment during startup by using the `chroot()` system call. This is a recommended security measure as of MySQL 4.0. (MySQL 3.23 is not able to provide a `chroot()` jail that is 100% closed.) Note that use of this option somewhat limits `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE`.

- `--character-set-server=charset`

  Use `charset` as the default server character set. This option is available as of MySQL 4.1.3. See Section 5.8.1, “The Character Set Used for Data and Sorting”.

- `--core-file`

  Write a core file if **mysqld** dies. For some systems, you must also specify the `--core-file-size` option to **mysqld_safe**. See Section 5.1.3, “The **mysqld_safe** Server Startup Script”. Note that on some systems, such as Solaris, you do not get a core file if you are also using the `--user` option.

- `--collation-server=collation`

  Use `collation` as the default server collation. This option is available as of MySQL 4.1.3. See Section 5.8.1, “The Character Set Used for Data and Sorting”.

- `--datadir=path, -h path`

  The path to the data directory.

- `--debug[=debug_options], -# [debug_options]`

  If MySQL is configured with `--with-debug`, you can use this option to get a trace file of what **mysqld** is doing. The *`debug_options`* string often is `'d:t:o,file_name'`. See Section E.1.2, “Creating Trace Files”.

- `--default-character-set=charset`

  Use *`charset`* as the default character set. This option is deprecated in favor of `--character-set-server` as of MySQL 4.1.3. See Section 5.8.1, “The Character Set Used for Data and Sorting”.

- `--default-collation=collation`

  Use *`collation`* as the default collation. This option is deprecated in favor of `--collation-server` as of MySQL 4.1.3. See Section 5.8.1, “The Character Set Used for Data and Sorting”.

- `--default-storage-engine=type`

  This option is a synonym for `--default-table-type`. It is available as of MySQL 4.1.2.

- `--default-table-type=type`

  Set the default table type for tables. See Chapter 14, *MySQL Storage Engines and Table Types*.

- `--default-time-zone=type`

  Set the default server time zone. This option sets the global `time_zone` system variable. If this option is not given, the default time zone is the same as the system time zone (given by the value of the `system_time_zone` system variable. This option is available as of MySQL 4.1.3.

- `--delay-key-write[= OFF | ON | ALL]`

  How the `DELAYED KEYS` option should be used. Delayed key writing causes key buffers not to be flushed between writes for `MyISAM` tables. `OFF` disables delayed key writes. `ON` enables delayed key writes for those tables that were created with the `DELAYED KEYS` option. `ALL` delays key writes for all `MyISAM` tables. Available as of MySQL 4.0.3. See Section 7.5.2, “Tuning Server Parameters”. See Section 14.1.1, “`MyISAM` Startup Options”.

  Note: If you set this variable to `ALL`, you should not use `MyISAM` tables from within another program (such as from another MySQL server or with **myisamchk**) when the table is in use. Doing so leads to index corruption.

- `--delay-key-write-for-all-tables`

  Old form of `--delay-key-write=ALL` for use prior to MySQL 4.0.3. As of 4.0.3, use `--delay-key-write` instead.

- `--des-key-file=file_name`

  Read the default keys used by `DES_ENCRYPT()` and `DES_DECRYPT()` from this file.

- `--enable-named-pipe`

  Enable support for named pipes. This option applies only on Windows NT, 2000, XP, and 2003 systems, and can be used only with the **mysqld-nt** and **mysqld-max-nt** servers that support named pipe connections.

- `--exit-info[=flags], -T [flags]`

  This is a bit mask of different flags you can use for debugging the **mysqld** server. Do not use this option unless you know exactly what it does!

- `--external-locking`

  Enable system locking. Note that if you use this option on a system on which `lockd` does not fully work (as on Linux), it is easy for **mysqld** to deadlock. This option previously was named `--enable-locking`.

  Note: If you use this option to enable updates to `MyISAM` tables from many MySQL processes, you have to ensure that these conditions are satisfied:

  - You should not use the query cache for queries that use tables that are updated by another process.
  - You should not use `--delay-key-write=ALL` or `DELAY_KEY_WRITE=1` on any shared tables.

  The easiest way to ensure this is to always use `--external-locking` together with `--delay-key-write=OFF --query-cache-size=0`.

  (This is not done by default because in many setups it's useful to have a mixture of the above options.)

- `--flush`

  Flush all changes to disk after each SQL statement. Normally MySQL does a write of all changes to disk only after each SQL statement and lets the operating system handle the synching to disk. See Section A.4.2, “What to Do If MySQL Keeps Crashing”.

- `--init-file=file`

  Read SQL statements from this file at startup. Each statement must be on a single line and should not include comments.

- `--innodb-safe-binlog`

  Adds consistency guarantees between the content of `InnoDB` tables and the binary log. See Section 5.9.4, “The Binary Log”.

- `--language=lang_name, -L lang_name`

  Client error messages in given language. `lang_name` can be given as the language name or as the full pathname to the directory where the language files are installed. See Section 5.8.2, “Setting the Error Message Language”.

- `--large-pages`

  Some hardware/operating system architectures support memory pages greater than the default (usually 4 KB). The actual implementation of this support depends on the underlying hardware and OS. Applications that perform a lot of memory access may obtain performance improvements by using large pages due to reduced Translation Lookaside Buffer (TLB) misses.

  Currently, MySQL supports only the Linux implementation of large pages support (which is called HugeTLB in Linux). We have plans to extend this support to FreeBSD, Solaris and possibly other platforms.

Before large pages can be used on Linux, it is necessary to configure the HugeTLB memory pool. For reference, consult the `hugetlbpage.txt` file in the Linux kernel source.

This option is disabled by default. It was added in MySQL 5.0.3.

- `--log[=file], -l [file]`

  Log connections and queries to this file. See Section 5.9.2, “The General Query Log”. If you don't specify a filename, MySQL uses `host_name.log` as the filename.

- `--log-bin=[file]`

  The binary log file. Log all queries that change data to this file. Used for backup and replication. See Section 5.9.4, “The Binary Log”. It is recommended to specify a filename (see Section 1.5.7.3, “Open Bugs and Design Deficiencies in MySQL” for the reason) otherwise MySQL uses `host_name-bin` as the log file basename.

- `--log-bin-index[=file]`

  The index file for binary log filenames. See Section 5.9.4, “The Binary Log”. If you don't specify a filename, and if you didn't specify one in `--log-bin`, MySQL uses `host_name-bin.index` as the filename.

- `--log-error[=file]`

  Log errors and startup messages to this file. See Section 5.9.1, “The Error Log”. If you don't specify a filename, MySQL uses `host_name.err` as the filename.

- `--log-isam[=file]`

  Log all `ISAM`/`MyISAM` changes to this file (used only when debugging `ISAM`/`MyISAM`).

- `--log-long-format`

  Log some extra information to the log files (update log, binary update log, and slow queries log, whatever log has been activated). For example, username and timestamp are logged for queries. Before MySQL 4.1, if you are using `--log-slow-queries` and `--log-long-format`, queries that are not using indexes also are logged to the slow query log. `--log-long-format` is deprecated as of MySQL version 4.1, when `--log-short-format` was introduced. (Long log format is the default setting since version 4.1.) Also note that starting with MySQL 4.1, the `--log-queries-not-using-indexes` option is available for the purpose of logging queries that do not use indexes to the slow query log.

- `--log-queries-not-using-indexes`

  If you are using this option with `--log-slow-queries`, then queries that are not using indexes also are logged to the slow query log. This option is available as of MySQL 4.1. See Section 5.9.5, “The Slow Query Log”.

- `--log-short-format`

  Log less information to the log files (update log, binary update log, and slow queries log, whatever log has been activated). For example, username and timestamp are not logged for queries. This option was introduced in MySQL 4.1.

- `--log-slow-queries[=file]`

  Log all queries that have taken more than `long_query_time` seconds to execute to this file. See

Section 5.9.5, “The Slow Query Log”. Note that the default for the amount of information logged has changed in MySQL 4.1. See the `--log-long-format` and `--log-short-format` options for details.

- `--log-update[=file]`

  Log updates to *file#* where # is a unique number if not given. See Section 5.9.3, “The Update Log”. The update log is deprecated and is removed in MySQL 5.0.0; you should use the binary log instead (`--log-bin`). See Section 5.9.4, “The Binary Log”. Starting from version 5.0.0, using `--log-update` turns on the binary log instead (see Section D.1.5, “Changes in release 5.0.0 (22 Dec 2003: Alpha)”).

- `--log-warnings, -W`

  Print out warnings such as `Aborted connection...` to the error log. Enabling this option is recommended, for example, if you use replication (you get more information about what is happening, such as messages about network failures and reconnections). This option is enabled by default as of MySQL 4.0.19 and 4.1.2; to disable it, use `--skip-log-warnings`. As of MySQL 4.0.21 and 4.1.3, aborted connections are not logged to the error log unless the value is greater than 1. See Section A.2.10, “Communication Errors and Aborted Connections”.

  This option was named `--warnings` before MySQL 4.0.

- `--low-priority-updates`

  Table-modifying operations (`INSERT`, `REPLACE`, `DELETE`, `UPDATE`) have lower priority than selects. This can also be done via `{INSERT | REPLACE | DELETE | UPDATE} LOW_PRIORITY ...` to lower the priority of only one query, or by `SET LOW_PRIORITY_UPDATES=1` to change the priority in one thread. See Section 7.3.2, “Table Locking Issues”.

- `--memlock`

  Lock the **mysqld** process in memory. This works on systems such as Solaris that support the `mlockall()` system call. This might help if you have a problem where the operating system is causing **mysqld** to swap on disk. Note that use of this option requires that you run the server as `root`, which is normally not a good idea for security reasons.

- `--myisam-recover [=option[,option...]]]`

  Set the `MyISAM` storage engine recovery mode. The option value is any combination of the values of `DEFAULT`, `BACKUP`, `FORCE`, or `QUICK`. If you specify multiple values, separate them by commas. You can also use a value of `""` to disable this option. If this option is used, **mysqld**, when it opens a `MyISAM` table, checks whether the table is marked as crashed or wasn't closed properly. (The last option works only if you are running with `--skip-external-locking`.) If this is the case, **mysqld** runs a check on the table. If the table was corrupted, **mysqld** attempts to repair it.

  The following options affect how the repair works:

Option	Description
`DEFAULT`	The same as not giving any option to `--myisam-recover`.
`BACKUP`	If the data file was changed during recovery, save a backup of the *`tbl_name`*`.MYD` file as *`tbl_name-datetime`*`.BAK`.
`FORCE`	Run recovery even if we would lose more than one row from the `.MYD` file.
`QUICK`	Don't check the rows in the table if there aren't any delete blocks.

Before a table is automatically repaired, MySQL adds a note about this in the error log. If you want to be able to recover from most problems without user intervention, you should use the options `BACKUP,FORCE`. This forces a repair of a table even if some rows would be deleted, but it keeps the old data file as a backup so that you can later examine what happened.

This option is available as of MySQL 3.23.25.

- `--ndb-connectstring=connect_string`

  When using the `NDB` storage engine, it is possible to point out the management server that distributes the cluster configuration by setting the connect string option. See Section 16.5.4.2, “The MySQL Cluster `connectstring`” for syntax.

- `--ndbcluster`

  If the binary includes support for the `NDB Cluster` storage engine (from version 4.1.3, the MySQL-Max binaries are built with `NDB Cluster` enabled) the default disabling of support for the `NDB Cluster` storage engine can be overruled by using this option. Using the `NDB Cluster` storage engine is necessary for using MySQL Cluster. See Chapter 16, *MySQL Cluster*.

- `--new`

  The `--new` option can be used to make the server behave as 4.1 in certain respects, easing a 4.0 to 4.1 upgrade:

  - Hexadecimal strings such as `0xFF` are treated as strings by default rather than as numbers. (Works in 4.0.12 and up.)

  - `TIMESTAMP` is returned as a string with the format `'YYYY-MM-DD HH:MM:SS'`. (Works in 4.0.13 and up.) See Chapter 11, *Column Types*.

  This option can be used to help you see how your applications behave in MySQL 4.1, without actually upgrading to 4.1.

- `--old-passwords`

  Force the server to generate short (pre-4.1) password hashes for new passwords. This is useful for compatibility when the server must support older client programs. See Section 5.5.9, “Password Hashing in MySQL 4.1”.

- `--old-protocol, -o`

  Use the 3.20 protocol for compatibility with some very old clients. See Section 2.10.6, “Upgrading from Version 3.20 to 3.21”.

- `--one-thread`

  Only use one thread (for debugging under Linux). This option is available only if the server is built with debugging enabled. See Section E.1, “Debugging a MySQL Server”.

- `--open-files-limit=count`

  To change the number of file descriptors available to **mysqld**. If this is not set or set to 0, then **mysqld** uses this value to reserve file descriptors to use with `setrlimit()`. If this value is 0 then **mysqld** reserves `max_connections*5` or `max_connections + table_cache*2` (whichever is larger) number of files. You should try increasing this if **mysqld** gives you the error "Too many open files."

- `--pid-file=path`

  The path to the process ID file used by **mysqld_safe**.

- `--port=port_num, -P port_num`

  The port number to use when listening for TCP/IP connections.

- `--safe-mode`

  Skip some optimization stages.

- `--safe-show-database`

  With this option, the `SHOW DATABASES` statement displays only the names of those databases for which the user has some kind of privilege. As of MySQL 4.0.2, this option is deprecated and doesn't do anything (it is enabled by default), because there is a `SHOW DATABASES` privilege that can be used to control access to database names on a per-account basis. See Section 5.5.3, “Privileges Provided by MySQL”.

- `--safe-user-create`

  If this is enabled, a user can't create new users with the `GRANT` statement, if the user doesn't have the `INSERT` privilege for the `mysql.user` table or any column in the table.

- `--secure-auth`

  Disallow authentication for accounts that have old (pre-4.1) passwords. This option is available as of MySQL 4.1.1.

- `--shared-memory`

  Enable shared-memory connections by local clients. This option is available only on Windows. It was added in MySQL 4.1.0.

- `--shared-memory-base-name=name`

  The name to use for shared-memory connections. This option is available only on Windows. It was added in MySQL 4.1.0.

- `--skip-bdb`

  Disable the `BDB` storage engine. This saves memory and might speed up some operations. Do not use this option if you require `BDB` tables.

- `--skip-concurrent-insert`

  Turn off the ability to select and insert at the same time on `MyISAM` tables. (This is to be used only if you think you have found a bug in this feature.)

- `--skip-delay-key-write`

  Ignore the `DELAY_KEY_WRITE` option for all tables. As of MySQL 4.0.3, you should use `--delay-key-write=OFF` instead. See Section 7.5.2, “Tuning Server Parameters”.

- `--skip-external-locking`

  Don't use system locking. To use **isamchk** or **myisamchk**, you must shut down the server. See Section 1.2.3, “MySQL Stability”. In MySQL 3.23, you can use `CHECK TABLE` and `REPAIR TABLE` to check and repair `MyISAM` tables. This option previously was named `--skip-locking`.

- `--skip-grant-tables`

  This option causes the server not to use the privilege system at all. This gives everyone *full access* to all databases! (You can tell a running server to start using the grant tables again by executing a **mysqladmin flush-privileges** or **mysqladmin reload** command, or by issuing a `FLUSH PRIVILEGES` statement.)

- `--skip-host-cache`

  Do not use the internal hostname cache for faster name-to-IP resolution. Instead, query the DNS server every time a client connects. See Section 7.5.6, “How MySQL Uses DNS”.

- `--skip-innodb`

  Disable the `InnoDB` storage engine. This saves memory and disk space and might speed up some operations. Do not use this option if you require `InnoDB` tables.

- `--skip-isam`

  Disable the `ISAM` storage engine. As of MySQL 4.1, `ISAM` is disabled by default, so this option applies only if the server was configured with support for `ISAM`. This option was added in MySQL 4.1.1.

- `--skip-name-resolve`

  Do not resolve hostnames when checking client connections. Use only IP numbers. If you use this option, all `Host` column values in the grant tables must be IP numbers or `localhost`. See Section 7.5.6, “How MySQL Uses DNS”.

- `--skip-ndbcluster`

  Disable the `NDB Cluster` storage engine. This is the default for binaries that were built with `NDB Cluster` storage engine support, this means that the system only allocates memory and other resources for this storage engine if it is explicitly enabled.

- `--skip-networking`

  Don't listen for TCP/IP connections at all. All interaction with **mysqld** must be made via named pipes or shared memory (on Windows) or Unix socket files (on Unix). This option is highly recommended for systems where only local clients are allowed. See Section 7.5.6, “How MySQL Uses DNS”.

- `--skip-new`

  Don't use new, possibly wrong routines.

- `--skip-symlink`

  This is the old form of `--skip-symbolic-links`, for use before MySQL 4.0.13.

- `--symbolic-links, --skip-symbolic-links`

  Enable or disable symbolic link support. This option has different effects on Windows and Unix:

  - On Windows, enabling symbolic links allows you to establish a symbolic link to a database directory by creating a `directory.sym` file that contains the path to the real directory. See Section 7.6.1.3, “Using Symbolic Links for Databases on Windows”.

- On Unix, enabling symbolic links means that you can link a `MyISAM` index file or data file to another directory with the `INDEX DIRECTORY` or `DATA DIRECTORY` options of the `CREATE TABLE` statement. If you delete or rename the table, the files that its symbolic links point to also are deleted or renamed. See Section 13.2.6, "`CREATE TABLE` Syntax".

This option was added in MySQL 4.0.13.

- `--skip-safemalloc`

  If MySQL is configured with `--with-debug=full`, all MySQL programs check for memory overruns during each memory allocation and memory freeing operation. This checking is very slow, so for the server you can avoid it when you don't need it by using the `--skip-safemalloc` option.

- `--skip-show-database`

  With this option, the `SHOW DATABASES` statement is allowed only to users who have the `SHOW DATABASES` privilege, and the statement displays all database names. Without this option, `SHOW DATABASES` is allowed to all users, but displays each database name only if the user has the `SHOW DATABASES` privilege or some privilege for the database.

- `--skip-stack-trace`

  Don't write stack traces. This option is useful when you are running **mysqld** under a debugger. On some systems, you also must use this option to get a core file. See Section E.1, "Debugging a MySQL Server".

- `--skip-thread-priority`

  Disable using thread priorities for faster response time.

- `--socket=`*`path`*

  On Unix, this option specifies the Unix socket file to use for local connections. The default value is `/tmp/mysql.sock`. On Windows, the option specifies the pipe name to use for local connections that use a named pipe. The default value is `MySQL`.

- `--sql-mode=`*`value`*`[,`*`value`*`[,`*`value`*`...]]`

  Set the SQL mode for MySQL. See Section 5.2.2, "The Server SQL Mode". This option was added in 3.23.41.

- `--temp-pool`

  This option causes most temporary files created by the server to use a small set of names, rather than a unique name for each new file. This works around a problem in the Linux kernel dealing with creating many new files with different names. With the old behavior, Linux seems to ``leak'' memory, because it's being allocated to the directory entry cache rather than to the disk cache.

- `--transaction-isolation=`*`level`*

  Sets the default transaction isolation level, which can be `READ-UNCOMMITTED`, `READ-COMMITTED`, `REPEATABLE-READ`, or `SERIALIZABLE`. See Section 13.4.6, "`SET TRANSACTION` Syntax".

- `--tmpdir=path, -t path`

  The path of the directory to use for creating temporary files. It might be useful if your default `/tmp` directory resides on a partition that is too small to hold temporary tables. Starting from MySQL 4.1, this option accepts several paths that are used in round-robin fashion. Paths should be separated by colon characters (`':'`) on Unix and semicolon characters (`';'`) on Windows, NetWare, and OS/2. If the MySQL server is acting as a replication slave, you should not set `--tmpdir` to point to a directory on a memory-based filesystem or to a directory that is cleared when the server host restarts. A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or `LOAD DATA INFILE` operations. If files in the temporary file directory are lost when the server restarts, replication fails.

- `--user={user_name | user_id}, -u {user_name | user_id}`

  Run the **mysqld** server as the user having the name *`user_name`* or the numeric user ID *`user_id`*. (``User'' in this context refers to a system login account, not a MySQL user listed in the grant tables.)

  This option is *mandatory* when starting **mysqld** as `root`. The server changes its user ID during its startup sequence, causing it to run as that particular user rather than as `root`. See Section 5.4.1, “General Security Guidelines”.

  Starting from MySQL 3.23.56 and 4.0.12: To avoid a possible security hole where a user adds a `--user=root` option to some `my.cnf` file (thus causing the server to run as `root`), **mysqld** uses only the first `--user` option specified and produces a warning if there are multiple `--user` options. Options in `/etc/my.cnf` and `$MYSQL_HOME/my.cnf` are processed before command-line options, so it is recommended that you put a `--user` option in `/etc/my.cnf` and specify a value other than `root`. The option in `/etc/my.cnf` is found before any other `--user` options, which ensures that the server runs as a user other than `root`, and that a warning results if any other `--user` option is found.

- `--version, -V`

  Display version information and exit.

As of MySQL 4.0, you can assign a value to a server system variable by using an option of the form `--var_name=value`. For example, `--key_buffer_size=32M` sets the `key_buffer_size` variable to a value of 32MB.

Note that when setting a variable to a value, MySQL might automatically correct it to stay within a given range, or adjust the value to the closest allowable value if only certain values are allowed.

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. However, this syntax is deprecated as of MySQL 4.0.

You can find a full description for all variables in Section 5.2.3, “Server System Variables”. The section on tuning server parameters includes information on how to optimize them. See Section 7.5.2, “Tuning Server Parameters”.

You can change the values of most system variables for a running server with the `SET` statement. See Section 13.5.3, “`SET` Syntax”.

If you want to restrict the maximum value that a startup option can be set to with `SET`, you can define this by using the `--maximum-var_name` command-line option.

5.2.2. The Server SQL Mode

The MySQL server can operate in different SQL modes, and (as of MySQL 4.1) can apply these modes

differentially for different clients. This allows an application to tailor server operation to its own requirements.

Modes define what SQL syntax MySQL should support and what kind of data validation checks it should perform. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers.

You can set the default SQL mode by starting **mysqld** with the `--sql-mode="modes"` option. The value also can be empty (`--sql-mode=""`) if you want to reset it.

Beginning with MySQL 4.1, you can also change the SQL mode after startup time by setting the `sql_mode` variable with a `SET [SESSION|GLOBAL] sql_mode='modes'` statement. Setting the `GLOBAL` variable requires the `SUPER` privilege and affects the operation of all clients that connect from that time on. Setting the `SESSION` variable affects only the current client. Any client can change its session `sql_mode` value.

`modes` is a list of different modes separated by comma (`','`) characters. You can retrieve the current mode by issuing a `SELECT @@sql_mode` statement. The default value is empty (no modes set).

The most important `sql_mode` values are probably these:

- `ANSI`

  Change syntax and behavior to be more conformant to standard SQL. (New in MySQL 4.1.1)

- `STRICT_TRANS_TABLES`

  If a value could not be inserted as given into a transactional table, abort the statement. For a non-transactional table, abort the statement if the value occurs in a single-row statement or the first row of a multiple-row statement. More detail is given later in this section. (New in MySQL 5.0.2)

- `TRADITIONAL`

  Make MySQL behave like a ``traditional'' SQL database system. A simple description of this mode is ``give an error instead of a warning'' when inserting an incorrect value into a column. **Note**: The `INSERT`/`UPDATE` aborts as soon as the error is noticed. This may not be what you want if you are using a non-transactional storage engine, because data changes made prior to the error are not be rolled back, resulting in a ``partially-done'' update. (New in MySQL 5.0.2)

When this manual refers to ``strict mode,'' it means a mode where at least one of `STRICT_TRANS_TABLES` or `STRICT_ALL_TABLES` is enabled.

The following list describes all the supported modes:

- `ALLOW_INVALID_DATES`

  Don't do full checking of dates in strict mode. Check only that the month is in the range from 1 to 12 and the day is in the range from 1 to 31. This is very convenient for Web applications where you obtain year, month, and day in three different fields and you want to store exactly what the user inserted (without date validation). This mode applies to `DATE` and `DATETIME` columns. It does not apply `TIMESTAMP` columns, which always require a valid date.

  This mode is new in MySQL 5.0.2. Before 5.0.2, this was the default MySQL date-handling mode. As of 5.0.2, enabling strict mode causes the server to require that month and day values be legal, not

just in the range from 1 to 12 and 1 to 31. For example, `'2004-04-31'` is legal with strict mode disabled, but illegal with strict mode enabled. To allow such dates in strict mode, enable `ALLOW_INVALID_DATES` as well.

- `ANSI_QUOTES`

  Treat `'"'` as an identifier quote character (like the `'`'` quote character) and not as a string quote character. You can still use `'`'` to quote identifiers in ANSI mode. With `ANSI_QUOTES` enabled, you cannot use double quotes to quote a literal string, because it is interpreted as an identifier. (New in MySQL 4.0.0)

- `ERROR_FOR_DIVISION_BY_ZERO`

  Produce an error in strict mode (otherwise a warning) when we encounter a division by zero (or `MOD(X,0)`) during an `INSERT` or `UPDATE`, or in any expression (for example, in a select list or `WHERE` clause) that involves table data and a division by zero. If this mode is not given, MySQL instead returns `NULL` for divisions by zero. If used in `INSERT IGNORE` or `UPDATE IGNORE`, MySQL generates a warning for divisions by zero, but the result of the operation is `NULL`. (New in MySQL 5.0.2)

- `HIGH_NOT_PRECEDENCE`

  From MySQL 5.0.2 on, the `NOT` operator precedence is handled so that expressions such as `NOT a BETWEEN b AND c` are parsed as `NOT (a BETWEEN b AND c)`. Before MySQL 5.0.2, the expression is parsed as `(NOT a) BETWEEN b AND c`. The old higher-precedence behavior can be obtained by enabling the `HIGH_NOT_PRECEDENCE` SQL mode. (New in MySQL 5.0.2)

```
mysql> SET sql_mode = '';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
        -> 0
mysql> SET sql_mode = 'broken_not';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
        -> 1
```

- `IGNORE_SPACE`

  Allow spaces between a function name and the `'('` character. This forces all function names to be treated as reserved words. As a result, if you want to access any database, table, or column name that is a reserved word, you must quote it. For example, because there is a `USER()` function, the name of the `user` table in the `mysql` database and the `User` column in that table become reserved, so you must quote them:

```
SELECT "User" FROM mysql."user";
```

  (New in MySQL 4.0.0)

- `NO_AUTO_CREATE_USER`

  Prevent `GRANT` from automatically creating new users if it would otherwise do so, unless a password also is specified. (New in MySQL 5.0.2)

- NO_AUTO_VALUE_ON_ZERO

  NO_AUTO_VALUE_ON_ZERO affects handling of AUTO_INCREMENT columns. Normally, you generate the next sequence number for the column by inserting either NULL or 0 into it. NO_AUTO_VALUE_ON_ZERO suppresses this behavior for 0 so that only NULL generates the next sequence number. (New in MySQL 4.1.1)

  This mode can be useful if 0 has been stored in a table's AUTO_INCREMENT column. (This is not a recommended practice, by the way.) For example, if you dump the table with **mysqldump** and then reload it, MySQL normally generates new sequence numbers when it encounters the 0 values, resulting in a table with different contents than the one that was dumped. Enabling NO_AUTO_VALUE_ON_ZERO before reloading the dump file solves this problem. As of MySQL 4.1.1, **mysqldump** automatically includes a statement in the dump output to enable NO_AUTO_VALUE_ON_ZERO.

- NO_BACKSLASH_ESCAPES

  Disable the use of the backslash character ('\') as an escape character within strings. With this mode enabled, backslash becomes any ordinary character like any other. (New in MySQL 5.0.1)

- NO_DIR_IN_CREATE

  When creating a table, ignore all INDEX DIRECTORY and DATA DIRECTORY directives. This option is useful on slave replication servers. (New in MySQL 4.0.15)

- NO_FIELD_OPTIONS

  Don't print MySQL-specific column options in the output of SHOW CREATE TABLE. This mode is used by **mysqldump** in portability mode. (New in MySQL 4.1.1)

- NO_KEY_OPTIONS

  Don't print MySQL-specific index options in the output of SHOW CREATE TABLE. This mode is used by **mysqldump** in portability mode. (New in MySQL 4.1.1)

- NO_TABLE_OPTIONS

  Don't print MySQL-specific table options (such as ENGINE) in the output of SHOW CREATE TABLE. This mode is used by **mysqldump** in portability mode. (New in MySQL 4.1.1)

- NO_UNSIGNED_SUBTRACTION

  In subtraction operations, don't mark the result as UNSIGNED if one of the operands is unsigned. Note that this makes UNSIGNED BIGINT not 100% usable in all contexts. See Section 12.7, “Cast Functions and Operators”. (New in MySQL 4.0.2)

- NO_ZERO_DATE

  Don't allow '0000-00-00' as a valid date. You can still insert zero dates with the IGNORE option. (New in MySQL 5.0.2)

- `NO_ZERO_IN_DATE`

  Don't accept dates where the month or day part is 0. If used with the `IGNORE` option, we insert a `'0000-00-00'` date for any such date. (New in MySQL 5.0.2)

- `ONLY_FULL_GROUP_BY`

  Don't allow queries that in the `GROUP BY` part refer to a not selected column. (New in MySQL 4.0.0)

- `PIPES_AS_CONCAT`

  Treat `||` as a string concatenation operator (same as `CONCAT()`) rather than as a synonym for `OR`. (New in MySQL 4.0.0)

- `REAL_AS_FLOAT`

  Treat `REAL` as a synonym for `FLOAT` rather than as a synonym for `DOUBLE`. (New in MySQL 4.0.0)

- `STRICT_ALL_TABLES`

  Enable strict mode for all storage engines. Invalid data values are rejected. Additional detail follows. (New in MySQL 5.0.2)

- `STRICT_TRANS_TABLES`

  Enable strict mode for transactional storage engines, and when possible for non-transactional storage engines. Additional detail follows. (New in MySQL 5.0.2)

Strict mode controls how MySQL handles values that are invalid or missing. A value can be invalid for several reasons. For example, it might have the wrong data type for the column, or it might be out of range. A value is missing when a new row to be inserted does not contain a value for a column that has no explicit `DEFAULT` clause in its definition.

For transactional tables, an error occurs for invalid or missing values in a statement when either of the `STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES` modes are enabled. The statement is aborted and rolled back.

For non-transactional tables, the behavior is the same for either mode, if the bad value occurs in the first row to be inserted or updated. The statement is aborted and the table remains unchanged. If the statement inserts or modifies multiple rows and the bad value occurs in the second or later row, the result depends on which strict option is enabled:

- For `STRICT_ALL_TABLES`, MySQL returns an error and ignores the rest of the rows. However, in this case, the earlier rows still have been inserted or updated. This means that you might get a partial update, which might not be what you want. To avoid this, it's best to use single-row statements because these can be aborted without changing the table.

- For `STRICT_TRANS_TABLES`, MySQL converts an invalid value to the closest valid value for the column and insert the adjusted value. If a value is missing, MySQL inserts the implicit default value for the column data type. In either case, MySQL generates a warning rather than an error and contin-

ues processing the statement. Implicit defaults are described in Section 13.2.6, “CREATE TABLE Syntax”.

Strict mode disallows invalid date values such as '2004-04-31'. It does not disallow dates with zero parts such as 2004-04-00' or ``zero'' dates. To disallow these as well, enable the NO_ZERO_IN_DATE and NO_ZERO_DATE SQL modes in addition to strict mode.

If you are not using strict mode (that is, neither STRICT_TRANS_TABLES nor STRICT_ALL_TABLES is enabled), MySQL inserts adjusted values for invalid or missing values and produces warnings. In strict mode, you can produce this behavior by using INSERT IGNORE or UPDATE IGNORE. See Section 13.5.4.20, “SHOW WARNINGS Syntax”.

The following special modes are provided as shorthand for combinations of mode values from the preceding list. All are available as of MySQL 4.1.1, except TRADITIONAL (5.0.2).

The descriptions include all mode values that are available in the most recent version of MySQL. For older versions, a combination mode does not include individual mode values that are not available except in newer versions.

- ANSI

  Equivalent to REAL_AS_FLOAT, PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE. Before MySQL 4.1.11 and 5.0.3, ANSI also includes ONLY_FULL_GROUP_BY. See Section 1.5.3, “Running MySQL in ANSI Mode”.

- DB2

  Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS.

- MAXDB

  Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS, NO_AUTO_CREATE_USER.

- MSSQL

  Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS.

- MYSQL323

  Equivalent to NO_FIELD_OPTIONS, HIGH_NOT_PRECEDENCE.

- MYSQL40

  Equivalent to NO_FIELD_OPTIONS, HIGH_NOT_PRECEDENCE.

- ORACLE

  Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS, NO_AUTO_CREATE_USER.

- POSTGRESQL

  Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS.

- TRADITIONAL

  Equivalent to STRICT_TRANS_TABLES, STRICT_ALL_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, NO_AUTO_CREATE_USER.

5.2.3. Server System Variables

The server maintains many system variables that indicate how it is configured. All of them have default values. They can be set at server startup using options on the command line or in option files. Most of them can be set at runtime using the SET statement.

Beginning with MySQL 4.0.3, the **mysqld** server maintains two kinds of variables. Global variables affect the overall operation of the server. Session variables affect its operation for individual client connections.

When the server starts, it initializes all global variables to their default values. These defaults can be changed by options specified in option files or on the command line. After the server starts, those global variables that are dynamic can be changed by connecting to the server and issuing a SET GLOBAL *var_name* statement. To change a global variable, you must have the SUPER privilege.

The server also maintains a set of session variables for each client that connects. The client's session variables are initialized at connect time using the current values of the corresponding global variables. For those session variables that are dynamic, the client can change them by issuing a SET SESSION *var_name* statement. Setting a session variable requires no special privilege, but a client can change only its own session variables, not those of any other client.

A change to a global variable is visible to any client that accesses that global variable. However, it affects the corresponding session variable that is initialized from the global variable only for clients that connect after the change. It does not affect the session variable for any client that is currently connected (not even that of the client that issues the SET GLOBAL statement).

When setting a variable using a startup option, variable values can be given with a suffix of K, M, or G to indicate kilobytes, megabytes, or gigabytes, respectively. For example, the following command starts the server with a key buffer size of 16 megabytes:

```
mysqld --key_buffer_size=16M
```

Before MySQL 4.0, use this syntax instead:

```
mysqld --set-variable=key_buffer_size=16M
```

The lettercase of suffix letters does not matter; 16M and 16m are equivalent.

At runtime, use the SET statement to set system variables. In this context, suffix letters cannot be used, but the value can take the form of an expression:

```
mysql> SET sort_buffer_size = 10 * 1024 * 1024;
```

To specify explicitly whether to set the global or session variable, use the `GLOBAL` or `SESSION` options:

```
mysql> SET GLOBAL sort_buffer_size = 10 * 1024 * 1024;
mysql> SET SESSION sort_buffer_size = 10 * 1024 * 1024;
```

Without either option, the statement sets the session variable.

The variables that can be set at runtime are listed in Section 5.2.3.1, “Dynamic System Variables”.

If you want to restrict the maximum value to which a system variable can be set with the `SET` statement, you can specify this maximum by using an option of the form `--maximum-`*`var_name`* at server startup. For example, to prevent the value of `query_cache_size` from being increased to more than 32MB at runtime, use the option `--maximum-query_cache_size=32M`. This feature is available as of MySQL 4.0.2.

You can view system variables and their values by using the `SHOW VARIABLES` statement. See Section 9.4, “System Variables” for more information.

```
mysql> SHOW VARIABLES;
mysql> SHOW VARIABLES;
+-----------------------------------+-----------------------------------------------
| Variable_name                     | Value
+-----------------------------------+-----------------------------------------------
| auto_increment_increment          | 1
| auto_increment_offset             | 1
| back_log                          | 50
| basedir                           | /usr/local/mysql
| bdb_cache_size                    | 8388600
| bdb_home                          | /usr/local/mysql
| bdb_log_buffer_size               | 131072
| bdb_logdir                        |
| bdb_max_lock                      | 10000
| bdb_shared_data                   | OFF
| bdb_tmpdir                        | /tmp/
| binlog_cache_size                 | 32768
| bulk_insert_buffer_size           | 8388608
| character_set_client              | latin1
| character_set_connection          | latin1
| character_set_database            | latin1
| character_set_results             | latin1
| character_set_server              | latin1
| character_set_system              | utf8
| character_sets_dir                | /usr/local/mysql/share/charsets/
| collation_connection              | latin1_swedish_ci
| collation_database                | latin1_swedish_ci
| collation_server                  | latin1_swedish_ci
| concurrent_insert                 | ON
| connect_timeout                   | 5
| datadir                           | /usr/local/mysql/data/
| date_format                       | %Y-%m-%d
| datetime_format                   | %Y-%m-%d %H:%i:%s
| default_week_format               | 0
| delay_key_write                   | ON
| delayed_insert_limit              | 100
| delayed_insert_timeout            | 300
| delayed_queue_size                | 1000
| expire_logs_days                  | 0
| flush                             | OFF
| flush_time                        | 1800
| ft_boolean_syntax                 | + -><()~*:""&|
| ft_max_word_len                   | 84
```

```
| ft_min_word_len                  | 4
| ft_query_expansion_limit         | 20
| ft_stopword_file                 | (built-in)
| group_concat_max_len             | 1024
| have_archive                     | NO
| have_bdb                         | YES
| have_compress                    | YES
| have_crypt                       | NO
| have_csv                         | NO
| have_example_engine              | NO
| have_geometry                    | YES
| have_innodb                      | YES
| have_isam                        | NO
| have_ndbcluster                  | NO
| have_openssl                     | YES
| have_query_cache                 | YES
| have_raid                        | NO
| have_rtree_keys                  | YES
| have_symlink                     | YES
| init_connect                     |
| init_file                        |
| init_slave                       |
| innodb_additional_mem_pool_size  | 2097152
| innodb_autoextend_increment      | 8
| innodb_buffer_pool_awe_mem_mb    | 0
| innodb_buffer_pool_size          | 8388608
| innodb_data_file_path            | ibdata1:10M:autoextend
| innodb_data_home_dir             |
| innodb_fast_shutdown             | ON
| innodb_file_io_threads           | 4
| innodb_file_per_table            | OFF
| innodb_locks_unsafe_for_binlog   | OFF
| innodb_flush_log_at_trx_commit   | 1
| innodb_flush_method              |
| innodb_force_recovery            | 0
| innodb_lock_wait_timeout         | 50
| innodb_log_arch_dir              |
| innodb_log_archive               | OFF
| innodb_log_buffer_size           | 1048576
| innodb_log_file_size             | 10485760
| innodb_log_files_in_group        | 2
| innodb_log_group_home_dir        | ./
| innodb_max_dirty_pages_pct       | 90
| innodb_max_purge_lag             | 0
| innodb_table_locks               | ON
| innodb_max_purge_lag             | 0
| innodb_mirrored_log_groups       | 1
| innodb_open_files                | 300
| innodb_thread_concurrency        | 8
| interactive_timeout              | 28800
| join_buffer_size                 | 131072
| key_buffer_size                  | 4194304
| key_cache_age_threshold          | 300
| key_cache_block_size             | 1024
| key_cache_division_limit         | 100
| language                         | /usr/local/mysql/share/english/
| large_files_support              | ON
| license                          | GPL
| local_infile                     | ON
| log                              | OFF
| log_bin                          | OFF
| log_error                        | ./gigan.err
| log_slave_updates                | OFF
| log_slow_queries                 | OFF
```

```
| log_update                        | OFF
| log_warnings                      | 1
| long_query_time                   | 10
| low_priority_updates              | OFF
| lower_case_file_system            | OFF
| lower_case_table_names            | 1
| max_allowed_packet                | 1048576
| max_binlog_cache_size             | 4294967295
| max_binlog_size                   | 1073741824
| max_connect_errors                | 10
| max_connections                   | 100
| max_delayed_threads               | 20
| max_error_count                   | 64
| max_heap_table_size               | 16777216
| max_insert_delayed_threads        | 20
| max_join_size                     | 4294967295
| max_length_for_sort_data          | 1024
| max_relay_log_size                | 0
| max_seeks_for_key                 | 4294967295
| max_sort_length                   | 1024
| max_tmp_tables                    | 32
| max_user_connections              | 0
| max_write_lock_count              | 4294967295
| myisam_data_pointer_size          | 4
| myisam_max_extra_sort_file_size   | 107374182400
| myisam_max_sort_file_size         | 107374182400
| myisam_recover_options            | OFF
| myisam_repair_threads             | 1
| myisam_sort_buffer_size           | 8388608
| named_pipe                        | OFF
| net_buffer_length                 | 16384
| net_read_timeout                  | 30
| net_retry_count                   | 10
| net_write_timeout                 | 60
| new                               | OFF
| old_passwords                     | OFF
| open_files_limit                  | 622
| optimizer_prune_level             | 1
| optimizer_search_depth            | 62
| pid_file                          | /usr/local/mysql/gigan.pid
| port                              | 3306
| preload_buffer_size               | 32768
| protocol_version                  | 10
| query_alloc_block_size            | 8192
| query_cache_limit                 | 1048576
| query_cache_min_res_unit          | 4096
| query_cache_size                  | 0
| query_cache_type                  | ON
| query_cache_wlock_invalidate      | OFF
| query_prealloc_size               | 8192
| range_alloc_block_size            | 2048
| read_buffer_size                  | 61440
| read_only                         | OFF
| read_rnd_buffer_size              | 258048
| relay_log_purge                   | ON
| rpl_recovery_rank                 | 0
| secure_auth                       | OFF
| shared_memory                     | OFF
| shared_memory_base_name           | MYSQL
| server_id                         | 0
| skip_external_locking             | ON
| skip_networking                   | OFF
| skip_show_database                | OFF
| slave_net_timeout                 | 3600
```

```
| slow_launch_time                | 2
| socket                          | /tmp/mysql.sock
| sort_buffer_size                | 217080
| sql_mode                        |
| storage_engine                  | MyISAM
| sync_binlog                     | 0
| sync_frm                        | ON
| system_time_zone                | E. Australia Standard Time
| table_cache                     | 256
| table_type                      | MyISAM
| thread_cache_size               | 0
| thread_stack                    | 196608
| time_format                     | %H:%i:%s
| time_zone                       | SYSTEM
| tmp_table_size                  | 5242880
| tmpdir                          | /tmp/
| transaction_alloc_block_size    | 8192
| transaction_prealloc_size       | 4096
| tx_isolation                    | REPEATABLE-READ
| updatable_views_with_limit      | YES
| version                         | 5.0.2-alpha-max
| version_bdb                     | Sleepycat Software: Berkeley DB 4.1.24: (Decem
| version_comment                 | Source distribution
| wait_timeout                    | 28800
+---------------------------------+---------------------------------------------
```

Most system variables are described here. Variables with no version indicated have been present since at least MySQL 3.22. `InnoDB` system variables are listed at Section 15.5, “`InnoDB` Startup Options”.

Values for buffer sizes, lengths, and stack sizes are given in bytes unless otherwise specified.

Information on tuning these variables can be found in Section 7.5.2, “Tuning Server Parameters”.

- `ansi_mode`

  This is `ON` if **mysqld** was started with `--ansi`. See Section 1.5.3, “Running MySQL in ANSI Mode”. This variable was added in MySQL 3.23.6 and removed in 3.23.41. See the description for `sql_mode`.

- `back_log`

  The number of outstanding connection requests MySQL can have. This comes into play when the main MySQL thread gets very many connection requests in a very short time. It then takes some time (although very little) for the main thread to check the connection and start a new thread. The `back_log` value indicates how many requests can be stacked during this short time before MySQL momentarily stops answering new requests. You need to increase this only if you expect a large number of connections in a short period of time.

  In other words, this value is the size of the listen queue for incoming TCP/IP connections. Your operating system has its own limit on the size of this queue. The manual page for the Unix `listen()` system call should have more details. Check your OS documentation for the maximum value for this variable. Attempting to set `back_log` higher than your operating system limit is ineffective.

- `basedir`

  The MySQL installation base directory. This variable can be set with the `--basedir` option.

- `bdb_cache_size`

The size of the buffer that is allocated for caching indexes and rows for BDB tables. If you don't use BDB tables, you should start **mysqld** with --skip-bdb to not waste memory for this cache. This variable was added in MySQL 3.23.14.

- bdb_home

  The base directory for BDB tables. This should be assigned the same value as the datadir variable. This variable was added in MySQL 3.23.14.

- bdb_log_buffer_size

  The size of the buffer that is allocated for caching indexes and rows for BDB tables. If you don't use BDB tables, you should set this to 0 or start **mysqld** with --skip-bdb to not waste memory for this cache. This variable was added in MySQL 3.23.31.

- bdb_logdir

  The directory where the BDB storage engine writes its log files. This variable can be set with the --bdb-logdir option. This variable was added in MySQL 3.23.14.

- bdb_max_lock

  The maximum number of locks you can have active on a BDB table (10,000 by default). You should increase this if errors such as the following occur when you perform long transactions or when **mysqld** has to examine many rows to calculate a query:

  ```
  bdb: Lock table is out of available locks
  Got error 12 from ...
  ```

  This variable was added in MySQL 3.23.29.

- bdb_shared_data

  This is ON if you are using --bdb-shared-data. This variable was added in MySQL 3.23.29.

- bdb_tmpdir

  The value of the --bdb-tmpdir option. This variable was added in MySQL 3.23.14.

- bdb_version

  See the description for version_bdb.

- binlog_cache_size

  The size of the cache to hold the SQL statements for the binary log during a transaction. A binary log cache is allocated for each client if the server supports any transactional storage engines and, starting from MySQL 4.1.2, if the server has binary log enabled (--log-bin option). If you often use big, multiple-statement transactions, you can increase this to get more performance. The Binlog_cache_use and Binlog_cache_disk_use status variables can be useful for tuning the size of this variable. This variable was added in MySQL 3.23.29. See Section 5.9.4, “The Binary Log”.

- bulk_insert_buffer_size

  MyISAM uses a special tree-like cache to make bulk inserts faster for INSERT ... SELECT, INSERT ... VALUES (...), (...), ..., and LOAD DATA INFILE. This variable limits

the size of the cache tree in bytes per thread. Setting it to 0 disables this optimization. **Note**: This cache is used only when adding data to a non-empty table. The default value is 8MB. This variable was added in MySQL 4.0.3. This variable previously was named `myisam_bulk_insert_tree_size`.

- `character_set`

  The default character set. This variable was added in MySQL 3.23.3, then removed in MySQL 4.1.1 and replaced by the various `character_set_`*`xxx`* variables.

- `character_set_client`

  The character set for statements that arrive from the client. This variable was added in MySQL 4.1.1.

- `character_set_connection`

  The character set used for literals that do not have a character set introducer and for number-to-string conversion. This variable was added in MySQL 4.1.1.

- `character_set_database`

  The character set used by the default database. The server sets this variable whenever the default database changes. If there is no default database, the variable has the same value as `character_set_server`. This variable was added in MySQL 4.1.1.

- `character_set_results`

  The character set used for returning query results to the client. This variable was added in MySQL 4.1.1.

- `character_set_server`

  The server default character set. This variable was added in MySQL 4.1.1.

- `character_set_system`

  The character set used by the server for storing identifiers. The value is always `utf8`. This variable was added in MySQL 4.1.1.

- `character_sets`

  The supported character sets. This variable was added in MySQL 3.23.15 and removed in MySQL 4.1.1. (Use `SHOW CHARACTER SET` for a list of character sets.)

- `character_sets_dir`

  The directory where character sets are installed. This variable was added in MySQL 4.1.2.

- `collation_connection`

  The collation of the connection character set. This variable was added in MySQL 4.1.1.

- `collation_database`

  The collation used by the default database. The server sets this variable whenever the default database changes. If there is no default database, the variable has the same value as `collation_server`. This variable was added in MySQL 4.1.1.

- `collation_server`

The server default collation. This variable was added in MySQL 4.1.1.

- `concurrent_insert`

  If `ON` (the default), MySQL allows `INSERT` and `SELECT` statements to run concurrently for `MyISAM` tables that have no free blocks in the middle. You can turn this option off by starting **mysqld** with `--safe` or `--skip-new`. This variable was added in MySQL 3.23.7.

- `connect_timeout`

  The number of seconds the **mysqld** server waits for a connect packet before responding with `Bad handshake`.

- `convert_character_set`

  The current character set mapping that was set by `SET CHARACTER SET`. This variable was removed in MySQL 4.1.

- `datadir`

  The MySQL data directory. This variable can be set with the `--datadir` option.

- `default_week_format`

  The default mode value to use for the `WEEK()` function. This variable is available as of MySQL 4.0.14.

- `delay_key_write`

  This option applies only to `MyISAM` tables. It can have one of the following values to affect handling of the `DELAY_KEY_WRITE` table option that can be used in `CREATE TABLE` statements.

Option	Description
`OFF`	`DELAYED_KEY_WRITE` is ignored.
`ON`	MySQL honors the `DELAY_KEY_WRITE` option for `CREATE TABLE`. This is the default value.
`ALL`	All new opened tables are treated as if they were created with the `DELAY_KEY_WRITE` option enabled.

  If `DELAY_KEY_WRITE` is enabled, this means that the key buffer for tables with this option are not flushed on every index update, but only when a table is closed. This speeds up writes on keys a lot, but if you use this feature, you should add automatic checking of all `MyISAM` tables by starting the server with the `--myisam-recover` option (for example, `--myisam-recover=BACKUP,FORCE`). See Section 5.2.1, “**mysqld** Command-Line Options” and Section 14.1.1, “`MyISAM` Startup Options”.

  Note that `--external-locking` doesn't offer any protection against index corruption for tables that use delayed key writes.

This variable was added in MySQL 3.23.8.

- `delayed_insert_limit`

  After inserting `delayed_insert_limit` delayed rows, the `INSERT DELAYED` handler thread checks whether there are any `SELECT` statements pending. If so, it allows them to execute before continuing to insert delayed rows.

- `delayed_insert_timeout`

  How long an `INSERT DELAYED` handler thread should wait for `INSERT` statements before terminating.

- `delayed_queue_size`

  This is a per-table limit on the number of rows to queue when handling `INSERT DELAYED` statements. If the queue becomes full, any client that issues an `INSERT DELAYED` statement waits until there is room in the queue again.

- `expire_logs_days`

  The number of days for automatic binary log removal. The default is 0, which means ``no automatic removal''. Possible removals happen at startup and at binary log rotation. This variable was added in MySQL 4.1.0.

- `flush`

  This is `ON` if you have started **mysqld** with the `--flush` option. This variable was added in MySQL 3.22.9.

- `flush_time`

  If this is set to a non-zero value, all tables are closed every `flush_time` seconds to free up resources and sync unflushed data to disk. We recommend this option only on Windows 9x or Me, or on systems with minimal resources available. This variable was added in MySQL 3.22.18.

- `ft_boolean_syntax`

  The list of operators supported by boolean full-text searches performed using `IN BOOLEAN MODE`. This variable was added in MySQL 4.0.1. See Section 12.6.1, “Boolean Full-Text Searches”.

  The default variable value is `'+ -><()~*:""&|'`. The rules for changing the value are as follows:

  - Operator function is determined by position within the string.
  - The replacement value must be 14 characters.
  - Each character must be an ASCII non-alphanumeric character.
  - Either the first or second character must be a space.
  - No duplicates are allowed except the phrase quoting operators in positions 11 and 12. These two characters are not required to be the same, but they are the only two that may be.
  - Positions 10, 13, and 14 (which by default are set to `':'`, `'&'`, and `'|'`) are reserved for future extensions.

- `ft_max_word_len`

  The maximum length of the word to be included in a `FULLTEXT` index. This variable was added in MySQL 4.0.0.

  Note: `FULLTEXT` indexes must be rebuilt after changing this variable. Use `REPAIR TABLE` *`tbl_name`* `QUICK`.

- `ft_min_word_len`

  The minimum length of the word to be included in a `FULLTEXT` index. This variable was added in MySQL 4.0.0.

  Note: `FULLTEXT` indexes must be rebuilt after changing this variable. Use `REPAIR TABLE` *`tbl_name`* `QUICK`.

- `ft_query_expansion_limit`

  The number of top matches to use for full-text searches performed using `WITH QUERY EXPANSION`. This variable was added in MySQL 4.1.1.

- `ft_stopword_file`

  The file from which to read the list of stopwords for full-text searches. All the words from the file are used; comments are *not* honored. By default, a built-in list of stopwords is used (as defined in the `myisam/ft_static.c` file). Setting this variable to the empty string (`''`) disables stopword filtering. This variable was added in MySQL 4.0.10.

  Note: `FULLTEXT` indexes must be rebuilt after changing this variable. Use `REPAIR TABLE` *`tbl_name`* `QUICK`.

- `group_concat_max_len`

  The maximum allowed result length for the `GROUP_CONCAT()` function. This variable was added in MySQL 4.1.0.

- `have_archive`

  `YES` if **mysqld** supports `ARCHIVE` tables, `NO` if not. This variable was added in MySQL 4.1.3.

- `have_bdb`

  `YES` if **mysqld** supports `BDB` tables. `DISABLED` if `--skip-bdb` is used. This variable was added in MySQL 3.23.30.

- `have_compress`

  Whether the `zlib` compression library is available to the server. If not, the `COMPRESS()` and `UNCOMPRESS()` functions cannot be used. This variable was added in MySQL 4.1.1.

- `have_crypt`

  Whether the `crypt()` system call is available to the server. If not, the `CRYPT()` function cannot be used. This variable was added in MySQL 4.0.10.

- `have_csv`

  `YES` if **mysqld** supports `ARCHIVE` tables, `NO` if not. This variable was added in MySQL 4.1.4.

- `have_example_engine`

YES if **mysqld** supports EXAMPLE tables, NO if not. This variable was added in MySQL 4.1.4.

- have_geometry

  Whether the server supports spatial data types. This variable was added in MySQL 4.1.3.

- have_innodb

  YES if **mysqld** supports InnoDB tables. DISABLED if --skip-innodb is used. This variable was added in MySQL 3.23.37.

- have_isam

  YES if **mysqld** supports ISAM tables. DISABLED if --skip-isam is used. This variable was added in MySQL 3.23.30.

- have_ndbcluster

  YES if **mysqld** supports NDB Cluster tables. DISABLED if --skip-ndbcluster is used. This variable was added in MySQL 4.1.2.

- have_openssl

  YES if **mysqld** supports SSL (encryption) of the client/server protocol. This variable was added in MySQL 3.23.43.

- have_query_cache

  YES if **mysqld** supports the query cache. This variable was added in MySQL 4.0.2.

- have_raid

  YES if **mysqld** supports the RAID option. This variable was added in MySQL 3.23.30.

- have_rtree_keys

  Whether RTREE indexes are available. (These are used for spatial indexed in MyISAM tables.) This variable was added in MySQL 4.1.3.

- have_symlink

  Whether symbolic link support is enabled. This is required on Unix for support of the DATA DIRECTORY and INDEX DIRECTORY table options.

  This variable was added in MySQL 4.0.0.

- init_connect

  A string to be executed by the server for each client that connects. The string consists of one or more SQL statements. To specify multiple statements, separate them by semicolon characters. For example, each client begins by default with autocommit mode enabled. There is no global server variable to specify that autocommit should be disabled by default, but init_connect can be used to

achieve the same effect:

```
SET GLOBAL init_connect='SET AUTOCOMMIT=0';
```

This variable can also be set on the command line or in an option file. To set the variable as just shown using an option file, include these lines:

```
[mysqld]
init_connect='SET AUTOCOMMIT=0'
```

Note that the content of `init_connect` is not executed for users having the `SUPER` privilege; this is in case that content has been wrongly set (contains a wrong query, for example with a syntax error), thus making all connections fail. Not executing it for `SUPER` users enables those to open a connection and fix `init_connect`. This variable was added in MySQL 4.1.2.

- `init_file`

  The name of the file specified with the `--init-file` option when you start the server. This is a file containing SQL statements that you want the server to execute when it starts. Each statement must be on a single line and should not include comments. This variable was added in MySQL 3.23.2.

- `init_slave`

  This variable is similar to `init_connect`, but is a string to be executed by a slave server each time the SQL thread starts. The format of the string is the same as for the `init_connect` variable. This variable was added in MySQL 4.1.2.

- `innodb_`*`xxx`*

  The `InnoDB` system variables are listed at Section 15.5, “`InnoDB` Startup Options”.

- `interactive_timeout`

  The number of seconds the server waits for activity on an interactive connection before closing it. An interactive client is defined as a client that uses the `CLIENT_INTERACTIVE` option to `mysql_real_connect()`. See also `wait_timeout`.

- `join_buffer_size`

  The size of the buffer that is used for full joins (joins that do not use indexes). Normally the best way to get fast joins is to add indexes. Increase the value of `join_buffer_size` to get a faster full join when adding indexes is not possible. One join buffer is allocated for each full join between two tables. For a complex join between several tables for which indexes are not used, multiple join buffers might be necessary.

- `key_buffer_size`

  Index blocks for `MyISAM` and `ISAM` tables are buffered and are shared by all threads. `key_buffer_size` is the size of the buffer used for index blocks. The key buffer is also known as the key cache.

  The maximum allowable setting for `key_buffer_size` is 4GB. The effective maximum size

might be less, depending on your available physical RAM and per-process RAM limits imposed by your operating system or hardware platform.

Increase the value to get better index handling (for all reads and multiple writes) to as much as you can afford. Using a value that is 25% of total memory on a machine that mainly runs MySQL is quite common. However, if you make the value too large (for example, more than 50% of your total memory) your system might start to page and become extremely slow. MySQL relies on the operating system to perform filesystem caching for data reads, so you must leave some room for the filesystem cache.

For even more speed when writing many rows at the same time, use `LOCK TABLES`. See Section 13.4.5, "`LOCK TABLES` and `UNLOCK TABLES` Syntax".

You can check the performance of the key buffer by issuing a `SHOW STATUS` statement and examining the `Key_read_requests`, `Key_reads`, `Key_write_requests`, and `Key_writes` status variables. See Section 13.5.4, "`SHOW` Syntax".

The `Key_reads/Key_read_requests` ratio should normally be less than 0.01. The `Key_writes/Key_write_requests` ratio is usually near 1 if you are using mostly updates and deletes, but might be much smaller if you tend to do updates that affect many rows at the same time or if you are using the `DELAY_KEY_WRITE` table option.

The fraction of the key buffer in use can be determined using `key_buffer_size` in conjunction with the `Key_blocks_unused` status variable and the buffer block size. From MySQL 4.1.1 on, the buffer block size is available from the `key_cache_block_size` server variable. The fraction of the buffer in use is:

```
1 - ((Key_blocks_unused * key_cache_block_size) / key_buffer_size)
```

This value is an approximation because some space in the key buffer may be allocated internally for administrative structures.

Before MySQL 4.1.1, key cache blocks are 1024 bytes, and before MySQL 4.1.2, `Key_blocks_unused` is unavailable. The `Key_blocks_used` variable can be used as follows to determine the fraction of the key buffer in use:

```
(Key_blocks_used * 1024) / key_buffer_size
```

However, `Key_blocks_used` indicates the maximum number of blocks that have ever been in use at once, so this formula does not necessary represent the current fraction of the buffer that is in use.

As of MySQL 4.1, it is possible to create multiple MyISAM key caches. The size limit of 4GB applies to each cache individually, not as a group. See Section 7.4.6, "The `MyISAM` Key Cache".

- `key_cache_age_threshold`

  This value controls the demotion of buffers from the hot sub-chain of a key cache to the warm sub-chain. Lower values cause demotion to happen more quickly. The minimum value is 100. The default value is 300. This variable was added in MySQL 4.1.1. See Section 7.4.6, "The `MyISAM` Key Cache".

- `key_cache_block_size`

  The size in bytes of blocks in the key cache. The default value is 1024. This variable was added in

MySQL 4.1.1. See Section 7.4.6, “The `MyISAM` Key Cache”.

- `key_cache_division_limit`

  The division point between the hot and warm sub-chains of the key cache buffer chain. The value is the percentage of the buffer chain to use for the warm sub-chain. Allowable values range from 1 to 100. The default value is 100. This variable was added in MySQL 4.1.1. See Section 7.4.6, “The `MyISAM` Key Cache”.

- `language`

  The language used for error messages.

- `large_file_support`

  Whether **mysqld** was compiled with options for large file support. This variable was added in MySQL 3.23.28.

- `large_pages`

  Indicates whether large page support is enabled. This variable was added in MySQL 5.0.3.

- `license`

  The type of license the server has. This variable was added in MySQL 4.0.19.

- `local_infile`

  Whether `LOCAL` is supported for `LOAD DATA INFILE` statements. This variable was added in MySQL 4.0.3.

- `locked_in_memory`

  Whether **mysqld** was locked in memory with `--memlock`. This variable was added in MySQL 3.23.25.

- `log`

  Whether logging of all queries to the general query log is enabled. See Section 5.9.2, “The General Query Log”.

- `log_bin`

  Whether the binary log is enabled. This variable was added in MySQL 3.23.14. See Section 5.9.4, “The Binary Log”.

- `log_error`

  The location of the error log. This variable was added in MySQL 4.0.10.

- `log_slave_updates`

  Whether updates received by a slave server from a master server should be logged to the slave's own binary log. Binary logging must be enabled on the slave for this to have any effect. This variable was added in MySQL 3.23.17. See Section 6.8, “Replication Startup Options”.

- `log_slow_queries`

  Whether slow queries should be logged. ``Slow'' is determined by the value of the

`long_query_time` variable. This variable was added in MySQL 4.0.2. See Section 5.9.5, “The Slow Query Log”.

- `log_update`

  Whether the update log is enabled. This variable was added in MySQL 3.22.18. Note that the binary log is preferable to the update log, which is unavailable as of MySQL 5.0. See Section 5.9.3, “The Update Log”.

- `log_warnings`

  Whether to produce additional warning messages. This variable was added in MySQL 4.0.3. It is enabled by default as of MySQL 4.0.19 and 4.1.2. As of MySQL 4.0.21 and 4.1.3, aborted connections are not logged to the error log unless the value is greater than 1.

- `long_query_time`

  If a query takes longer than this many seconds, the `Slow_queries` status variable is incremented. If you are using the `--log-slow-queries` option, the query is logged to the slow query log file. This value is measured in real time, not CPU time, so a query that is under the threshold on a lightly loaded system might be above the threshold on a heavily loaded one. See Section 5.9.5, “The Slow Query Log”.

- `low_priority_updates`

  If set to `1`, all `INSERT`, `UPDATE`, `DELETE`, and `LOCK TABLE WRITE` statements wait until there is no pending `SELECT` or `LOCK TABLE READ` on the affected table. This variable previously was named `sql_low_priority_updates`. It was added in MySQL 3.22.5.

- `lower_case_file_system`

  This variable indicates whether the filesystem where the data directory is located has case insensitive filenames. `ON` means filenames are case insensitive, `OFF` means they are case sensitive. This variable was added in MySQL 4.0.19.

- `lower_case_table_names`

  If set to 1, table names are stored in lowercase on disk and table name comparisons are not case sensitive. This variable was added in MySQL 3.23.6. If set to 2 (new in 4.0.18), table names are stored as given but compared in lowercase. From MySQL 4.0.2, this option also applies to database names. From 4.1.1, it also applies to table aliases. See Section 9.2.2, “Identifier Case Sensitivity”.

  You should *not* set this variable to 0 if you are running MySQL on a system that does not have case-sensitive filenames (such as Windows or Mac OS X). New in 4.0.18: If this variable is not set at startup and the filesystem on which the data directory is located does not have case-sensitive filenames, MySQL automatically sets `lower_case_table_names` to 2.

- `max_allowed_packet`

  The maximum size of one packet or any generated/intermediate string.

  The packet message buffer is initialized to `net_buffer_length` bytes, but can grow up to `max_allowed_packet` bytes when needed. This value by default is small, to catch big (possibly wrong) packets.

  You must increase this value if you are using big `BLOB` columns or long strings. It should be as big as the biggest `BLOB` you want to use. The protocol limit for `max_allowed_packet` is 16MB before MySQL 4.0 and 1GB thereafter.

- `max_binlog_cache_size`

  If a multiple-statement transaction requires more than this amount of memory, you get the error `Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage`. This variable was added in MySQL 3.23.29.

- `max_binlog_size`

  If a write to the binary log exceeds the given value, rotate the binary logs. You cannot set this variable to more than 1GB or to less than 4096 bytes. (The minimum before MYSQL 4.0.14 is 1024 bytes.) The default value is 1GB. This variable was added in MySQL 3.23.33.

  Note if you are using transactions: A transaction is written in one chunk to the binary log, hence it is never split between several binary logs. Therefore, if you have big transactions, you might see binary logs bigger than `max_binlog_size`.

  If `max_relay_log_size` is 0, the value of `max_binlog_size` applies to relay logs as well. `max_relay_log_size` was added in MySQL 4.0.14.

- `max_connect_errors`

  If there are more than this number of interrupted connections from a host, that host is blocked from further connections. You can unblock blocked hosts with the `FLUSH HOSTS` statement.

- `max_connections`

  The number of simultaneous client connections allowed. Increasing this value increases the number of file descriptors that **mysqld** requires. See Section 7.4.8, “How MySQL Opens and Closes Tables” for comments on file descriptor limits. Also see Section A.2.6, “`Too many connections`”.

- `max_delayed_threads`

  Don't start more than this number of threads to handle `INSERT DELAYED` statements. If you try to insert data into a new table after all `INSERT DELAYED` threads are in use, the row is inserted as if the `DELAYED` attribute wasn't specified. If you set this to 0, MySQL never creates a thread to handle `DELAYED` rows; in effect, this disables `DELAYED` entirely. This variable was added in MySQL 3.23.0.

- `max_error_count`

  The maximum number of error, warning, and note messages to be stored for display by `SHOW ERRORS` or `SHOW WARNINGS`. This variable was added in MySQL 4.1.0.

- `max_heap_table_size`

  This variable sets the maximum size to which `MEMORY` (`HEAP`) tables are allowed to grow. The value of the variable is used to calculate `MEMORY` table `MAX_ROWS` values. Setting this variable has no effect on any existing `MEMORY` table, unless the table is re-created with a statement such as `CREATE TABLE` or `TRUNCATE TABLE`, or altered with `ALTER TABLE`. This variable was added in MySQL 3.23.0.

- `max_insert_delayed_threads`

  This variable is a synonym for `max_delayed_threads`. It was added in MySQL 4.0.19.

- `max_join_size`

  Don't allow `SELECT` statements that probably need to examine more than `max_join_size` rows

(for single-table statements) or row combinations (for multiple-table statements) or that are likely to do more than `max_join_size` disk seeks. By setting this value, you can catch `SELECT` statements where keys are not used properly and that would probably take a long time. Set it if your users tend to perform joins that lack a `WHERE` clause, that take a long time, or that return millions of rows.

Setting this variable to a value other than `DEFAULT` resets the `SQL_BIG_SELECTS` value to `0`. If you set the `SQL_BIG_SELECTS` value again, the `max_join_size` variable is ignored.

If a query result is in the query cache, no result size check is performed, because the result has previously been computed and it does not burden the server to send it to the client.

This variable previously was named `sql_max_join_size`.

- `max_length_for_sort_data`

  The cutoff on the size of index values that determines which `filesort` algorithm to use. See Section 7.2.10, "How MySQL Optimizes `ORDER BY`". This variable was added in MySQL 4.1.1

- `max_relay_log_size`

  If a write by a replication slave to its relay log exceeds the given value, rotate the relay log. This variable enables you to put different size constraints on relay logs and binary logs. However, setting the variable to 0 makes MySQL use `max_binlog_size` for both binary logs and relay logs. You must set `max_relay_log_size` to between 4096 bytes and 1GB (inclusive), or to 0. The default value is 0. This variable was added in MySQL 4.0.14. See Section 6.3, "Replication Implementation Details".

- `max_seeks_for_key`

  Limit the assumed maximum number of seeks when looking up rows based on a key. The MySQL optimizer assumes that no more than this number of key seeks are required when searching for matching rows in a table by scanning a key, regardless of the actual cardinality of the key (see Section 13.5.4.11, "`SHOW INDEX` Syntax"). By setting this to a low value (100?), you can force MySQL to prefer keys instead of table scans.

  This variable was added in MySQL 4.0.14.

- `max_sort_length`

  The number of bytes to use when sorting `BLOB` or `TEXT` values. Only the first `max_sort_length` bytes of each value are used; the rest are ignored.

- `max_tmp_tables`

  The maximum number of temporary tables a client can keep open at the same time. (This option doesn't yet do anything.)

- `max_user_connections`

  The maximum number of simultaneous connections allowed to any given MySQL account. A value of 0 means ``no limit.'' This variable was added in MySQL 3.23.34.

  Before MySQL 5.0.3, this variable has only a global form. Beginning with MySQL 5.0.3, it also has a read-only session form. The session variable has the same value as the global variable unless the current account has a non-zero `MAX_USER_CONNECTIONS` resource limit. In that case, the session value reflects the account limit.

- `max_write_lock_count`

After this many write locks, allow some read locks to run in between. This variable was added in MySQL 3.23.7.

- `multi_read_range`

  Specifies the maximum number of ranges to send to a storage engine during range selects. The default value is 256. Sending multiple ranges to an engine is a feature that can improve the performance of certain selects dramatically, particularly for `NDBCLUSTER`. This engine needs to send the range requests to all nodes, and sending many of those requests at once reduces the communication costs significantly. This variable was added in MySQL 5.0.3.

- `myisam_data_pointer_size`

  The default pointer size in bytes, to be used by `CREATE TABLE` for `MyISAM` tables when no `MAX_ROWS` option is specified. This variable cannot be less than 2 or larger than 8. The default value is 4. This variable was added in MySQL 4.1.2. See Section A.2.11, "`The table is full`".

- `myisam_max_extra_sort_file_size`

  If the temporary file used for fast `MyISAM` index creation would be larger than using the key cache by the amount specified here, prefer the key cache method. This is mainly used to force long character keys in large tables to use the slower key cache method to create the index. This variable was added in MySQL 3.23.37. **Note**: The value is given in megabytes before 4.0.3 and in bytes thereafter.

- `myisam_max_sort_file_size`

  The maximum size of the temporary file MySQL is allowed to use while re-creating a `MyISAM` index (during `REPAIR TABLE`, `ALTER TABLE`, or `LOAD DATA INFILE`). If the file size would be bigger than this value, the index is created using the key cache instead, which is slower. This variable was added in MySQL 3.23.37. **Note**: The value is given in megabytes before 4.0.3 and in bytes thereafter.

- `myisam_recover_options`

  The value of the `--myisam-recover` option. This variable was added in MySQL 3.23.36.

- `myisam_repair_threads`

  If this value is greater than 1, `MyISAM` table indexes are created in parallel (each index in its own thread) during the `Repair by sorting` process. The default value is 1. **Note**: Multi-threaded repair is still *alpha* quality code. This variable was added in MySQL 4.0.13.

- `myisam_sort_buffer_size`

  The buffer that is allocated when sorting `MyISAM` indexes during a `REPAIR TABLE` or when creating indexes with `CREATE INDEX` or `ALTER TABLE`. This variable was added in MySQL 3.23.16.

- `named_pipe`

  On Windows, indicates whether the server supports connections over named pipes. This variable was added in MySQL 3.23.50.

- `net_buffer_length`

  The communication buffer is reset to this size between queries. This should not normally be changed, but if you have very little memory, you can set it to the expected length of SQL statements sent by clients. If statements exceed this length, the buffer is automatically enlarged, up to

`max_allowed_packet` bytes.

- `net_read_timeout`

  The number of seconds to wait for more data from a connection before aborting the read. When the server is reading from the client, `net_read_timeout` is the timeout value controlling when to abort. When the server is writing to the client, `net_write_timeout` is the timeout value controlling when to abort. See also `slave_net_timeout`. This variable was added in MySQL 3.23.20.

- `net_retry_count`

  If a read on a communication port is interrupted, retry this many times before giving up. This value should be set quite high on FreeBSD because internal interrupts are sent to all threads. This variable was added in MySQL 3.23.7.

- `net_write_timeout`

  The number of seconds to wait for a block to be written to a connection before aborting the write. See also `net_read_timeout`. This variable was added in MySQL 3.23.20.

- `new`

  This variable is used in MySQL 4.0 to turn on some 4.1 behaviors. This variable was added in MySQL 4.0.12.

- `old_passwords`

  Whether the server should use pre-4.1-style passwords for MySQL user accounts. This variable was added in MySQL 4.1.1.

- `open_files_limit`

  The number of files that the operating system allows **mysqld** to open. This is the real value allowed by the system and might be different from the value you gave **mysqld** as a startup option. The value is 0 on systems where MySQL can't change the number of open files. This variable was added in MySQL 3.23.20.

- `optimizer_prune_level`

  Controls the heuristics applied during query optimization to prune less-promising partial plans from the optimizer search space. A value of 0 disables heuristics so that the optimizer performs an exhaustive search. A value of 1 causes the optimizer to prune plans based on the number of rows retrieved by intermediate plans. This variable was added in MySQL 5.0.1.

- `optimizer_search_depth`

  The maximum depth of search performed by the query optimizer. Values larger than the number of relations in a query result in better query plans, but take longer to generate an execution plan for a query. Values smaller than the number of relations in a query return an execution plan quicker, but the resulting plan may be far from being optimal. If set to 0, the system automatically picks a reasonable value. If set to the maximum number of tables used in a query plus 2, the optimizer switches to the original algorithm used before MySQL 5.0.1 that performs an exhaustive search. This variable was added in MySQL 5.0.1.

- `pid_file`

  The pathname of the process ID (PID) file. This variable can be set with the `--pid-file` option. This variable was added in MySQL 3.23.23.

- `port`

  The port on which the server listens for TCP/IP connections. This variable can be set with the `--port` option.

- `preload_buffer_size`

  The size of the buffer that is allocated when preloading indexes. This variable was added in MySQL 4.1.1.

- `protocol_version`

  The version of the client/server protocol used by the MySQL server. This variable was added in MySQL 3.23.18.

- `query_alloc_block_size`

  The allocation size of memory blocks that are allocated for objects created during query parsing and execution. If you have problems with memory fragmentation, it might help to increase this a bit. This variable was added in MySQL 4.0.16.

- `query_cache_limit`

  Don't cache results that are bigger than this. The default value is 1MB. This variable was added in MySQL 4.0.1.

- `query_cache_min_res_unit`

  The minimum size for blocks allocated by the query cache. The default value is 4KB. Tuning information for this variable is given in Section 5.11.3, “Query Cache Configuration”. This variable is present from MySQL 4.1.

- `query_cache_size`

  The amount of memory allocated for caching query results. The default value is 0, which disables the query cache. Note that this amount of memory is allocated even if `query_cache_type` is set to 0. This variable was added in MySQL 4.0.1.

- `query_cache_type`

  Set query cache type. Setting the `GLOBAL` value sets the type for all clients that connect thereafter. Individual clients can set the `SESSION` value to affect their own use of the query cache.

Option	Description
`0` or `OFF`	Don't cache or retrieve results. Note that this does not deallocate the query cache buffer. To do that, you should set `query_cache_size` to 0.
`1` or `ON`	Cache all query results except for those that begin with `SELECT SQL_NO_CACHE`.
`2` or `DEMAND`	Cache results only for queries that begin with `SELECT SQL_CACHE`.

  This variable was added in MySQL 4.0.3.

- `query_cache_wlock_invalidate`

  Normally, when one client acquires a `WRITE` lock on a `MyISAM` table, other clients are not blocked from issuing queries for the table if the query results are present in the query cache. Setting this vari-

able to 1 causes acquisition of a `WRITE` lock for a table to invalidate any queries in the query cache that refer to the table. This forces other clients that attempt to access the table to wait while the lock is in effect. This variable was added in MySQL 4.0.19.

- `query_prealloc_size`

  The size of the persistent buffer used for query parsing and execution. This buffer is not freed between queries. If you are running complex queries, a larger `query_prealloc_size` value might be helpful in improving performance, because it can reduce the need for the server to perform memory allocation during query execution operations.

  This variable was added in MySQL 4.0.16.

- `range_alloc_block_size`

  The size of blocks that are allocated when doing range optimization. This variable was added in MySQL 4.0.16.

- `read_buffer_size`

  Each thread that does a sequential scan allocates a buffer of this size for each table it scans. If you do many sequential scans, you might want to increase this value. This variable was added in MySQL 4.0.3. Previously, it was named `record_buffer`.

- `read_only`

  When the variable is set to `ON` for a replication slave server, it causes the slave to allow no updates except from slave threads or from users with the `SUPER` privilege. This can be useful to ensure that a slave server accepts no updates from clients. This variable was added in MySQL 4.0.14.

- `relay_log_purge`

  Disables or enables automatic purging of relay logs as soon as they are not needed any more. The default value is 1 (enabled). This variable was added in MySQL 4.1.1.

- `read_rnd_buffer_size`

  When reading rows in sorted order after a sort, the rows are read through this buffer to avoid disk seeks. Setting the variable to a large value can improve `ORDER BY` performance by a lot. However, this is a buffer allocated for each client, so you should not set the global variable to a large value. Instead, change the session variable only from within those clients that need to run large queries. This variable was added in MySQL 4.0.3. Previously, it was named `record_rnd_buffer`.

- `safe_show_database`

  Don't show databases for which the user has no database or table privileges. This can improve security if you're concerned about people being able to see what databases other users have. See also `skip_show_database`.

  This variable was removed in MySQL 4.0.5. Instead, use the `SHOW DATABASES` privilege to control access by MySQL accounts to database names.

- `secure_auth`

  If the MySQL server has been started with the `--secure-auth` option, it blocks connections from all accounts that have passwords stored in the old (pre-4.1) format. In that case, the value of this variable is `ON`, otherwise it is `OFF`.

You should enable this option if you want to prevent all usage of passwords in old format (and hence insecure communication over the network). This variable was added in MySQL 4.1.1.

Server startup fails with an error if this option is enabled and the privilege tables are in pre-4.1 format.

When used as a client-side option, the client refuses to connect to a server if the server requires a password in old format for the client account.

- `server_id`

  The value of the `--server-id` option. It is used for master and slave replication servers. This variable was added in MySQL 3.23.26.

- `shared_memory`

  Whether or not the server allows shared-memory connections. Currently, only Windows servers support this. This variable was added in MySQL 4.1.1.

- `shared_memory_base_name`

  Indicates whether or not the server allows shared-memory connections, and sets the identifier for the shared memory. This is useful when running multiple MYSQL instances on a single physical machine. Currently, only Windows servers support this. This variable was added in MySQL 4.1.0.

- `skip_external_locking`

  This is `OFF` if **mysqld** uses external locking. This variable was added in MySQL 4.0.3. Previously, it was named `skip_locking`.

- `skip_networking`

  This is `ON` if the server allows only local (non-TCP/IP) connections. On Unix, local connections use a Unix socket file. On Windows, local connections use a named pipe or shared memory. On NetWare, only TCP/IP connections are supported, so do not set this variable to `ON`. This variable was added in MySQL 3.22.23.

- `skip_show_database`

  This prevents people from using the `SHOW DATABASES` statement if they don't have the `SHOW DATABASES` privilege. This can improve security if you're concerned about people being able to see what databases other users have. See also `safe_show_database`. This variable was added in MySQL 3.23.4. As of MySQL 4.0.2, its effect also depends on the `SHOW DATABASES` privilege: If the variable value is `ON`, the `SHOW DATABASES` statement is allowed only to users who have the `SHOW DATABASES` privilege, and the statement displays all database names. If the value is `OFF`, `SHOW DATABASES` is allowed to all users, but displays each database name only if the user has the `SHOW DATABASES` privilege or some privilege for the database.

- `slave_compressed_protocol`

  Whether to use compression of the slave/master protocol if both the slave and the master support it. This variable was added in MySQL 4.0.3.

- `slave_net_timeout`

  The number of seconds to wait for more data from a master/slave connection before aborting the read. This variable was added in MySQL 3.23.40.

- `slow_launch_time`

  If creating a thread takes longer than this many seconds, the server increments the `Slow_launch_threads` status variable. This variable was added in MySQL 3.23.15.

- `socket`

  On Unix, this is the Unix socket file used for local client connections. On Windows, this is the name of the named pipe used for local client connections.

- `sort_buffer_size`

  Each thread that needs to do a sort allocates a buffer of this size. Increase this value for faster `ORDER BY` or `GROUP BY` operations. See Section A.4.4, “Where MySQL Stores Temporary Files”.

- `sql_mode`

  The current server SQL mode. This variable was added in MySQL 3.23.41. It can be set dynamically as of MySQL 4.1.1. See Section 5.2.2, “The Server SQL Mode”.

- `sql_slave_skip_counter`

  The number of events from the master that a slave server should skip. It was added in MySQL 3.23.33.

- `storage_engine`

  This variable is a synonym for `table_type`. It was added in MySQL 4.1.2.

- `sync_binlog`

  If positive, the MySQL server synchronizes its binary log to disk (`fdatasync()`) after every `sync_binlog`'th write to this binary log. Note that there is one write to the binary log per statement if in autocommit mode, and otherwise one write per transaction. The default value is 0 which does no sync'ing to disk. A value of 1 is the safest choice, because in case of crash you lose at most one statement/transaction from the binary log; but it is also the slowest choice (unless the disk has a battery-backed cache, which makes sync'ing very fast). This variable was added in MySQL 4.1.3.

- `sync_frm`

  This was added as a command-line option in MySQL 4.0.18, and is also a settable global variable since MySQL 4.1.3. If set to 1, when a non-temporary table is created it synchronizes its `.frm` file to disk (`fdatasync()`); this is slower but safer in case of crash. Default is 1.

- `system_time_zone`

  The server system time zone. When the server begins executing, it inherits a time zone setting from the machine defaults, possibly modified by the environment of the account used for running the server or the startup script. The value is used to set `system_time_zone`. Typically the time zone is specified by the `TZ` environment variable. It also can be specified using the `--timezone` option of the **mysqld_safe** script. This variable was added in MySQL 4.1.3.

- `table_cache`

  The number of open tables for all threads. Increasing this value increases the number of file descriptors that **mysqld** requires. You can check whether you need to increase the table cache by checking the `Opened_tables` status variable. See Section 5.2.4, “Server Status Variables”. If the value of `Opened_tables` is large and you don't do `FLUSH TABLES` a lot (which just forces all tables to be closed and reopened), then you should increase the value of the `table_cache` vari-

able.

For more information about the table cache, see Section 7.4.8, “How MySQL Opens and Closes Tables”.

- `table_type`

  The default table type (storage engine). To set the table type at server startup, use the `--default-table-type` option. This variable was added in MySQL 3.23.0. See Section 5.2.1, “**mysqld** Command-Line Options”.

- `thread_cache_size`

  How many threads the server should cache for reuse. When a client disconnects, the client's threads are put in the cache if there are fewer than `thread_cache_size` threads there. Requests for threads are satisfied by reusing threads taken from the cache if possible, and only when the cache is empty is a new thread created. This variable can be increased to improve performance if you have a lot of new connections. (Normally this doesn't give a notable performance improvement if you have a good thread implementation.) By examining the difference between the `Connections` and `Threads_created` status variables (see Section 5.2.4, “Server Status Variables” for details) you can see how efficient the thread cache is. This variable was added in MySQL 3.23.16.

- `thread_concurrency`

  On Solaris, **mysqld** calls `thr_setconcurrency()` with this value. This function allows applications to give the threads system a hint about the desired number of threads that should be run at the same time. This variable was added in MySQL 3.23.7.

- `thread_stack`

  The stack size for each thread. Many of the limits detected by the `crash-me` test are dependent on this value. The default is large enough for normal operation. See Section 7.1.4, “The MySQL Benchmark Suite”.

- `time_zone`

  The current time zone. The initial value of this is `'SYSTEM'` (use the value of `system_time_zone`), but can be specified explicitly at server startup time with the `--default-time-zone` option. This variable was added in MySQL 4.1.3.

- `timezone`

  The time zone for the server. This is set from the `TZ` environment variable when **mysqld** is started. The time zone also can be set by giving a `--timezone` argument to **mysqld_safe**. This variable was added in MySQL 3.23.15. As of MySQL 4.1.3, it is obsolete and has been replaced by the `system_time_zone` variable. See Section A.4.6, “Time Zone Problems”.

- `tmp_table_size`

  If an in-memory temporary table exceeds this size, MySQL automatically converts it to an on-disk `MyISAM` table. Increase the value of `tmp_table_size` if you do many advanced `GROUP BY` queries and you have lots of memory.

- `tmpdir`

  The directory used for temporary files and temporary tables. Starting from MySQL 4.1, this variable

can be set to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters (`':'`) on Unix and semicolon characters (`';'`) on Windows, NetWare, and OS/2.

This feature can be used to spread the load between several physical disks. If the MySQL server is acting as a replication slave, you should not set `tmpdir` to point to a directory on a memory-based filesystem or to a directory that is cleared when the server host restarts. A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or `LOAD DATA INFILE` operations. If files in the temporary file directory are lost when the server restarts, replication fails.

This variable was added in MySQL 3.22.4.

- `transaction_alloc_block_size`

  The allocation size of memory blocks that are allocated for storing queries that are part of a transaction to be stored in the binary log when doing a commit. This variable was added in MySQL 4.0.16.

- `transaction_prealloc_size`

  The size of the persistent buffer for `transaction_alloc_blocks` that is not freed between queries. By making this big enough to fit all queries in a common transaction, you can avoid a lot of `malloc()` calls. This variable was added in MySQL 4.0.16.

- `tx_isolation`

  The default transaction isolation level. This variable was added in MySQL 4.0.3.

- `updatable_views_with_limit`

  This variable controls whether updates can be made using a view that does not contain a primary key in the underlying table, if the update contains a `LIMIT` clause. (Such updates often are generated by GUI tools.) An update is an `UPDATE` or `DELETE` statement. Primary key here means a `PRIMARY KEY`, or a `UNIQUE` index in which no column can contain `NULL`.

  The variable can have two values:

  - `1` or `YES`: Issue a warning only (not an error message). This is the default value.
  - `0` or `NO`: Prohibit the update.

  This variable was added in MySQL 5.0.2.

- `version`

  The version number for the server.

- `version_bdb`

  The `BDB` storage engine version. This variable was added in MySQL 3.23.31 with the name `bdb_version` and renamed to `version_bdb` in MySQL 4.1.1.

- `version_comment`

  The **configure** script has a `--with-comment` option that allows a comment to be specified when building MySQL. This variable contains the value of that comment. This variable was added in MySQL 4.0.17.

- `version_compile_machine`

The type of machine MySQL was built on. This variable was added in MySQL 4.1.1.

- `version_compile_os`

  The type of operating system MySQL was built on. This variable was added in MySQL 4.0.19.

- `wait_timeout`

  The number of seconds the server waits for activity on a non-interactive connection before closing it.

  On thread startup, the session `wait_timeout` value is initialized from the global `wait_timeout` value or from the global `interactive_timeout` value, depending on the type of client (as defined by the `CLIENT_INTERACTIVE` connect option to `mysql_real_connect()`). See also `interactive_timeout`.

5.2.3.1. Dynamic System Variables

Beginning with MySQL 4.0.3, many server system variables are dynamic and can be set at runtime using `SET GLOBAL` or `SET SESSION`. You can also select their values using `SELECT`. See Section 9.4, “System Variables”.

The following table shows the full list of all dynamic system variables. The last column indicates for each variable whether `GLOBAL` or `SESSION` (or both) apply.

Variable Name	Value Type	Type
`autocommit`	boolean	`SESSION`
`big_tables`	boolean	`SESSION`
`binlog_cache_size`	numeric	`GLOBAL`
`bulk_insert_buffer_size`	numeric	`GLOBAL` | `SESSION`
`character_set_client`	string	`GLOBAL` | `SESSION`
`character_set_connection`	string	`GLOBAL` | `SESSION`
`character_set_results`	string	`GLOBAL` | `SESSION`
`character_set_server`	string	`GLOBAL` | `SESSION`
`collation_connection`	string	`GLOBAL` | `SESSION`
`collation_server`	string	`GLOBAL` | `SESSION`
`concurrent_insert`	boolean	`GLOBAL`
`connect_timeout`	numeric	`GLOBAL`
`convert_character_set`	string	`GLOBAL` | `SESSION`
`default_week_format`	numeric	`GLOBAL` | `SESSION`
`delay_key_write`	`OFF` | `ON` | `ALL`	`GLOBAL`
`delayed_insert_limit`	numeric	`GLOBAL`
`delayed_insert_timeout`	numeric	`GLOBAL`
`delayed_queue_size`	numeric	`GLOBAL`
`error_count`	numeric	`SESSION`
`expire_logs_days`	numeric	`GLOBAL`
`flush`	boolean	`GLOBAL`
`flush_time`	numeric	`GLOBAL`

foreign_key_checks	boolean	SESSION
ft_boolean_syntax	numeric	GLOBAL
group_concat_max_len	numeric	GLOBAL | SESSION
identity	numeric	SESSION
innodb_autoextend_increment	numeric	GLOBAL
innodb_concurrency_tickets	numeric	GLOBAL
innodb_max_dirty_pages_pct	numeric	GLOBAL
innodb_max_purge_lag	numeric	GLOBAL
innodb_sync_spin_loops	numeric	GLOBAL
innodb_table_locks	boolean	GLOBAL | SESSION
innodb_thread_concurrency	numeric GLOBAL	
innodb_thread_sleep_delay	numeric GLOBAL	
insert_id	boolean	SESSION
interactive_timeout	numeric	GLOBAL | SESSION
join_buffer_size	numeric	GLOBAL | SESSION
key_buffer_size	numeric	GLOBAL
last_insert_id	numeric	SESSION
local_infile	boolean	GLOBAL
log_warnings	numeric	GLOBAL
long_query_time	numeric	GLOBAL | SESSION
low_priority_updates	boolean	GLOBAL | SESSION
max_allowed_packet	numeric	GLOBAL | SESSION
max_binlog_cache_size	numeric	GLOBAL
max_binlog_size	numeric	GLOBAL
max_connect_errors	numeric	GLOBAL
max_connections	numeric	GLOBAL
max_delayed_threads	numeric	GLOBAL
max_error_count	numeric	GLOBAL | SESSION
max_heap_table_size	numeric	GLOBAL | SESSION
max_insert_delayed_threads	numeric	GLOBAL
max_join_size	numeric	GLOBAL | SESSION
max_relay_log_size	numeric	GLOBAL
max_seeks_for_key	numeric	GLOBAL | SESSION
max_sort_length	numeric	GLOBAL | SESSION
max_tmp_tables	numeric	GLOBAL | SESSION
max_user_connections	numeric	GLOBAL
max_write_lock_count	numeric	GLOBAL
multi_read_range	numeric	GLOBAL | SESSION
myisam_data_pointer_size	numeric	GLOBAL
myis- am_max_extra_sort_file_size	numeric	GLOBAL | SESSION
myisam_max_sort_file_size	numeric	GLOBAL | SESSION

myisam_repair_threads	numeric	GLOBAL | SESSION
myisam_sort_buffer_size	numeric	GLOBAL | SESSION
net_buffer_length	numeric	GLOBAL | SESSION
net_read_timeout	numeric	GLOBAL | SESSION
net_retry_count	numeric	GLOBAL | SESSION
net_write_timeout	numeric	GLOBAL | SESSION
old_passwords	numeric	GLOBAL | SESSION
optimizer_prune_level	numeric	GLOBAL | SESSION
optimizer_search_depth	numeric	GLOBAL | SESSION
preload_buffer_size	numeric	GLOBAL | SESSION
query_alloc_block_size	numeric	GLOBAL | SESSION
query_cache_limit	numeric	GLOBAL
query_cache_size	numeric	GLOBAL
query_cache_type	enumeration	GLOBAL | SESSION
query_cache_wlock_invalidate	boolean	GLOBAL | SESSION
query_prealloc_size	numeric	GLOBAL | SESSION
range_alloc_block_size	numeric	GLOBAL | SESSION
read_buffer_size	numeric	GLOBAL | SESSION
read_only	numeric	GLOBAL
read_rnd_buffer_size	numeric	GLOBAL | SESSION
rpl_recovery_rank	numeric	GLOBAL
safe_show_database	boolean	GLOBAL
secure_auth	boolean	GLOBAL
server_id	numeric	GLOBAL
slave_compressed_protocol	boolean	GLOBAL
slave_net_timeout	numeric	GLOBAL
slave_transaction_retries	numeric	GLOBAL
slow_launch_time	numeric	GLOBAL
sort_buffer_size	numeric	GLOBAL | SESSION
sql_auto_is_null	boolean	SESSION
sql_big_selects	boolean	SESSION
sql_big_tables	boolean	SESSION
sql_buffer_result	boolean	SESSION
sql_log_bin	boolean	SESSION
sql_log_off	boolean	SESSION
sql_log_update	boolean	SESSION
sql_low_priority_updates	boolean	GLOBAL | SESSION
sql_max_join_size	numeric	GLOBAL | SESSION
sql_mode	enumeration	GLOBAL | SESSION
sql_notes	boolean	SESSION
sql_quote_show_create	boolean	SESSION
sql_safe_updates	boolean	SESSION

sql_select_limit	numeric	SESSION
sql_slave_skip_counter	numeric	GLOBAL
updatable_views_with_limit	enumeration	GLOBAL | SESSION
sql_warnings	boolean	SESSION
sync_binlog	numeric	GLOBAL
sync_frm	boolean	GLOBAL
storage_engine	enumeration	GLOBAL | SESSION
table_cache	numeric	GLOBAL
table_type	enumeration	GLOBAL | SESSION
thread_cache_size	numeric	GLOBAL
time_zone	string	GLOBAL | SESSION
timestamp	boolean	SESSION
tmp_table_size	enumeration	GLOBAL | SESSION
transaction_alloc_block_size	numeric	GLOBAL | SESSION
transaction_prealloc_size	numeric	GLOBAL | SESSION
tx_isolation	enumeration	GLOBAL | SESSION
unique_checks	boolean	SESSION
wait_timeout	numeric	GLOBAL | SESSION
warning_count	numeric	SESSION

Variables that are marked as ``string'' take a string value. Variables that are marked as ``numeric'' take a numeric value. Variables that are marked as ``boolean'' can be set to 0, 1, ON or OFF. Variables that are marked as ``enumeration'' normally should be set to one of the available values for the variable, but can also be set to the number that corresponds to the desired enumeration value. For enumeration-valued system variables, the first enumeration value corresponds to 0. This differs from ENUM columns, for which the first enumeration value corresponds to 1.

5.2.4. Server Status Variables

The server maintains many status variables that provide information about its operations. You can view these variables and their values by using the SHOW STATUS statement:

```
mysql> SHOW STATUS;
+---------------------------+------------+
| Variable_name             | Value      |
+---------------------------+------------+
| Aborted_clients           | 0          |
| Aborted_connects          | 0          |
| Bytes_received            | 155372598  |
| Bytes_sent                | 1176560426 |
| Connections               | 30023      |
| Created_tmp_disk_tables   | 0          |
| Created_tmp_files         | 60         |
| Created_tmp_tables        | 8340       |
| Delayed_errors            | 0          |
| Delayed_insert_threads    | 0          |
| Delayed_writes            | 0          |
| Flush_commands            | 1          |
| Handler_delete            | 462604     |
| Handler_read_first        | 105881     |
| Handler_read_key          | 27820558   |
```

```
| Handler_read_next         | 390681754  |
| Handler_read_prev         | 6022500    |
| Handler_read_rnd          | 30546748   |
| Handler_read_rnd_next     | 246216530  |
| Handler_update            | 16945404   |
| Handler_write             | 60356676   |
| Key_blocks_used           | 14955      |
| Key_read_requests         | 96854827   |
| Key_reads                 | 162040     |
| Key_write_requests        | 7589728    |
| Key_writes                | 3813196    |
| Max_used_connections      | 0          |
| Not_flushed_delayed_rows  | 0          |
| Not_flushed_key_blocks    | 0          |
| Open_files                | 2          |
| Open_streams              | 0          |
| Open_tables               | 1          |
| Opened_tables             | 44600      |
| Qcache_free_blocks        | 36         |
| Qcache_free_memory        | 138488     |
| Qcache_hits               | 79570      |
| Qcache_inserts            | 27087      |
| Qcache_lowmem_prunes      | 3114       |
| Qcache_not_cached         | 22989      |
| Qcache_queries_in_cache   | 415        |
| Qcache_total_blocks       | 912        |
| Questions                 | 2026873    |
| Select_full_join          | 0          |
| Select_full_range_join    | 0          |
| Select_range              | 99646      |
| Select_range_check        | 0          |
| Select_scan               | 30802      |
| Slave_open_temp_tables    | 0          |
| Slave_running             | OFF        |
| Slow_launch_threads       | 0          |
| Slow_queries              | 0          |
| Sort_merge_passes         | 30         |
| Sort_range                | 500        |
| Sort_rows                 | 30296250   |
| Sort_scan                 | 4650       |
| Table_locks_immediate     | 1920382    |
| Table_locks_waited        | 0          |
| Threads_cached            | 0          |
| Threads_connected         | 1          |
| Threads_created           | 30022      |
| Threads_running           | 1          |
| Uptime                    | 80380      |
+---------------------------+------------+
```

Many status variables are reset to 0 by the `FLUSH STATUS` statement.

The status variables have the following meanings. The `Com_xxx` statement counter variables were added beginning with MySQL 3.23.47. The `Qcache_xxx` query cache variables were added beginning with MySQL 4.0.1. Otherwise, variables with no version indicated have been present since at least MySQL 3.22.

- `Aborted_clients`

  The number of connections that were aborted because the client died without closing the connection properly. See Section A.2.10, “Communication Errors and Aborted Connections”.

- `Aborted_connects`

  The number of tries to connect to the MySQL server that failed. See Section A.2.10, “Communication Errors and Aborted Connections”.

- `Binlog_cache_disk_use`

  The number of transactions that used the temporary binary log cache but that exceeded the value of `binlog_cache_size` and used a temporary file to store statements from the transaction. This variable was added in MySQL 4.1.2.

- `Binlog_cache_use`

  The number of transactions that used the temporary binary log cache. This variable was added in MySQL 4.1.2.

- `Bytes_received`

  The number of bytes received from all clients. This variable was added in MySQL 3.23.7.

- `Bytes_sent`

  The number of bytes sent to all clients. This variable was added in MySQL 3.23.7.

- `Com_`*`xxx`*

  The number of times each *`xxx`* statement has been executed. There is one status variable for each type of statement. For example, `Com_delete` and `Com_insert` count `DELETE` and `INSERT` statements.

- `Connections`

  The number of connection attempts (successful or not) to the MySQL server.

- `Created_tmp_disk_tables`

  The number of temporary tables on disk created automatically by the server while executing statements. This variable was added in MySQL 3.23.24.

- `Created_tmp_files`

  How many temporary files **mysqld** has created. This variable was added in MySQL 3.23.28.

- `Created_tmp_tables`

  The number of in-memory temporary tables created automatically by the server while executing statements. If `Created_tmp_disk_tables` is big, you may want to increase the `tmp_table_size` value to cause temporary tables to be memory-based instead of disk-based.

- `Delayed_errors`

  The number of rows written with `INSERT DELAYED` for which some error occurred (probably `duplicate key`).

- `Delayed_insert_threads`

  The number of `INSERT DELAYED` handler threads in use.

- `Delayed_writes`

The number of `INSERT DELAYED` rows written.

- `Flush_commands`

  The number of executed `FLUSH` statements.

- `Handler_commit`

  The number of internal `COMMIT` statements. This variable was added in MySQL 4.0.2.

- `Handler_discover`

  The MySQL server can ask the `NDB Cluster` storage engine if it knows about a table with a given name. This is called discovery. `Handler_discover` indicates the number of time tables have been discovered. This variable was added in MySQL 4.1.2.

- `Handler_delete`

  The number of times a row was deleted from a table.

- `Handler_read_first`

  The number of times the first entry was read from an index. If this is high, it suggests that the server is doing a lot of full index scans; for example, `SELECT col1 FROM foo`, assuming that `col1` is indexed.

- `Handler_read_key`

  The number of requests to read a row based on a key. If this is high, it is a good indication that your queries and tables are properly indexed.

- `Handler_read_next`

  The number of requests to read the next row in key order. This is incremented if you are querying an index column with a range constraint or if you are doing an index scan.

- `Handler_read_prev`

  The number of requests to read the previous row in key order. This read method is mainly used to optimize `ORDER BY ... DESC`. This variable was added in MySQL 3.23.6.

- `Handler_read_rnd`

  The number of requests to read a row based on a fixed position. This is high if you are doing a lot of queries that require sorting of the result. You probably have a lot of queries that require MySQL to scan whole tables or you have joins that don't use keys properly.

- `Handler_read_rnd_next`

  The number of requests to read the next row in the data file. This is high if you are doing a lot of table scans. Generally this suggests that your tables are not properly indexed or that your queries are not written to take advantage of the indexes you have.

- `Handler_rollback`

  The number of internal `ROLLBACK` statements. This variable was added in MySQL 4.0.2.

- `Handler_update`

The number of requests to update a row in a table.

- `Handler_write`

  The number of requests to insert a row in a table.

- `Innodb_buffer_pool_pages_data`

  The number of pages containing data (dirty or clean). Added in MySQL 5.0.2.

- `Innodb_buffer_pool_pages_dirty`

  The number of pages currently dirty. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_pages_flushed`

  The number of buffer pool pages that have been requested to be flushed. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_pages_free`

  The number of free pages. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_pages_latched`

  The number of latched pages in `InnoDB` buffer pool. These are pages currently being read or written or that can't be flushed or removed for some other reason. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_pages_misc`

  The number of pages busy because they have been allocated for administrative overhead such as row locks or the adaptive hash index. This value can also be calculated as `Innodb_buffer_pool_pages_total - Innodb_buffer_pool_pages_free - Innodb_buffer_pool_pages_data`. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_pages_total`

  Total size of buffer pool, in pages. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_read_ahead_rnd`

  The number of ``random'' read-aheads `InnoDB` initiated. This happens when a query is to scan a large portion of a table but in random order. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_read_ahead_seq`

  The number of sequential read-aheads `InnoDB` initiated. This happens when `InnoDB` does a sequential full table scan. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_read_requests`

  The number of logical read requests `InnoDB` has done. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_reads`

  The number of logical reads that `InnoDB` could not satisfy from buffer pool and had to do a single-page read. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_wait_free`

Normally, writes to the `InnoDB` buffer pool happen in the background. However, if it's necessary to read or create a page and no clean pages are available, it's necessary to wait for pages to be flushed first. This counter counts instances of these waits. If the buffer pool size was set properly, this value should be small. Added in MySQL 5.0.2.

- `Innodb_buffer_pool_write_requests`

  The number writes done to the `InnoDB` buffer pool. Added in MySQL 5.0.2.

- `Innodb_data_fsyncs`

  The number of `fsync()` operations so far. Added in MySQL 5.0.2.

- `Innodb_data_pending_fsyncs`

  The current number of pending `fsync()` operations. Added in MySQL 5.0.2.

- `Innodb_data_pending_reads`

  The current number of pending reads. Added in MySQL 5.0.2.

- `Innodb_data_pending_writes`

  The current number of pending writes. Added in MySQL 5.0.2.

- `Innodb_data_read`

  The amount of data read so far, in bytes. Added in MySQL 5.0.2.

- `Innodb_data_reads`

  The total number of data reads. Added in MySQL 5.0.2.

- `Innodb_data_writes`

  The total number of data writes. Added in MySQL 5.0.2.

- `Innodb_data_written`

  The amount of data written so far, in bytes. Added in MySQL 5.0.2.

- `Innodb_dblwr_writes` , `Innodb_dblwr_pages_written`

  The number of doublewrite writes that have been performed and the number of pages that have been written for this purpose. Added in MySQL 5.0.2.

- `Innodb_log_waits`

  The number of waits we had because log buffer was too small and we had to wait for it to be flushed before continuing. Added in MySQL 5.0.2.

- `Innodb_log_write_requests`

  The number of log write requests. Added in MySQL 5.0.2.

- `Innodb_log_writes`

  The number of physical writes to the log file. Added in MySQL 5.0.2.

- `Innodb_os_log_fsyncs`

  The number of fsyncs writes done to the log file. Added in MySQL 5.0.2.

- `Innodb_os_log_pending_fsyncs`

  The number of pending log file fsyncs. Added in MySQL 5.0.2.

- `Innodb_os_log_pending_writes`

  Pending log file writes. Added in MySQL 5.0.2.

- `Innodb_os_log_written`

  The number of bytes written to the log file. Added in MySQL 5.0.2.

- `Innodb_page_size`

  The compiled-in `InnoDB` page size (default 16KB). Many values are counted in pages; the page size allows them to be easily converted to bytes. Added in MySQL 5.0.2.

- `Innodb_pages_created`

  The number of pages created. Added in MySQL 5.0.2.

- `Innodb_pages_read`

  The number of pages read. Added in MySQL 5.0.2.

- `Innodb_pages_written`

  The number of pages written. Added in MySQL 5.0.2.

- `Innodb_row_lock_current_waits`

  The number of row locks currently being waited for. Added in MySQL 5.0.3.

- `Innodb_row_lock_time`

  The total time spent in acquiring row locks, in milliseconds. Added in MySQL 5.0.3.

- `Innodb_row_lock_time_avg`

  The average time to acquire a row lock, in milliseconds. Added in MySQL 5.0.3.

- `Innodb_row_lock_time_max`

  The maximum time to acquire a row lock, in milliseconds. Added in MySQL 5.0.3.

- `Innodb_row_lock_waits`

  The number of times a row lock had to be waited for. Added in MySQL 5.0.3.

- `Innodb_rows_deleted`

  The number of rows deleted from `InnoDB` tables. Added in MySQL 5.0.2.

- `Innodb_rows_inserted`

  The number of rows inserted in `InnoDB` tables. Added in MySQL 5.0.2.

- `Innodb_rows_read`

  The number of rows read from `InnoDB` tables. Added in MySQL 5.0.2.

- `Innodb_rows_updated`

  The number of rows updated in `InnoDB` tables. Added in MySQL 5.0.2.

- `Key_blocks_not_flushed`

  The number of key blocks in the key cache that have changed but haven't yet been flushed to disk. This variable was added in MySQL 4.1.1. It used to be known as `Not_flushed_key_blocks`.

- `Key_blocks_unused`

  The number of unused blocks in the key cache. You can use this value to determine how much of the key cache is in use; see the discussion of `key_buffer_size` in Section 5.2.3, “Server System Variables”. This variable was added in MySQL 4.1.2. Section 5.2.3, “Server System Variables”.

- `Key_blocks_used`

  The number of used blocks in the key cache. This value is a high-water mark that indicates the maximum number of blocks that have ever been in use at one time.

- `Key_read_requests`

  The number of requests to read a key block from the cache.

- `Key_reads`

  The number of physical reads of a key block from disk. If `Key_reads` is big, then your `key_buffer_size` value is probably too small. The cache miss rate can be calculated as `Key_reads`/`Key_read_requests`.

- `Key_write_requests`

  The number of requests to write a key block to the cache.

- `Key_writes`

  The number of physical writes of a key block to disk.

- `Last_query_cost`

  The total cost of the last compiled query as computed by the query optimizer. Useful for comparing the cost of different query plans for the same query. The default value of #1 means that no query has been compiled yet. This variable was added in MySQL 5.0.1.

- `Max_used_connections`

  The maximum number of connections that have been in use simultaneously since the server started.

- `Not_flushed_delayed_rows`

  The number of rows waiting to be written in `INSERT DELAY` queues.

- `Not_flushed_key_blocks`

  The old name for `Key_blocks_not_flushed` before MySQL 4.1.1.

- `Open_files`

  The number of files that are open.

- `Open_streams`

  The number of streams that are open (used mainly for logging).

- `Open_tables`

  The number of tables that are open.

- `Opened_tables`

  The number of tables that have been opened. If `Opened_tables` is big, your `table_cache` value is probably too small.

- `Qcache_free_blocks`

  The number of free memory blocks in query cache.

- `Qcache_free_memory`

  The amount of free memory for query cache.

- `Qcache_hits`

  The number of cache hits.

- `Qcache_inserts`

  The number of queries added to the cache.

- `Qcache_lowmem_prunes`

  The number of queries that were deleted from the cache because of low memory.

- `Qcache_not_cached`

  The number of non-cached queries (not cachable, or due to `query_cache_type`).

- `Qcache_queries_in_cache`

  The number of queries registered in the cache.

- `Qcache_total_blocks`

  The total number of blocks in the query cache.

- `Questions`

  The number of queries that have been sent to the server.

- `Rpl_status`

  The status of failsafe replication (not yet implemented).

- `Select_full_join`

  The number of joins that do not use indexes. If this value is not 0, you should carefully check the in-

dexes of your tables. This variable was added in MySQL 3.23.25.

- `Select_full_range_join`

  The number of joins that used a range search on a reference table. This variable was added in MySQL 3.23.25.

- `Select_range`

  The number of joins that used ranges on the first table. (It's normally not critical even if this is big.) This variable was added in MySQL 3.23.25.

- `Select_range_check`

  The number of joins without keys that check for key usage after each row. (If this is not 0, you should carefully check the indexes of your tables.) This variable was added in MySQL 3.23.25.

- `Select_scan`

  The number of joins that did a full scan of the first table. This variable was added in MySQL 3.23.25.

- `Slave_open_temp_tables`

  The number of temporary tables currently open by the slave SQL thread. This variable was added in MySQL 3.23.29.

- `Slave_running`

  This is `ON` if this server is a slave that is connected to a master. This variable was added in MySQL 3.23.16.

- `Slow_launch_threads`

  The number of threads that have taken more than `slow_launch_time` seconds to create. This variable was added in MySQL 3.23.15.

- `Slow_queries`

  The number of queries that have taken more than `long_query_time` seconds. See Section 5.9.5, “The Slow Query Log”.

- `Sort_merge_passes`

  The number of merge passes the sort algorithm has had to do. If this value is large, you should consider increasing the value of the `sort_buffer_size` system variable. This variable was added in MySQL 3.23.28.

- `Sort_range`

  The number of sorts that were done with ranges. This variable was added in MySQL 3.23.25.

- `Sort_rows`

  The number of sorted rows. This variable was added in MySQL 3.23.25.

- `Sort_scan`

  The number of sorts that were done by scanning the table. This variable was added in MySQL

3.23.25.

- `Ssl_xxx`

  Variables used for SSL connections. These variables were added in MySQL 4.0.0.

- `Table_locks_immediate`

  The number of times that a table lock was acquired immediately. This variable was added as of MySQL 3.23.33.

- `Table_locks_waited`

  The number of times that a table lock could not be acquired immediately and a wait was needed. If this is high, and you have performance problems, you should first optimize your queries, and then either split your table or tables or use replication. This variable was added as of MySQL 3.23.33.

- `Threads_cached`

  The number of threads in the thread cache. This variable was added in MySQL 3.23.17.

- `Threads_connected`

  The number of currently open connections.

- `Threads_created`

  The number of threads created to handle connections. If `Threads_created` is big, you may want to increase the `thread_cache_size` value. The cache hit rate can be calculated as `Threads_created`/`Connections`. This variable was added in MySQL 3.23.31.

- `Threads_running`

  The number of threads that are not sleeping.

- `Uptime`

  The number of seconds the server has been up.

5.3. The MySQL Server Shutdown Process

The server shutdown process can be summarized like this:

1. The shutdown process is initiated
2. The server creates a shutdown thread if necessary
3. The server stops accepting new connections
4. The server terminates current activity
5. Storage engines are shut down or closed
6. The server exits

A more detailed description of the process follows:

1. The shutdown process is initiated.

   Server shutdown can be initiated several ways. For example, a user with the `SHUTDOWN` privilege can execute a **mysqladmin shutdown** command. **mysqladmin** can be used on any platform supported by MySQL. Other operating system-specific shutdown initiation methods are possible as well: The server shuts down on Unix when it receives a `SIGTERM` signal. A server running as a service on Windows shuts down when the services manager tells it to.

2. The server creates a shutdown thread if necessary.

   Depending on how shutdown was initiated, the server might create a thread to handle the shutdown process. If shutdown was requested by a client, a shutdown thread is created. If shutdown is the result of receiving a `SIGTERM` signal, the signal thread might handle shutdown itself, or it might create a separate thread to do so. If the server tries to create a shutdown thread and cannot (for example, if memory is exhausted), it issues a diagnostic message that appears in the error log:

   ```
   Error: Can't create thread to kill server
   ```

3. The server stops accepting new connections.

   To prevent new activity from being initiated during shutdown, the server stops accepting new client connections. It does this by closing the network connections to which it normally listens for connections: the TCP/IP port, the Unix socket file, the Windows named pipe, and shared memory on Windows.

4. The server terminates current activity.

   For each thread that is associated with a client connection, the connection to the client is broken and the thread is marked as killed. Threads die when they notice that they are so marked. Threads for idle connections die quickly. Threads that currently are processing queries check their state periodically and take longer to die. For additional information about thread termination, see Section 13.5.5.3, "`KILL` Syntax", in particular for the instructions about killed `REPAIR TABLE` or `OPTIMIZE TABLE` operations on `MyISAM` tables.

   For threads that have an open transaction, the transaction is rolled back. Note that if a thread is updating a non-transactional table, an operation such as a multiple-row `UPDATE` or `INSERT` may leave the table partially updated, because the operation can terminate before completion.

   If the server is a master replication server, threads associated with currently connected slaves are treated like other client threads. That is, each one is marked as killed and exits when it next checks its state.

   If the server is a slave replication server, the I/O and SQL threads, if active, are stopped before client threads are marked as killed. The SQL thread is allowed to finish its current statement (to avoid causing replication problems) then stops. If the SQL thread was in the middle of a transaction at this point, the transaction is rolled back.

5. Storage engines are shut down or closed.

   At this stage, the table cache is flushed and all open tables are closed.

   Each storage engine performs any actions necessary for tables that it manages. For example, MyISAM flushes any pending index writes for a table. InnoDB flushes its buffer pool to disk, writes the current LSN to the tablespace, and terminates its own internal threads.

6. The server exits.

5.4. General Security Issues

This section describes some general security issues to be aware of and what you can do to make your MySQL installation more secure against attack or misuse. For information specifically about the access control system that MySQL uses for setting up user accounts and checking database access, see Section 5.5, “The MySQL Access Privilege System”.

5.4.1. General Security Guidelines

Anyone using MySQL on a computer connected to the Internet should read this section to avoid the most common security mistakes.

In discussing security, we emphasize the necessity of fully protecting the entire server host (not just the MySQL server) against all types of applicable attacks: eavesdropping, altering, playback, and denial of service. We do not cover all aspects of availability and fault tolerance here.

MySQL uses security based on Access Control Lists (ACLs) for all connections, queries, and other operations that users can attempt to perform. There is also some support for SSL-encrypted connections between MySQL clients and servers. Many of the concepts discussed here are not specific to MySQL at all; the same general ideas apply to almost all applications.

When running MySQL, follow these guidelines whenever possible:

- **Do not ever give anyone (except MySQL `root` accounts) access to the `user` table in the `mysql` database!** This is critical. **The encrypted password is the real password in MySQL.** Anyone who knows the password that is listed in the `user` table and has access to the host listed for the account **can easily log in as that user**.

- Learn the MySQL access privilege system. The `GRANT` and `REVOKE` statements are used for controlling access to MySQL. Do not grant any more privileges than necessary. Never grant privileges to all hosts.

  Checklist:

  - Try `mysql -u root`. If you are able to connect successfully to the server without being asked for a password, you have problems. Anyone can connect to your MySQL server as the MySQL `root` user with full privileges! Review the MySQL installation instructions, paying particular attention to the information about setting a `root` password. See Section 2.9.3, “Securing the Initial MySQL Accounts”.

  - Use the `SHOW GRANTS` statement and check to see who has access to what. Then use the `REVOKE` statement to remove those privileges that are not necessary.

- Do not store any plain-text passwords in your database. If your computer becomes compromised, the intruder can take the full list of passwords and use them. Instead, use `MD5()`, `SHA1()`, or some other one-way hashing function.

- Do not choose passwords from dictionaries. There are special programs to break them. Even passwords like ``xfish98'' are very bad. Much better is ``duag98'' which contains the same word ``fish'' but typed one key to the left on a standard QWERTY keyboard. Another method is to use ``Mhall'' which is taken from the first characters of each word in the sentence ``Mary had a little lamb.'' This is easy to remember and type, but difficult to guess for someone who does not know it.

- Invest in a firewall. This protects you from at least 50% of all types of exploits in any software. Put MySQL behind the firewall or in a demilitarized zone (DMZ).

  Checklist:

  - Try to scan your ports from the Internet using a tool such as `nmap`. MySQL uses port 3306 by default. This port should not be accessible from untrusted hosts. Another simple way to check whether or not your MySQL port is open is to try the following command from some remote machine, where `server_host` is the host on which your MySQL server runs:

    ```
    shell> telnet server_host 3306
    ```

    If you get a connection and some garbage characters, the port is open, and should be closed on your firewall or router, unless you really have a good reason to keep it open. If `telnet` just hangs or the connection is refused, everything is OK; the port is blocked.

- Do not trust any data entered by users of your applications. They can try to trick your code by entering special or escaped character sequences in Web forms, URLs, or whatever application you have built. Be sure that your application remains secure if a user enters something like ``; DROP DATABASE mysql;''. This is an extreme example, but large security leaks and data loss might occur as a result of hackers using similar techniques, if you do not prepare for them.

  A common mistake is to protect only string data values. Remember to check numeric data as well. If an application generates a query such as `SELECT * FROM table WHERE ID=234` when a user enters the value `234`, the user can enter the value `234 OR 1=1` to cause the application to generate the query `SELECT * FROM table WHERE ID=234 OR 1=1`. As a result, the server retrieves every record in the table. This exposes every record and causes excessive server load. The simplest way to protect from this type of attack is to use apostrophes around the numeric constants: `SELECT * FROM table WHERE ID='234'`. If the user enters extra information, it all becomes part of the string. In numeric context, MySQL automatically converts this string to a number and strips any trailing non-numeric characters from it.

  Sometimes people think that if a database contains only publicly available data, it need not be protected. This is incorrect. Even if it is allowable to display any record in the database, you should still protect against denial of service attacks (for example, those that are based on the technique in the preceding paragraph that causes the server to waste resources). Otherwise, your server becomes unresponsive to legitimate users.

  Checklist:

  - Try to enter `''` and `'"'` in all your Web forms. If you get any kind of MySQL error, investigate the problem right away.

  - Try to modify any dynamic URLs by adding `%22` ('"'), `%23` ('#'), and `%27` (`''`) in the URL.

  - Try to modify data types in dynamic URLs from numeric ones to character ones containing characters from previous examples. Your application should be safe against this and similar attacks.

  - Try to enter characters, spaces, and special symbols rather than numbers in numeric fields. Your application should remove them before passing them to MySQL or else generate an error. Passing unchecked values to MySQL is very dangerous!

  - Check data sizes before passing them to MySQL.

  - Consider having your application connect to the database using a different username than the one you use for administrative purposes. Do not give your applications any access privileges they do not need.

- Many application programming interfaces provide a means of escaping special characters in data values. Properly used, this prevents application users from entering values that cause the application to generate statements that have a different effect than you intend:
  - MySQL C API: Use the `mysql_real_escape_string()` API call.
  - MySQL++: Use the `escape` and `quote` modifiers for query streams.
  - PHP: Use the `mysql_escape_string()` function, which is based on the function of the same name in the MySQL C API. Prior to PHP 4.0.3, use `addslashes()` instead.
  - Perl DBI: Use the `quote()` method or use placeholders.
  - Java JDBC: Use a `PreparedStatement` object and placeholders.

  Other programming interfaces might have similar capabilities.
- Do not transmit plain (unencrypted) data over the Internet. This information is accessible to everyone who has the time and ability to intercept it and use it for their own purposes. Instead, use an encrypted protocol such as SSL or SSH. MySQL supports internal SSL connections as of Version 4.0.0. SSH port-forwarding can be used to create an encrypted (and compressed) tunnel for the communication.
- Learn to use the `tcpdump` and `strings` utilities. For most cases, you can check whether MySQL data streams are unencrypted by issuing a command like the following:

  ```
  shell> tcpdump -l -i eth0 -w - src or dst port 3306 | strings
  ```

  (This works under Linux and should work with small modifications under other systems.) Warning: If you do not see plaintext data, this doesn't always mean that the information actually is encrypted. If you need high security, you should consult with a security expert.

5.4.2. Making MySQL Secure Against Attackers

When you connect to a MySQL server, you should use a password. The password is not transmitted in clear text over the connection. Password handling during the client connection sequence was upgraded in MySQL 4.1.1 to be very secure. If you are using an older version of MySQL, or are still using pre-4.1.1-style passwords, the encryption algorithm is less strong and with some effort a clever attacker who can sniff the traffic between the client and the server can crack the password. (See Section 5.5.9, “Password Hashing in MySQL 4.1” for a discussion of the different password handling methods.) If the connection between the client and the server goes through an untrusted network, you should use an SSH tunnel to encrypt the communication.

All other information is transferred as text that can be read by anyone who is able to watch the connection. If you are concerned about this, you can use the compressed protocol (in MySQL 3.22 and above) to make traffic much more difficult to decipher. To make the connection even more secure, you should use SSH to get an encrypted TCP/IP connection between a MySQL server and a MySQL client. You can find an Open Source SSH client at http://www.openssh.org/, and a commercial SSH client at http://www.ssh.com/.

If you are using MySQL 4.0 or newer, you can also use internal OpenSSL support. See Section 5.6.7, “Using Secure Connections”.

To make a MySQL system secure, you should strongly consider the following suggestions:

- Use passwords for all MySQL users. A client program does not necessarily know the identity of the person running it. It is common for client/server applications that the user can specify any username to the client program. For example, anyone can use the **mysql** program to connect as any other person simply by invoking it as `mysql -u other_user db_name` if `other_user` has no password. If all users have a password, connecting using another user's account becomes much more difficult.

  To change the password for a user, use the `SET PASSWORD` statement. It is also possible to update the `user` table in the `mysql` database directly. For example, to change the password of all MySQL accounts that have a username of `root`, do this:

  ```
  shell> mysql -u root
  mysql> UPDATE mysql.user SET Password=PASSWORD('newpwd')
      -> WHERE User='root';
  mysql> FLUSH PRIVILEGES;
  ```

- Don't run the MySQL server as the Unix `root` user. This is very dangerous, because any user with the `FILE` privilege is able to create files as `root` (for example, `~root/.bashrc`). To prevent this, **mysqld** refuses to run as `root` unless that is specified explicitly using a `--user=root` option.

  mysqld can be run as an ordinary unprivileged user instead. You can also create a separate Unix account named `mysql` to make everything even more secure. Use the account only for administering MySQL. To start **mysqld** as another Unix user, add a `user` option that specifies the username to the `[mysqld]` group of the `/etc/my.cnf` option file or the `my.cnf` option file in the server's data directory. For example:

  ```
  [mysqld]
  user=mysql
  ```

  This causes the server to start as the designated user whether you start it manually or by using **mysqld_safe** or **mysql.server**. For more details, see Section A.3.2, “How to Run MySQL as a Normal User”.

  Running **mysqld** as a Unix user other than `root` does not mean that you need to change the `root` username in the `user` table. Usernames for MySQL accounts have nothing to do with usernames for Unix accounts.

- Don't allow the use of symlinks to tables. (This can be disabled with the `--skip-symbolic-links` option.) This is especially important if you run **mysqld** as `root`, because anyone that has write access to the server's data directory then could delete any file in the system! See Section 7.6.1.2, “Using Symbolic Links for Tables on Unix”.

- Make sure that the only Unix user with read or write privileges in the database directories is the user that **mysqld** runs as.

- Don't grant the `PROCESS` or `SUPER` privilege to non-administrative users. The output of **mysqladmin processlist** shows the text of the currently executing queries, so any user who is allowed to execute that command might be able to see if another user issues an `UPDATE user SET password=PASSWORD('not_secure')` query.

  mysqld reserves an extra connection for users who have the `SUPER` privilege (`PROCESS` before MySQL 4.0.2), so that a MySQL `root` user can log in and check server activity even if all normal connections are in use.

  The `SUPER` privilege can be used to terminate client connections, change server operation by changing the value of system variables, and control replication servers.

- Don't grant the `FILE` privilege to non-administrative users. Any user that has this privilege can write a file anywhere in the filesystem with the privileges of the **mysqld** daemon! To make this a bit safer, files generated with `SELECT ... INTO OUTFILE` do not overwrite existing files and are writable by everyone.

  The `FILE` privilege may also be used to read any file that is world-readable or accessible to the Unix user that the server runs as. With this privilege, you can read any file into a database table. This could be abused, for example, by using `LOAD DATA` to load `/etc/passwd` into a table, which then can be displayed with `SELECT`.

- If you don't trust your DNS, you should use IP numbers rather than hostnames in the grant tables. In any case, you should be very careful about creating grant table entries using hostname values that contain wildcards!

- If you want to restrict the number of connections allowed to a single account, you can do so by setting the `max_user_connections` variable in **mysqld**. The `GRANT` statement also supports resource control options for limiting the extent of server use allowed to an account.

5.4.3. Startup Options for mysqld Concerning Security

The following **mysqld** options affect security:

- `--allow-suspicious-udfs`

  This option controls whether user-defined functions that have only an `xxx` symbol for the main function can be loaded. By default, the option is off and only UDFs that have at least one auxiliary symbol can be loaded. This prevents attempts at loading functions from shared object files other than those containing legitimate UDFs. This option was added in MySQL 4.0.24, 4.1.10a, and 5.0.3. See Section 25.2.3.6, “User-defined Function Security Precautions”.

- `--local-infile[={0|1}]`

  If you start the server with `--local-infile=0`, clients cannot use `LOCAL` in `LOAD DATA` statements. See Section 5.4.4, “Security Issues with `LOAD DATA LOCAL`”.

- `--old-passwords`

  Force the server to generate short (pre-4.1) password hashes for new passwords. This is useful for compatibility when the server must support older client programs. See Section 5.5.9, “Password Hashing in MySQL 4.1”.

- `--safe-show-database`

  With this option, the `SHOW DATABASES` statement displays the names of only those databases for which the user has some kind of privilege. As of MySQL 4.0.2, this option is deprecated and doesn't do anything (it is enabled by default), because there is a `SHOW DATABASES` privilege that can be used to control access to database names on a per-account basis. See Section 13.5.1.3, “`GRANT` and `REVOKE` Syntax”.

- `--safe-user-create`

  If this is enabled, a user cannot create new users with the `GRANT` statement unless the user has the `INSERT` privilege for the `mysql.user` table. If you want a user to have the ability to create new users with those privileges that the user has right to grant, you should grant the user the following privilege:

```
mysql> GRANT INSERT(user) ON mysql.user TO 'user_name'@'host_name';
```

This ensures that the user can't change any privilege columns directly, but has to use the `GRANT` statement to give privileges to other users.

- `--secure-auth`

  Disallow authentication for accounts that have old (pre-4.1) passwords. This option is available as of MySQL 4.1.1.

- `--skip-grant-tables`

  This option causes the server not to use the privilege system at all. This gives everyone *full access* to all databases! (You can tell a running server to start using the grant tables again by executing a **mysqladmin flush-privileges** or **mysqladmin reload** command, or by issuing a `FLUSH PRIVILEGES` statement.)

- `--skip-name-resolve`

  Hostnames are not resolved. All `Host` column values in the grant tables must be IP numbers or `localhost`.

- `--skip-networking`

  Don't allow TCP/IP connections over the network. All connections to **mysqld** must be made via Unix socket files. This option is unsuitable when using a MySQL version prior to 3.23.27 with the MIT-pthreads package, because Unix socket files were not supported by MIT-pthreads at that time.

- `--skip-show-database`

  With this option, the `SHOW DATABASES` statement is allowed only to users who have the `SHOW DATABASES` privilege, and the statement displays all database names. Without this option, `SHOW DATABASES` is allowed to all users, but displays each database name only if the user has the `SHOW DATABASES` privilege or some privilege for the database.

5.4.4. Security Issues with `LOAD DATA LOCAL`

The `LOAD DATA` statement can load a file that is located on the server host, or it can load a file that is located on the client host when the `LOCAL` keyword is specified.

There are two potential security issues with supporting the `LOCAL` version of `LOAD DATA` statements:

- The transfer of the file from the client host to the server host is initiated by the MySQL server. In theory, a patched server could be built that would tell the client program to transfer a file of the server's choosing rather than the file named by the client in the `LOAD DATA` statement. Such a server could access any file on the client host to which the client user has read access.

- In a Web environment where the clients are connecting from a Web server, a user could use `LOAD DATA LOCAL` to read any files that the Web server process has read access to (assuming that a user could run any command against the SQL server). In this environment, the client with respect to the MySQL server actually is the Web server, not the program being run by the user connecting to the Web server.

To deal with these problems, we changed how `LOAD DATA LOCAL` is handled as of MySQL 3.23.49 and MySQL 4.0.2 (4.0.13 on Windows):

- By default, all MySQL clients and libraries in binary distributions are compiled with the `--enable-local-infile` option, to be compatible with MySQL 3.23.48 and before.
- If you build MySQL from source but don't use the `--enable-local-infile` option to **configure**, `LOAD DATA LOCAL` cannot be used by any client unless it is written explicitly to invoke `mysql_options(... MYSQL_OPT_LOCAL_INFILE, 0)`. See Section 22.2.3.43, "`mysql_options()`".
- You can disable all `LOAD DATA LOCAL` commands from the server side by starting **mysqld** with the `--local-infile=0` option.
- For the **mysql** command-line client, `LOAD DATA LOCAL` can be enabled by specifying the `--local-infile[=1]` option, or disabled with the `--local-infile=0` option. Similarly, for **mysqlimport**, the `--local` or `-L` option enables local data file loading. In any case, successful use of a local loading operation requires that the server is enabled to allow it.
- If you use `LOAD DATA LOCAL` in Perl scripts or other programs that read the `[client]` group from option files, you can add the `local-infile=1` option to that group. However, to keep this from causing problems for programs that do not understand `local-infile`, specify it using the `loose-` prefix:

  ```
  [client]
  loose-local-infile=1
  ```

  The `loose-` prefix can be used as of MySQL 4.0.2.
- If `LOAD DATA LOCAL INFILE` is disabled, either in the server or the client, a client that attempts to issue such a statement receives the following error message:

  ```
  ERROR 1148: The used command is not allowed with this MySQL version
  ```

5.5. The MySQL Access Privilege System

MySQL has an advanced but non-standard security and privilege system. This section describes how it works.

5.5.1. What the Privilege System Does

The primary function of the MySQL privilege system is to authenticate a user connecting from a given host, and to associate that user with privileges on a database such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE`.

Additional functionality includes the ability to have anonymous users and to grant privileges for MySQL-specific functions such as `LOAD DATA INFILE` and administrative operations.

5.5.2. How the Privilege System Works

The MySQL privilege system ensures that all users may perform only the operations allowed to them. As a user, when you connect to a MySQL server, your identity is determined by *the host from which you connect* and *the username you specify*. When you issue requests after connecting, the system grants priv-

ileges according to your identity and *what you want to do*.

MySQL considers both your hostname and username in identifying you because there is little reason to assume that a given username belongs to the same person everywhere on the Internet. For example, the user `joe` who connects from `office.com` need not be the same person as the user `joe` who connects from `elsewhere.com`. MySQL handles this by allowing you to distinguish users on different hosts that happen to have the same name: You can grant one set of privileges for connections by `joe` from `office.com`, and a different set of privileges for connections by `joe` from `elsewhere.com`.

MySQL access control involves two stages:

- Stage 1: The server checks whether it should allow you to connect.
- Stage 2: Assuming that you can connect, the server checks each statement you issue to see whether you have sufficient privileges to perform it. For example, if you try to select rows from a table in a database or drop a table from the database, the server verifies that you have the `SELECT` privilege for the table or the `DROP` privilege for the database.

If your privileges are changed (either by yourself or someone else) while you are connected, those changes do not necessarily take effect immediately for the next statement you issue. See Section 5.5.7, “When Privilege Changes Take Effect” for details.

The server stores privilege information in the grant tables of the `mysql` database (that is, in the database named `mysql`). The MySQL server reads the contents of these tables into memory when it starts and re-reads them under the circumstances indicated in Section 5.5.7, “When Privilege Changes Take Effect”. Access-control decisions are based on the in-memory copies of the grant tables.

Normally, you manipulate the contents of the grant tables indirectly by using the `GRANT` and `REVOKE` statements to set up accounts and control the privileges available to each one. See Section 13.5.1.3, “`GRANT` and `REVOKE` Syntax”. The discussion here describes the underlying structure of the grant tables and how the server uses their contents when interacting with clients.

The server uses the `user`, `db`, and `host` tables in the `mysql` database at both stages of access control. The columns in these grant tables are shown here:

Table Name	**user**	**db**	**host**
Scope columns	`Host`	`Host`	`Host`
	`User`	`Db`	`Db`
	`Password`	`User`	
Privilege columns	`Select_priv`	`Select_priv`	`Select_priv`
	`Insert_priv`	`Insert_priv`	`Insert_priv`
	`Update_priv`	`Update_priv`	`Update_priv`
	`Delete_priv`	`Delete_priv`	`Delete_priv`
	`Index_priv`	`Index_priv`	`Index_priv`
	`Alter_priv`	`Alter_priv`	`Alter_priv`
	`Create_priv`	`Create_priv`	`Create_priv`
	`Drop_priv`	`Drop_priv`	`Drop_priv`
	`Grant_priv`	`Grant_priv`	`Grant_priv`
	`Create_view_priv`	`Create_view_priv`	`Create_view_priv`
	`Show_view_priv`	`Show_view_priv`	`Show_view_priv`
	`Cre-ate_routine_priv`	`Cre-ate_routine_priv`	

	Al-ter_routine_priv	Al-ter_routine_priv	
	References_priv	References_priv	References_priv
	Reload_priv		
	Shutdown_priv		
	Process_priv		
	File_priv		
	Show_db_priv		
	Super_priv		
	Cre-ate_tmp_table_pr iv	Cre-ate_tmp_table_pr iv	Cre-ate_tmp_table_pr iv
	Lock_tables_priv	Lock_tables_priv	Lock_tables_priv
	Execute_priv		
	Repl_slave_priv		
	Repl_client_priv		
Security columns	ssl_type		
	ssl_cipher		
	x509_issuer		
	x509_subject		
Resource control columns	max_questions		
	max_updates		
	max_connections		
	max_user_connect ions		

The `ssl_type`, `ssl_cipher`, `x509_issuer`, and `x509_subject` columns were added in MySQL 4.0.0.

The `Create_tmp_table_priv`, `Execute_priv`, `Lock_tables_priv`, `Repl_client_priv`, `Repl_slave_priv`, `Show_db_priv`, `Super_priv`, `max_questions`, `max_updates`, and `max_connections` columns were added in MySQL 4.0.2. `Execute_priv` is not operational until MySQL 5.0.3, however.

The `Create_view_priv` and `Show_view_priv` columns were added in MySQL 5.0.1.

The `Create_routine_priv`, `Alter_routine_priv`, and `max_user_connections` columns were added in MySQL 5.0.3.

During the second stage of access control, the server performs request verification to make sure that each client has sufficient privileges for each request that it issues. In addition to the `user`, `db`, and `host` grant tables, the server may also consult the `tables_priv` and `columns_priv` tables for requests that involve tables. The `tables_priv` and `columns_priv` tables provide finer privilege control at the table and column levels. They have the following columns:

Table Name	**tables_priv**	**columns_priv**
Scope columns	Host	Host

	Db	Db
	User	User
	Table_name	Table_name
		Column_name
Privilege columns	Table_priv	Column_priv
	Column_priv	
Other columns	Timestamp	Timestamp
	Grantor	

The Timestamp and Grantor columns currently are unused and are discussed no further here.

For verification of requests that involve stored routines, the server may consult the procs_priv table. This table exists as of MySQL 5.0.3 and has the following columns:

Table Name	**procs_priv**
Scope columns	Host
	Db
	User
	Routine_name
Privilege columns	Proc_priv
Other columns	Timestamp
	Grantor

The Timestamp and Grantor columns currently are unused and are discussed no further here.

Each grant table contains scope columns and privilege columns:

- Scope columns determine the scope of each entry (row) in the tables; that is, the context in which the row applies. For example, a user table row with Host and User values of 'thomas.loc.gov' and 'bob' would be used for authenticating connections made to the server from the host thomas.loc.gov by a client that specifies a username of bob. Similarly, a db table row with Host, User, and Db column values of 'thomas.loc.gov', 'bob' and 'reports' would be used when bob connects from the host thomas.loc.gov to access the reports database. The tables_priv and columns_priv tables contain scope columns indicating tables or table/column combinations to which each row applies. The procs_priv scope columns indicate the store routine to which each row applies.

- Privilege columns indicate which privileges are granted by a table row; that is, what operations can be performed. The server combines the information in the various grant tables to form a complete description of a user's privileges. The rules used to do this are described in Section 5.5.6, “Access Control, Stage 2: Request Verification”.

Scope columns contain strings. They are declared as shown here; the default value for each is the empty string:

Column Name	**Type**
Host	CHAR(60)
User	CHAR(16)

Password	CHAR(16)
Db	CHAR(64)
Table_name	CHAR(64)
Column_name	CHAR(64)
Routine_name	CHAR(64)

Before MySQL 3.23, the Db column is CHAR(32) in some tables and CHAR(60) in others.

For access-checking purposes, comparisons of Host values are case-insensitive. User, Password, Db, and Table_name values are case sensitive. Column_name values are case insensitive in MySQL 3.22.12 or later.

In the user, db, and host tables, each privilege is listed in a separate column that is declared as ENUM('N','Y') DEFAULT 'N'. In other words, each privilege can be disabled or enabled, with the default being disabled.

In the tables_priv, columns_priv, and procs_priv tables, the privilege columns are declared as SET columns. Values in these columns can contain any combination of the privileges controlled by the table:

Table Name	Column Name	Possible Set Elements
tables_priv	Table_priv	'Select', 'Insert', 'Update', 'Delete', 'Create', 'Drop', 'Grant', 'References', 'Index', 'Alter'
tables_priv	Column_priv	'Select', 'Insert', 'Update', 'References'
columns_priv	Column_priv	'Select', 'Insert', 'Update', 'References'
procs_priv	Proc_priv	'Execute', 'Alter Routine', 'Grant'

Briefly, the server uses the grant tables as follows:

- The user table scope columns determine whether to reject or allow incoming connections. For allowed connections, any privileges granted in the user table indicate the user's global (superuser) privileges. These privileges apply to *all* databases on the server.
- The db table scope columns determine which users can access which databases from which hosts. The privilege columns determine which operations are allowed. A privilege granted at the database level applies to the database and to all its tables.
- The host table is used in conjunction with the db table when you want a given db table row to apply to several hosts. For example, if you want a user to be able to use a database from several hosts in your network, leave the Host value empty in the user's db table row, then populate the host table with a row for each of those hosts. This mechanism is described more detail in Section 5.5.6, “Access Control, Stage 2: Request Verification”.

  Note: The host table is not affected by the GRANT and REVOKE statements. Most MySQL installations need not use this table at all.
- The tables_priv and columns_priv tables are similar to the db table, but are more fine-grained: They apply at the table and column levels rather than at the database level. A privilege granted at the table level applies to the table and to all its columns. A privilege granted at the column level applies only to a specific column.

- The procs_priv table applies to stored routines. A privilege granted at the routine level applies only to a single routine.

Administrative privileges (such as RELOAD or SHUTDOWN) are specified only in the user table. This is because administrative operations are operations on the server itself and are not database-specific, so there is no reason to list these privileges in the other grant tables. In fact, to determine whether you can perform an administrative operation, the server need consult only the user table.

The FILE privilege also is specified only in the user table. It is not an administrative privilege as such, but your ability to read or write files on the server host is independent of the database you are accessing.

The **mysqld** server reads the contents of the grant tables into memory when it starts. You can tell it to re-read the tables by issuing a FLUSH PRIVILEGES statement or executing a **mysqladmin flush-privileges** or **mysqladmin reload** command. Changes to the grant tables take effect as indicated in Section 5.5.7, "When Privilege Changes Take Effect".

When you modify the contents of the grant tables, it is a good idea to make sure that your changes set up privileges the way you want. To check the privileges for a given account, use the SHOW GRANTS statement. For example, to determine the privileges that are granted to an account with Host and User values of pc84.example.com and bob, issue this statement:

```
mysql> SHOW GRANTS FOR 'bob'@'pc84.example.com';
```

A useful diagnostic tool is the **mysqlaccess** script, which Yves Carlier has provided for the MySQL distribution. Invoke **mysqlaccess** with the --help option to find out how it works. Note that **mysqlaccess** checks access using only the user, db, and host tables. It does not check table, column, or routine privileges specified in the tables_priv, columns_priv, or procs_priv tables.

For additional help in diagnosing privilege-related problems, see Section 5.5.8, "Causes of Access denied Errors". For general advice on security issues, see Section 5.4, "General Security Issues".

5.5.3. Privileges Provided by MySQL

Information about account privileges is stored in the user, db, host, tables_priv, columns_priv, and procs_priv tables in the mysql database. The MySQL server reads the contents of these tables into memory when it starts and re-reads them under the circumstances indicated in Section 5.5.7, "When Privilege Changes Take Effect". Access-control decisions are based on the in-memory copies of the grant tables.

The names used in the GRANT and REVOKE statements to refer to privileges are shown in the following table, along with the column name associated with each privilege in the grant tables and the context in which the privilege applies. Further information about the meaning of each privilege may be found at Section 13.5.1.3, "GRANT and REVOKE Syntax".

Privilege	Column	Context
CREATE	Create_priv	databases, tables, or indexes
DROP	Drop_priv	databases or tables
GRANT	Grant_priv	databases, tables, or stored routines
REFERENCES	References_priv	databases or tables
ALTER	Alter_priv	tables
DELETE	Delete_priv	tables
INDEX	Index_priv	tables
INSERT	Insert_priv	tables

SELECT	Select_priv	tables
UPDATE	Update_priv	tables
CREATE VIEW	Create_view_priv	views
SHOW VIEW	Show_view_priv	views
ALTER ROUTINE	Alter_routine_priv	stored routines
CREATE ROUTINE	Create_routine_priv	stored routines
EXECUTE	Execute_priv	stored routines
CREATE TEMPORARY TABLES	Create_tmp_table_priv	server administration
FILE	File_priv	file access on server host
LOCK TABLES	Lock_tables_priv	server administration
PROCESS	Process_priv	server administration
RELOAD	Reload_priv	server administration
REPLICATION CLIENT	Repl_client_priv	server administration
REPLICATION SLAVE	Repl_slave_priv	server administration
SHOW DATABASES	Show_db_priv	server administration
SHUTDOWN	Shutdown_priv	server administration
SUPER	Super_priv	server administration

The CREATE TEMPORARY TABLES, EXECUTE, LOCK TABLES, REPLICATION CLIENT, REPLICATION SLAVE, SHOW DATABASES, and SUPER privileges were added in MySQL 4.0.2. (EXECUTE is not operational until MySQL 5.0.3.) CREATE VIEW and SHOW VIEW were added in MySQL 5.0.1. CREATE ROUTINE and ALTER ROUTINE were added in MySQL 5.0.3. To use these privileges when upgrading from an earlier version of MySQL that does not have them, , you must upgrade your grant tables. See Section 2.10.7, “Upgrading the Grant Tables”.

The CREATE and DROP privileges allow you to create new databases and tables, or to drop (remove) existing databases and tables. If you grant the DROP privilege for the mysql database to a user, that user can drop the database in which the MySQL access privileges are stored!

The SELECT, INSERT, UPDATE, and DELETE privileges allow you to perform operations on rows in existing tables in a database.

SELECT statements require the SELECT privilege only if they actually retrieve rows from a table. Some SELECT statements do not access tables and can be executed without permission for any database. For example, you can use the **mysql** client as a simple calculator to evaluate expressions that make no reference to tables:

```
mysql> SELECT 1+1;
mysql> SELECT PI()*2;
```

The INDEX privilege allows you to create or drop (remove) indexes. INDEX applies to existing tables. If you have the CREATE privilege for a table, you can include index definitions in the CREATE TABLE statement.

The ALTER privilege allows you to use ALTER TABLE to change the structure of or rename tables.

The CREATE ROUTINE privilege is needed for creating stored routines (functions and procedures).

ALTER ROUTINE privilege is needed for altering or dropping stored routines, and EXECUTE is needed for executing stored routines.

The GRANT privilege allows you to give to other users those privileges that you yourself possess. It can be used for databases, tables, and stored routines.

The FILE privilege gives you permission to read and write files on the server host using the LOAD DATA INFILE and SELECT ... INTO OUTFILE statements. A user who has the FILE privilege can read any file on the server host that is either world-readable or readable by the MySQL server. (This implies the user can read any file in any database directory, because the server can access any of those files.) The FILE privilege also allows the user to create new files in any directory where the MySQL server has write access. Existing files cannot be overwritten.

The remaining privileges are used for administrative operations. Many of them can be performed by using the **mysqladmin** program or by issuing SQL statements. The following table shows which **mysqladmin** commands each administrative privilege allows you to execute:

Privilege	Commands Permitted to Privilege Holders
RELOAD	flush-hosts, flush-logs, flush-privileges, flush-status, flush-tables, flush-threads, refresh, reload
SHUTDOWN	shutdown
PROCESS	processlist
SUPER	kill

The reload command tells the server to re-read the grant tables into memory. flush-privileges is a synonym for reload. The refresh command closes and reopens the log files and flushes all tables. The other flush-*xxx* commands perform functions similar to refresh, but are more specific and may be preferable in some instances. For example, if you want to flush just the log files, flush-logs is a better choice than refresh.

The shutdown command shuts down the server. This command can be issued only from **mysqladmin**. There is no corresponding SQL statement.

The processlist command displays information about the threads executing within the server (that is, about the statements being executed by clients associated with other accounts). The kill command terminates server threads. You can always display or kill your own threads, but you need the PROCESS privilege to display threads initiated by other users and the SUPER privilege to kill them. See Section 13.5.5.3, "KILL Syntax". Prior to MySQL 4.0.2 when SUPER was introduced, the PROCESS privilege controls the ability to both see and terminate threads for other clients.

The CREATE TEMPORARY TABLES privilege allows the use of the keyword TEMPORARY in CREATE TABLE statements.

The LOCK TABLES privilege allows the use of explicit LOCK TABLES statements to lock tables for which you have the SELECT privilege. This includes the use of write locks, which prevents anyone else from reading the locked table.

The REPLICATION CLIENT privilege allows the use of SHOW MASTER STATUS and SHOW SLAVE STATUS.

The REPLICATION SLAVE privilege should be granted to accounts that are used by slave servers to connect to the current server as their master. Without this privilege, the slave cannot request updates that have been made to databases on the master server.

The SHOW DATABASES privilege allows the account to see database names by issuing the SHOW DATABASE statement. Accounts that do not have this privilege see only databases for which they have some privileges, and cannot use the statement at all if the server was started with the -

`-skip-show-database` option.

It is a good idea in general to grant to an account only those privileges that it needs. You should exercise particular caution in granting the `FILE` and administrative privileges:

- The `FILE` privilege can be abused to read into a database table any files that the MySQL server can read on the server host. This includes all world-readable files and files in the server's data directory. The table can then be accessed using `SELECT` to transfer its contents to the client host.
- The `GRANT` privilege allows users to give their privileges to other users. Two users with different privileges and with the `GRANT` privilege are able to combine privileges.
- The `ALTER` privilege may be used to subvert the privilege system by renaming tables.
- The `SHUTDOWN` privilege can be abused to deny service to other users entirely by terminating the server.
- The `PROCESS` privilege can be used to view the plain text of currently executing queries, including queries that set or change passwords.
- The `SUPER` privilege can be used to terminate other clients or change how the server operates.
- Privileges granted for the `mysql` database itself can be used to change passwords and other access privilege information. Passwords are stored encrypted, so a malicious user cannot simply read them to know the plain text password. However, a user with write access to the `user` table `Password` column can change an account's password, and then connect to the MySQL server using that account.

There are some things that you cannot do with the MySQL privilege system:

- You cannot explicitly specify that a given user should be denied access. That is, you cannot explicitly match a user and then refuse the connection.
- You cannot specify that a user has privileges to create or drop tables in a database but not to create or drop the database itself.

5.5.4. Connecting to the MySQL Server

MySQL client programs generally expect you to specify connection parameters when you want to access a MySQL server:

- The name of the host where the MySQL server is running
- Your username
- Your password

For example, the **mysql** client can be started as follows from a command-line prompt (indicated here by `shell>`):

```
shell> mysql -h host_name -u user_name -pyour_pass
```

Alternate forms of the `-h`, `-u`, and `-p` options are `--host=host_name`, `--user=user_name`, and

`--password=your_pass`. Note that there is *no space* between `-p` or `--password=` and the password following it.

If you use a `-p` or `--password` option but do not specify the password value, the client program prompts you to enter the password. The password is not displayed as you enter it. This is more secure than giving the password on the command line. Any user on your system may be able to see a password specified on the command line by executing a command such as **ps auxww**. See Section 5.6.6, "Keeping Your Password Secure".

MySQL client programs use default values for any connection parameter option that you do not specify:

- The default hostname is `localhost`.
- The default username is `ODBC` on Windows and your Unix login name on Unix.
- No password is supplied if `-p` is missing.

Thus, for a Unix user with a login name of `joe`, all of the following commands are equivalent:

```
shell> mysql -h localhost -u joe
shell> mysql -h localhost
shell> mysql -u joe
shell> mysql
```

Other MySQL clients behave similarly.

You can specify different default values to be used when you make a connection so that you need not enter them on the command line each time you invoke a client program. This can be done in a couple of ways:

- You can specify connection parameters in the `[client]` section of an option file. The relevant section of the file might look like this:

  ```
  [client]
  host=host_name
  user=user_name
  password=your_pass
  ```

  Option files are discussed further in Section 4.3.2, "Using Option Files".

- You can specify some connection parameters using environment variables. The host can be specified for **mysql** using `MYSQL_HOST`. The MySQL username can be specified using `USER` (this is for Windows and NetWare only). The password can be specified using `MYSQL_PWD`, although this is insecure; see Section 5.6.6, "Keeping Your Password Secure". For a list of variables, see Appendix F, *Environment Variables*.

5.5.5. Access Control, Stage 1: Connection Verification

When you attempt to connect to a MySQL server, the server accepts or rejects the connection based on your identity and whether you can verify your identity by supplying the correct password. If not, the server denies access to you completely. Otherwise, the server accepts the connection, then enters Stage 2 and waits for requests.

Your identity is based on two pieces of information:

- The client host from which you connect
- Your MySQL username

Identity checking is performed using the three `user` table scope columns (`Host`, `User`, and `Password`). The server accepts the connection only if the `Host` and `User` columns in some `user` table record match the client hostname and username, and the client supplies the password specified in that record.

`Host` values in the `user` table may be specified as follows:

- A `Host` value may be a hostname or an IP number, or `'localhost'` to indicate the local host.
- You can use the wildcard characters '%' and '_' in `Host` column values. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator. For example, a `Host` value of `'%'` matches any hostname, whereas a value of `'%.mysql.com'` matches any host in the `mysql.com` domain.
- As of MySQL 3.23, for `Host` values specified as IP numbers, you can specify a netmask indicating how many address bits to use for the network number. For example:

  ```
  mysql> GRANT ALL PRIVILEGES ON db.*
      -> TO david@'192.58.197.0/255.255.255.0';
  ```

  This allows `david` to connect from any client host having an IP number `client_ip` for which the following condition is true:

  ```
  client_ip & netmask = host_ip
  ```

  That is, for the `GRANT` statement just shown:

  ```
  client_ip & 255.255.255.0 = 192.58.197.0
  ```

  IP numbers that satisfy this condition and can connect to the MySQL server are those that lie in the range from `192.58.197.0` to `192.58.197.255`.
- A blank `Host` value in a `db` table record means that its privileges should be combined with those in the row in the `host` table that matches the client hostname. The privileges are combined using an AND (intersection) operation, not OR (union). You can find more information about the `host` table in Section 5.5.6, “Access Control, Stage 2: Request Verification”.

  A blank `Host` value in the other grant tables is the same as `'%'`.

Because you can use IP wildcard values in the `Host` column (for example, `'144.155.166.%'` to match every host on a subnet), someone could try to exploit this capability by naming a host `144.155.166.somewhere.com`. To foil such attempts, MySQL disallows matching on hostnames that start with digits and a dot. Thus, if you have a host named something like `1.2.foo.com`, its name never matches the `Host` column of the grant tables. An IP wildcard value can match only IP numbers, not hostnames.

In the `User` column, wildcard characters are not allowed, but you can specify a blank value, which matches any name. If the `user` table row that matches an incoming connection has a blank username, the user is considered to be an anonymous user with no name, not a user with the name that the client actually specified. This means that a blank username is used for all further access checking for the duration

of the connection (that is, during Stage 2).

The `Password` column can be blank. This is not a wildcard and does not mean that any password matches. It means that the user must connect without specifying a password.

Non-blank `Password` values in the `user` table represent encrypted passwords. MySQL does not store passwords in plaintext form for anyone to see. Rather, the password supplied by a user who is attempting to connect is encrypted (using the `PASSWORD()` function). The encrypted password then is used during the connection process when checking whether the password is correct. (This is done without the encrypted password ever traveling over the connection.) From MySQL's point of view, the encrypted password is the REAL password, so you should not give anyone access to it! In particular, don't give non-administrative users read access to the tables in the `mysql` database!

From version 4.1 on, MySQL employs a stronger authentication method that has better password protection during the connection process than in earlier versions. It is secure even if TCP/IP packets are sniffed or the `mysql` database is captured. Password encryption is discussed further in Section 5.5.9, “Password Hashing in MySQL 4.1”.

The following examples show how various combinations of `Host` and `User` values in the `user` table apply to incoming connections:

`Host` **Value**	`User` **Value**	**Connections Matched by Entry**
`'thomas.loc.gov'`	`'fred'`	`fred`, connecting from `thomas.loc.gov`
`'thomas.loc.gov'`	`''`	Any user, connecting from `thomas.loc.gov`
`'%'`	`'fred'`	`fred`, connecting from any host
`'%'`	`''`	Any user, connecting from any host
`'%.loc.gov'`	`'fred'`	`fred`, connecting from any host in the `loc.gov` domain
`'x.y.%'`	`'fred'`	`fred`, connecting from `x.y.net`, `x.y.com`, `x.y.edu`, and so on. (this is probably not useful)
`'144.155.166.177'`	`'fred'`	`fred`, connecting from the host with IP address `144.155.166.177`
`'144.155.166.%'`	`'fred'`	`fred`, connecting from any host in the `144.155.166` class C subnet
`'144.155.166.0/255.255.255.0'`	`'fred'`	Same as previous example

It is possible for the client hostname and username of an incoming connection to match more than one row in the `user` table. The preceding set of examples demonstrates this: Several of the entries shown match a connection from `thomas.loc.gov` by `fred`.

When multiple matches are possible, the server must determine which of them to use. It resolves this issue as follows:

- Whenever the server reads the `user` table into memory, it sorts the entries.
- When a client attempts to connect, the server looks through the entries in sorted order.
- The server uses the first row that matches the client hostname and username.

To see how this works, suppose that the `user` table looks like this:

```
+-----------+----------+-
```

```
| Host      | User     | ...
+-----------+----------+-
| %         | root     | ...
| %         | jeffrey  | ...
| localhost | root     | ...
| localhost |          | ...
+-----------+----------+-
```

When the server reads in the table, it orders the entries with the most-specific `Host` values first. Literal hostnames and IP numbers are the most specific. The pattern `'%'` means ``any host'' and is least specific. Entries with the same `Host` value are ordered with the most-specific `User` values first (a blank `User` value means ``any user'' and is least specific). For the `user` table just shown, the result after sorting looks like this:

```
+-----------+----------+-
| Host      | User     | ...
+-----------+----------+-
| localhost | root     | ...
| localhost |          | ...
| %         | jeffrey  | ...
| %         | root     | ...
+-----------+----------+-
```

When a client attempts to connect, the server looks through the sorted entries and uses the first match found. For a connection from `localhost` by `jeffrey`, two of the entries in the table match: the one with `Host` and `User` values of `'localhost'` and `''`, and the one with values of `'%'` and `'jeffrey'`. The `'localhost'` row appears first in sorted order, so that is the one the server uses.

Here is another example. Suppose that the `user` table looks like this:

```
+----------------+----------+-
| Host           | User     | ...
+----------------+----------+-
| %              | jeffrey  | ...
| thomas.loc.gov |          | ...
+----------------+----------+-
```

The sorted table looks like this:

```
+----------------+----------+-
| Host           | User     | ...
+----------------+----------+-
| thomas.loc.gov |          | ...
| %              | jeffrey  | ...
+----------------+----------+-
```

A connection by `jeffrey` from `thomas.loc.gov` is matched by the first row, whereas a connection by `jeffrey` from `whitehouse.gov` is matched by the second.

It is a common misconception to think that, for a given username, all entries that explicitly name that user are used first when the server attempts to find a match for the connection. This is simply not true. The previous example illustrates this, where a connection from `thomas.loc.gov` by `jeffrey` is first matched not by the row containing `'jeffrey'` as the `User` column value, but by the row with no username. As a result, `jeffrey` is authenticated as an anonymous user, even though he specified a username when connecting.

If you are able to connect to the server, but your privileges are not what you expect, you probably are

being authenticated as some other account. To find out what account the server used to authenticate you, use the `CURRENT_USER()` function. It returns a value in *`user_name@host_name`* format that indicates the `User` and `Host` values from the matching `user` table record. Suppose that `jeffrey` connects and issues the following query:

```
mysql> SELECT CURRENT_USER();
+----------------+
| CURRENT_USER() |
+----------------+
| @localhost     |
+----------------+
```

The result shown here indicates that the matching `user` table row had a blank `User` column value. In other words, the server is treating `jeffrey` as an anonymous user.

The `CURRENT_USER()` function is available as of MySQL 4.0.6. See Section 12.8.3, “Information Functions”. Another thing you can do to diagnose authentication problems is to print out the `user` table and sort it by hand to see where the first match is being made.

5.5.6. Access Control, Stage 2: Request Verification

Once you establish a connection, the server enters Stage 2 of access control. For each request that comes in on the connection, the server determines what operation you want to perform, then checks whether you have sufficient privileges to do so. This is where the privilege columns in the grant tables come into play. These privileges can come from any of the `user`, `db`, `host`, `tables_priv`, or `columns_priv` tables. (You may find it helpful to refer to Section 5.5.2, “How the Privilege System Works”, which lists the columns present in each of the grant tables.)

The `user` table grants privileges that are assigned to you on a global basis and that apply no matter what the current database is. For example, if the `user` table grants you the `DELETE` privilege, you can delete rows from any table in any database on the server host! In other words, `user` table privileges are superuser privileges. It is wise to grant privileges in the `user` table only to superusers such as database administrators. For other users, you should leave the privileges in the `user` table set to `'N'` and grant privileges at more specific levels only. You can grant privileges for particular databases, tables, or columns.

The `db` and `host` tables grant database-specific privileges. Values in the scope columns of these tables can take the following forms:

- The wildcard characters '%' and '_' can be used in the `Host` and `Db` columns of either table. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator. If you want to use either character literally when granting privileges, you must escape it with a backslash. For example, to include '_' character as part of a database name, specify it as '_' in the `GRANT` statement.
- A `'%'` `Host` value in the `db` table means ``any host.'' A blank `Host` value in the `db` table means ``consult the `host` table for further information'' (a process that is described later in this section).
- A `'%'` or blank `Host` value in the `host` table means ``any host.''
- A `'%'` or blank `Db` value in either table means ``any database.''
- A blank `User` value in either table matches the anonymous user.

The server reads in and sorts the `db` and `host` tables at the same time that it reads the `user` table. The server sorts the `db` table based on the `Host`, `Db`, and `User` scope columns, and sorts the `host` table

based on the `Host` and `Db` scope columns. As with the `user` table, sorting puts the most-specific values first and least-specific values last, and when the server looks for matching entries, it uses the first match that it finds.

The `tables_priv` and `columns_priv` tables grant table-specific and column-specific privileges. Values in the scope columns of these tables can take the following form:

- The wildcard characters '%' and '_' can be used in the `Host` column of either table. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator.
- A `'%'` or blank `Host` value in either table means ``any host."
- The `Db`, `Table_name`, and `Column_name` columns cannot contain wildcards or be blank in either table.

The server sorts the `tables_priv` and `columns_priv` tables based on the `Host`, `Db`, and `User` columns. This is similar to `db` table sorting, but simpler because only the `Host` column can contain wildcards.

The request verification process is described here. (If you are familiar with the access-checking source code, you may notice that the description here differs slightly from the algorithm used in the code. The description is equivalent to what the code actually does; it differs only to make the explanation simpler.)

For requests that require administrative privileges such as `SHUTDOWN` or `RELOAD`, the server checks only the `user` table row because that is the only table that specifies administrative privileges. Access is granted if the row allows the requested operation and denied otherwise. For example, if you want to execute **mysqladmin shutdown** but your `user` table row doesn't grant the `SHUTDOWN` privilege to you, the server denies access without even checking the `db` or `host` tables. (They contain no `Shutdown_priv` column, so there is no need to do so.)

For database-related requests (`INSERT`, `UPDATE`, and so on), the server first checks the user's global (superuser) privileges by looking in the `user` table row. If the row allows the requested operation, access is granted. If the global privileges in the `user` table are insufficient, the server determines the user's database-specific privileges by checking the `db` and `host` tables:

1. The server looks in the `db` table for a match on the `Host`, `Db`, and `User` columns. The `Host` and `User` columns are matched to the connecting user's hostname and MySQL username. The `Db` column is matched to the database that the user wants to access. If there is no row for the `Host` and `User`, access is denied.
2. If there is a matching `db` table row and its `Host` column is not blank, that row defines the user's database-specific privileges.
3. If the matching `db` table row's `Host` column is blank, it signifies that the `host` table enumerates which hosts should be allowed access to the database. In this case, a further lookup is done in the `host` table to find a match on the `Host` and `Db` columns. If no `host` table row matches, access is denied. If there is a match, the user's database-specific privileges are computed as the intersection (*not* the union!) of the privileges in the `db` and `host` table entries; that is, the privileges that are `'Y'` in both entries. (This way you can grant general privileges in the `db` table row and then selectively restrict them on a host-by-host basis using the `host` table entries.)

After determining the database-specific privileges granted by the `db` and `host` table entries, the server adds them to the global privileges granted by the `user` table. If the result allows the requested operation, access is granted. Otherwise, the server successively checks the user's table and column privileges in the `tables_priv` and `columns_priv` tables, adds those to the user's privileges, and allows or denies access based on the result.

Expressed in boolean terms, the preceding description of how a user's privileges are calculated may be summarized like this:

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
```

It may not be apparent why, if the global `user` row privileges are initially found to be insufficient for the requested operation, the server adds those privileges to the database, table, and column privileges later. The reason is that a request might require more than one type of privilege. For example, if you execute an `INSERT INTO ... SELECT` statement, you need both the `INSERT` and the `SELECT` privileges. Your privileges might be such that the `user` table row grants one privilege and the `db` table row grants the other. In this case, you have the necessary privileges to perform the request, but the server cannot tell that from either table by itself; the privileges granted by the entries in both tables must be combined.

The `host` table is not affected by the `GRANT` or `REVOKE` statements, so it is unused in most MySQL installations. If you modify it directly, you can use it for some specialized purposes, such as to maintain a list of secure servers. For example, at TcX, the `host` table contains a list of all machines on the local network. These are granted all privileges.

You can also use the `host` table to indicate hosts that are *not* secure. Suppose that you have a machine `public.your.domain` that is located in a public area that you do not consider secure. You can allow access to all hosts on your network except that machine by using `host` table entries like this:

```
+--------------------+----+-
| Host               | Db | ...
+--------------------+----+-
| public.your.domain | %  | ... (all privileges set to 'N')
| %.your.domain      | %  | ... (all privileges set to 'Y')
+--------------------+----+-
```

Naturally, you should always test your entries in the grant tables (for example, by using `SHOW GRANTS` or **mysqlaccess**) to make sure that your access privileges are actually set up the way you think they are.

5.5.7. When Privilege Changes Take Effect

When **mysqld** starts, all grant table contents are read into memory and become effective for access control at that point.

When the server reloads the grant tables, privileges for existing client connections are affected as follows:

- Table and column privilege changes take effect with the client's next request.
- Database privilege changes take effect at the next `USE` *`db_name`* statement.
- Changes to global privileges and passwords take effect the next time the client connects.

If you modify the grant tables using `GRANT`, `REVOKE`, or `SET PASSWORD`, the server notices these changes and reloads the grant tables into memory again immediately.

If you modify the grant tables directly using statements such as `INSERT`, `UPDATE`, or `DELETE`, your changes have no effect on privilege checking until you either restart the server or tell it to reload the

tables. To reload the grant tables manually, issue a `FLUSH PRIVILEGES` statement or execute a **mysqladmin flush-privileges** or **mysqladmin reload** command.

If you change the grant tables directly but forget to reload them, your changes have *no effect* until you restart the server. This may leave you wondering why your changes don't seem to make any difference!

5.5.8. Causes of `Access denied` Errors

If you encounter problems when you try to connect to the MySQL server, the following items describe some courses of action you can take to correct the problem.

- Make sure that the server is running. If it is not running, you cannot connect to it. For example, if you attempt to connect to the server and see a message such as one of those following, one cause might be that the server is not running:

  ```
  shell> mysql
  ERROR 2003: Can't connect to MySQL server on 'host_name' (111)
  shell> mysql
  ERROR 2002: Can't connect to local MySQL server through socket
  '/tmp/mysql.sock' (111)
  ```

  It might also be that the server is running, but you are trying to connect using a TCP/IP port, named pipe, or Unix socket file different from those on which the server is listening. To correct this when you invoke a client program, specify a `--port` option to indicate the proper port, or a `--socket` option to indicate the proper named pipe or Unix socket file. To find out where the socket file is, you can do:

  ```
  shell> netstat -ln | grep mysql
  ```

- The grant tables must be properly set up so that the server can use them for access control. For some distribution types (such as binary distributions on Windows on RPM distributions on Linux), the installation process initializes the `mysql` database containing the grant tables. For distributions that do not do this, you should initialize the grant tables manually by running the **mysql_install_db** script. For details, see Section 2.9.2, “Unix Post-Installation Procedures”.

  One way to determine whether you need to initialize the grant tables is to look for a `mysql` directory under the data directory. (The data directory normally is named `data` or `var` and is located under your MySQL installation directory.) Make sure that you have a file named `user.MYD` in the `mysql` database directory. If you do not, execute the **mysql_install_db** script. After running this script and starting the server, test the initial privileges by executing this command:

  ```
  shell> mysql -u root test
  ```

  The server should let you connect without error.

- After a fresh installation, you should connect to the server and set up your users and their access permissions:

  ```
  shell> mysql -u root mysql
  ```

  The server should let you connect because the MySQL `root` user has no password initially. That is also a security risk, so setting the password for the `root` accounts is something you should do while you're setting up your other MySQL users. For instructions on setting the initial passwords, see Section 2.9.3, “Securing the Initial MySQL Accounts”.

- If you have updated an existing MySQL installation to a newer version, did you run the **mysql_fix_privilege_tables** script? If not, do so. The structure of the grant tables changes occasionally when new capabilities are added, so after an upgrade you should always make sure that your tables have the current structure. For instructions, see Section 2.10.7, “Upgrading the Grant Tables”.

- If a client program receives the following error message when it tries to connect, it means that the server expects passwords in a newer format than the client is capable of generating:

  ```
  shell> mysql
  Client does not support authentication protocol requested
  by server; consider upgrading MySQL client
  ```

  For information on how to deal with this, see Section 5.5.9, “Password Hashing in MySQL 4.1” and Section A.2.3, “`Client does not support authentication protocol`”.

- If you try to connect as `root` and get the following error, it means that you don't have an row in the `user` table with a `User` column value of `'root'` and that **mysqld** cannot resolve the hostname for your client:

  ```
  Access denied for user ''@'unknown' to database mysql
  ```

  In this case, you must restart the server with the `--skip-grant-tables` option and edit your `/etc/hosts` or `\windows\hosts` file to add an entry for your host.

- Remember that client programs use connection parameters specified in option files or environment variables. If a client program seems to be sending incorrect default connection parameters when you don't specify them on the command line, check your environment and any applicable option files. For example, if you get `Access denied` when you run a client without any options, make sure that you haven't specified an old password in any of your option files!

  You can suppress the use of option files by a client program by invoking it with the `--no-defaults` option. For example:

  ```
  shell> mysqladmin --no-defaults -u root version
  ```

  The option files that clients use are listed in Section 4.3.2, “Using Option Files”. Environment variables are listed in Appendix F, *Environment Variables*.

- If you get the following error, it means that you are using an incorrect `root` password:

  ```
  shell> mysqladmin -u root -pxxxx ver
  Access denied for user 'root'@'localhost' (using password: YES)
  ```

  If the preceding error occurs even when you haven't specified a password, it means that you have an incorrect password listed in some option file. Try the `--no-defaults` option as described in the previous item.

  For information on changing passwords, see Section 5.6.5, “Assigning Account Passwords”.

  If you have lost or forgotten the `root` password, you can restart **mysqld** with `--skip-grant-tables` to change the password. See Section A.4.1, “How to Reset the Root Password”.

- If you change a password by using `SET PASSWORD`, `INSERT`, or `UPDATE`, you must encrypt the password using the `PASSWORD()` function. If you do not use `PASSWORD()` for these statements, the password does not work. For example, the following statement sets a password, but fails to en-

crypt it, so the user is not able to connect afterward:

```
mysql> SET PASSWORD FOR 'abe'@'host_name' = 'eagle';
```

Instead, set the password like this:

```
mysql> SET PASSWORD FOR 'abe'@'host_name' = PASSWORD('eagle');
```

The `PASSWORD()` function is unnecessary when you specify a password using the `GRANT` statement or the **mysqladmin password** command, both of which automatically use `PASSWORD()` to encrypt the password. See Section 5.6.5, “Assigning Account Passwords”.

- `localhost` is a synonym for your local hostname, and is also the default host to which clients try to connect if you specify no host explicitly. However, connections to `localhost` on Unix systems do not work if you are using a MySQL version older than 3.23.27 that uses MIT-pthreads: `localhost` connections are made using Unix socket files, which were not supported by MIT-pthreads at that time.

  To avoid this problem on such systems, you can use a `--host=127.0.0.1` option to name the server host explicitly. This will make a TCP/IP connection to the local **mysqld** server. You can also use TCP/IP by specifying a `--host` option that uses the actual hostname of the local host. In this case, the hostname must be specified in a `user` table row on the server host, even though you are running the client program on the same host as the server.

- If you get an `Access denied` error when trying to connect to the database with `mysql -u user_name`, you may have a problem with the `user` table. Check this by executing `mysql -u root mysql` and issuing this SQL statement:

  ```
  mysql> SELECT * FROM user;
  ```

  The result should include an row with the `Host` and `User` columns matching your computer's hostname and your MySQL username.

- The `Access denied` error message tells you who you are trying to log in as, the client host from which you are trying to connect, and whether or not you were using a password. Normally, you should have one row in the `user` table that exactly matches the hostname and username that were given in the error message. For example, if you get an error message that contains `using password: NO`, it means that you tried to log in without an password.

- If the following error occurs when you try to connect from a host other than the one on which the MySQL server is running, it means that there is no row in the `user` table with a `Host` value that matches the client host:

  ```
  Host ... is not allowed to connect to this MySQL server
  ```

  You can fix this by setting up an account for the combination of client hostname and username that you are using when trying to connect.

  If you don't know the IP number or hostname of the machine from which you are connecting, you should put an row with `'%'` as the `Host` column value in the `user` table and restart **mysqld** with the `--log` option on the server machine. After trying to connect from the client machine, the information in the MySQL log indicates how you really did connect. (Then change the `'%'` in the `user` table row to the actual hostname that shows up in the log. Otherwise, you'll have a system that is insecure because it allows connections from any host for the given username.)

  On Linux, another reason that this error might occur is that you are using a binary MySQL version

that is compiled with a different version of the `glibc` library than the one you are using. In this case, you should either upgrade your operating system or `glibc`, or download a source distribution of MySQL version and compile it yourself. A source RPM is normally trivial to compile and install, so this isn't a big problem.

- If you specify a hostname when trying to connect, but get an error message where the hostname is not shown or is an IP number, it means that the MySQL server got an error when trying to resolve the IP number of the client host to a name:

  ```
  shell> mysqladmin -u root -pxxxx -h some-hostname ver
  Access denied for user 'root'@'' (using password: YES)
  ```

  This indicates a DNS problem. To fix it, execute **mysqladmin flush-hosts** to reset the internal DNS hostname cache. See Section 7.5.6, “How MySQL Uses DNS”.

  Some permanent solutions are:

  - Try to find out what is wrong with your DNS server and fix it.
  - Specify IP numbers rather than hostnames in the MySQL grant tables.
  - Put an entry for the client machine name in `/etc/hosts`.
  - Start **mysqld** with the `--skip-name-resolve` option.
  - Start **mysqld** with the `--skip-host-cache` option.
  - On Unix, if you are running the server and the client on the same machine, connect to `localhost`. Unix connections to `localhost` use a Unix socket file rather than TCP/IP.
  - On Windows, if you are running the server and the client on the same machine and the server supports named pipe connections, connect to the hostname `.` (period). Connections to `.` use a named pipe rather than TCP/IP.

- If `mysql -u root test` works but `mysql -h your_hostname -u root test` results in `Access denied` (where *`your_hostname`* is the actual hostname of the local host), you may not have the correct name for your host in the `user` table. A common problem here is that the `Host` value in the `user` table row specifies an unqualified hostname, but your system's name resolution routines return a fully qualified domain name (or vice versa). For example, if you have an entry with host `'tcx'` in the `user` table, but your DNS tells MySQL that your hostname is `'tcx.subnet.se'`, the entry does not work. Try adding an entry to the `user` table that contains the IP number of your host as the `Host` column value. (Alternatively, you could add an entry to the `user` table with a `Host` value that contains a wildcard; for example, `'tcx.%'`. However, use of hostnames ending with '%' is *insecure* and is *not* recommended!)

- If `mysql -u user_name test` works but `mysql -u user_name other_db_name` does not, you have not granted database access for *`other_db_name`* to the given user.

- If `mysql -u user_name` works when executed on the server host, but `mysql -h host_name -u user_name` doesn't work when executed on a remote client host, you have not enabled access to the server for the given username from the remote host.

- If you can't figure out why you get `Access denied`, remove from the `user` table all entries that have `Host` values containing wildcards (entries that contain '%' or '_'). A very common error is to insert a new entry with `Host='%'` and `User='some_user'`, thinking that this allows you to specify `localhost` to connect from the same machine. The reason that this doesn't work is that the default privileges include an entry with `Host='localhost'` and `User=''`. Because that entry

has a `Host` value `'localhost'` that is more specific than `'%'`, it is used in preference to the new entry when connecting from `localhost`! The correct procedure is to insert a second entry with `Host='localhost'` and `User='some_user'`, or to delete the entry with `Host='localhost'` and `User=''`. After deleting the entry, remember to issue a `FLUSH PRIVILEGES` statement to reload the grant tables.

- If you get the following error, you may have a problem with the `db` or `host` table:

  ```
  Access to database denied
  ```

  If the entry selected from the `db` table has an empty value in the `Host` column, make sure that there are one or more corresponding entries in the `host` table specifying which hosts the `db` table entry applies to.

- If you are able to connect to the MySQL server, but get an `Access denied` message whenever you issue a `SELECT ... INTO OUTFILE` or `LOAD DATA INFILE` statement, your entry in the `user` table doesn't have the `FILE` privilege enabled.

- If you change the grant tables directly (for example, by using `INSERT`, `UPDATE`, or `DELETE` statements) and your changes seem to be ignored, remember that you must execute a `FLUSH PRIVILEGES` statement or a **mysqladmin flush-privileges** command to cause the server to re-read the privilege tables. Otherwise, your changes have no effect until the next time the server is restarted. Remember that after you change the `root` password with an `UPDATE` command, you won't need to specify the new password until after you flush the privileges, because the server won't know you've changed the password yet!

- If your privileges seem to have changed in the middle of a session, it may be that a MySQL administrator has changed them. Reloading the grant tables affects new client connections, but it also affects existing connections as indicated in Section 5.5.7, “When Privilege Changes Take Effect”.

- If you have access problems with a Perl, PHP, Python, or ODBC program, try to connect to the server with `mysql -u user_name db_name` or `mysql -u user_name -pyour_pass db_name`. If you are able to connect using the **mysql** client, the problem lies with your program, not with the access privileges. (There is no space between `-p` and the password; you can also use the `--password=your_pass` syntax to specify the password. If you use the `-p` option alone, MySQL prompts you for the password.)

- For testing, start the **mysqld** server with the `--skip-grant-tables` option. Then you can change the MySQL grant tables and use the **mysqlaccess** script to check whether your modifications have the desired effect. When you are satisfied with your changes, execute **mysqladmin flush-privileges** to tell the **mysqld** server to start using the new grant tables. (Reloading the grant tables overrides the `--skip-grant-tables` option. This allows you to tell the server to begin using the grant tables again without stopping and restarting it.)

- If everything else fails, start the **mysqld** server with a debugging option (for example, `--debug=d,general,query`). This prints host and user information about attempted connections, as well as information about each command issued. See Section E.1.2, “Creating Trace Files”.

- If you have any other problems with the MySQL grant tables and feel you must post the problem to the mailing list, always provide a dump of the MySQL grant tables. You can dump the tables with the **mysqldump mysql** command. As always, post your problem using the **mysqlbug** script. See Section 1.4.1.3, “How to Report Bugs or Problems”. In some cases, you may need to restart **mysqld** with `--skip-grant-tables` to run **mysqldump**.

5.5.9. Password Hashing in MySQL 4.1

MySQL user accounts are listed in the `user` table of the `mysql` database. Each MySQL account is assigned a password, although what is stored in the `Password` column of the `user` table is not the plaintext version of the password, but a hash value computed from it. Password hash values are computed by the `PASSWORD()` function.

MySQL uses passwords in two phases of client/server communication:

- When a client attempts to connect to the server, there is an initial authentication step in which the client must present a password that has a hash value matching the hash value stored in the `user` table for the account that the client wants to use.
- After the client connects, it can (if it has sufficient privileges) set or change the password hashes for accounts listed in the `user` table. The client can do this by using the `PASSWORD()` function to generate a password hash, or by using the `GRANT` or `SET PASSWORD` statements.

In other words, the server *uses* hash values during authentication when a client first attempts to connect. The server *generates* hash values if a connected client invokes the `PASSWORD()` function or uses a `GRANT` or `SET PASSWORD` statement to set or change a password.

The password hashing mechanism was updated in MySQL 4.1 to provide better security and to reduce the risk of passwords being intercepted. However, this new mechanism is understood only by the 4.1 server and 4.1 clients, which can result in some compatibility problems. A 4.1 client can connect to a pre-4.1 server, because the client understands both the old and new password hashing mechanisms. However, a pre-4.1 client that attempts to connect to a 4.1 server may run into difficulties. For example, a 4.0 **mysql** client that attempts to connect to a 4.1 server may fail with the following error message:

```
shell> mysql -h localhost -u root
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

The following discussion describes the differences between the old and new password mechanisms, and what you should do if you upgrade your server to 4.1 but need to maintain backward compatibility with pre-4.1 clients. Additional information can be found in Section A.2.3, “`Client does not support authentication protocol`”.

Note: This discussion contrasts 4.1 behavior with pre-4.1 behavior, but the 4.1 behavior described here actually begins with 4.1.1. MySQL 4.1.0 is an ``odd'' release because it has a slightly different mechanism than that implemented in 4.1.1 and up. Differences between 4.1.0 and more recent versions are described further in Section 5.5.9.2, “Password Hashing in MySQL 4.1.0”.

Prior to MySQL 4.1, password hashes computed by the `PASSWORD()` function are 16 bytes long. Such hashes look like this:

```
mysql> SELECT PASSWORD('mypass');
+--------------------+
| PASSWORD('mypass') |
+--------------------+
| 6f8c114b58f2ce9e   |
+--------------------+
```

The `Password` column of the `user` table (in which these hashes are stored) also is 16 bytes long before MySQL 4.1.

As of MySQL 4.1, the `PASSWORD()` function has been modified to produce a longer 41-byte hash value:

```
mysql> SELECT PASSWORD('mypass');
+-------------------------------------------+
| PASSWORD('mypass')                        |
+-------------------------------------------+
| *43c8aa34cdc98eddd3de1fe9a9c2c2a9f92bb2098d75 |
+-------------------------------------------+
```

Accordingly, the `Password` column in the `user` table also must be 41 bytes long to store these values:

- If you perform a new installation of MySQL 4.1, the `Password` column is made 41 bytes long automatically.
- If you upgrade an older installation to 4.1, you should run the **mysql_fix_privilege_tables** script to increase the length of the `Password` column from 16 to 41 bytes. (The script does not change existing password values, which remain 16 bytes long.)

A widened `Password` column can store password hashes in both the old and new formats. The format of any given password hash value can be determined two ways:

- The obvious difference is the length (16 bytes versus 41 bytes).
- A second difference is that password hashes in the new format always begin with a `'*'` character, whereas passwords in the old format never do.

The longer password hash format has better cryptographic properties, and client authentication based on long hashes is more secure than that based on the older short hashes.

The differences between short and long password hashes are relevant both for how the server uses passwords during authentication and for how it generates password hashes for connected clients that perform password-changing operations.

The way in which the server uses password hashes during authentication is affected by the width of the `Password` column:

- If the column is short, only short-hash authentication is used.
- If the column is long, it can hold either short or long hashes, and the server can use either format:
  - Pre-4.1 clients can connect, although because they know only about the old hashing mechanism, they can authenticate only for accounts that have short hashes.
  - 4.1 clients can authenticate for accounts that have short or long hashes.

For short-hash accounts, the authentication process is actually a bit more secure for 4.1 clients than for older clients. In terms of security, the gradient from least to most secure is:

- Pre-4.1 client authenticating for account with short password hash
- 4.1 client authenticating for account with short password hash
- 4.1 client authenticating for account with long password hash

The way in which the server generates password hashes for connected clients is affected by the width of the `Password` column and by the `--old-passwords` option. A 4.1 server generates long hashes only if certain conditions are met: The `Password` column must be wide enough to hold long values and the `--old-passwords` option must not be given. These conditions apply as follows:

- The `Password` column must be wide enough to hold long hashes (41 bytes). If the column has not been updated and still has the pre-4.1 width of 16 bytes, the server notices that long hashes cannot fit into it and generates only short hashes when a client performs password-changing operations using `PASSWORD()`, `GRANT`, or `SET PASSWORD`. This is the behavior that occurs if you have upgraded to 4.1 but have not yet run the **mysql_fix_privilege_tables** script to widen the `Password` column.
- If the `Password` column is wide, it can store either short or long password hashes. In this case, `PASSWORD()`, `GRANT`, and `SET PASSWORD` generate long hashes unless the server was started with the `--old-passwords` option. That option forces the server to generate short password hashes instead.

The purpose of the `--old-passwords` option is to allow you to maintain backward compatibility with pre-4.1 clients under circumstances where the server would otherwise generate long password hashes. The option doesn't affect authentication (4.1 clients can still use accounts that have long password hashes), but it does prevent creation of a long password hash in the `user` table as the result of a password-changing operation. Were that to occur, the account no longer could be used by pre-4.1 clients. Without the `--old-passwords` option, the following undesirable scenario is possible:

- An old client connects to an account that has a short password hash.
- The client changes its own password. Without `--old-passwords`, this results in the account having a long password hash.
- The next time the old client attempts to connect to the account, it cannot, because the account has a long password hash that requires the new hashing mechanism during authentication. (Once an account has a long password hash in the user table, only 4.1 clients can authenticate for it, because pre-4.1 clients do not understand long hashes.)

This scenario illustrates that, if you must support older pre-4.1 clients, it is dangerous to run a 4.1 server without using the `--old-passwords` option. By running the server with `--old-passwords`, password-changing operations do not generate long password hashes and thus do not cause accounts to become inaccessible to older clients. (Those clients cannot inadvertently lock themselves out by changing their password and ending up with a long password hash.)

The downside of the `--old-passwords` option is that any passwords you create or change use short hashes, even for 4.1 clients. Thus, you lose the additional security provided by long password hashes. If you want to create an account that has a long hash (for example, for use by 4.1 clients), you must do so while running the server without `--old-passwords`.

The following scenarios are possible for running a 4.1 server:

Scenario 1: Short `Password` column in user table:

- Only short hashes can be stored in the `Password` column.
- The server uses only short hashes during client authentication.
- For connected clients, password hash-generating operations involving `PASSWORD()`, `GRANT`, or `SET PASSWORD` use short hashes exclusively. Any change to an account's password results in that

account having a short password hash.

- The `--old-passwords` option can be used but is superfluous because with a short `Password` column, the server generates only short password hashes anyway.

Scenario 2: Long `Password` column; server not started with `--old-passwords` option:

- Short or long hashes can be stored in the `Password` column.
- 4.1 clients can authenticate for accounts that have short or long hashes.
- Pre-4.1 clients can authenticate only for accounts that have short hashes.
- For connected clients, password hash-generating operations involving `PASSWORD()`, `GRANT`, or `SET PASSWORD` use long hashes exclusively. A change to an account's password results in that account having a long password hash.

As indicated earlier, a danger in this scenario is that it is possible for accounts that have a short password hash to become inaccessible to pre-4.1 clients. A change to such an account's password made via `GRANT`, `PASSWORD()`, or `SET PASSWORD` results in the account being given a long password hash. From that point on, no pre-4.1 client can authenticate to that account until the client upgrades to 4.1.

To deal with this problem, you can change a password in a special way. For example, normally you use `SET PASSWORD` as follows to change an account password:

```
mysql> SET PASSWORD FOR 'some_user'@'some_host' = PASSWORD('mypass');
```

To change the password but create a short hash, use the `OLD_PASSWORD()` function instead:

```
mysql> SET PASSWORD FOR 'some_user'@'some_host' = OLD_PASSWORD('mypass');
```

`OLD_PASSWORD()` is useful for situations in which you explicitly want to generate a short hash.

Scenario 3: Long `Password` column; server started with `--old-passwords` option:

- Short or long hashes can be stored in the `Password` column.
- 4.1 clients can authenticate for accounts that have short or long hashes (but note that it is possible to create long hashes only when the server is started without `--old-passwords`).
- Pre-4.1 clients can authenticate only for accounts that have short hashes.
- For connected clients, password hash-generating operations involving `PASSWORD()`, `GRANT`, or `SET PASSWORD` use short hashes exclusively. Any change to an account's password results in that account having a short password hash.

In this scenario, you cannot create accounts that have long password hashes, because the `--old-passwords` option prevents generation of long hashes. Also, if you create an account with a long hash before using the `--old-passwords` option, changing the account's password while `--old-passwords` is in effect results in the account being given a short password, causing it to lose the security benefits of a longer hash.

The disadvantages for these scenarios may be summarized as follows:

In scenario 1, you cannot take advantage of longer hashes that provide more secure authentication.

In scenario 2, accounts with short hashes become inaccessible to pre-4.1 clients if you change their passwords without explicitly using `OLD_PASSWORD()`.

In scenario 3, `--old-passwords` prevents accounts with short hashes from becoming inaccessible, but password-changing operations cause accounts with long hashes to revert to short hashes, and you cannot change them back to long hashes while `--old-passwords` is in effect.

5.5.9.1. Implications of Password Hashing Changes for Application Programs

An upgrade to MySQL 4.1 can cause a compatibility issue for applications that use `PASSWORD()` to generate passwords for their own purposes. Applications really should not do this, because `PASSWORD()` should be used only to manage passwords for MySQL accounts. But some applications use `PASSWORD()` for their own purposes anyway.

If you upgrade to 4.1 and run the server under conditions where it generates long password hashes, an application that uses `PASSWORD()` for its own passwords breaks. The recommended course of action is to modify the application to use another function, such as `SHA1()` or `MD5()`, to produce hashed values. If that is not possible, you can use the `OLD_PASSWORD()` function, which is provided to generate short hashes in the old format. But note that `OLD_PASSWORD()` may one day no longer be supported.

If the server is running under circumstances where it generates short hashes, `OLD_PASSWORD()` is available but is equivalent to `PASSWORD()`.

5.5.9.2. Password Hashing in MySQL 4.1.0

Password hashing in MySQL 4.1.0 differs from hashing in 4.1.1 and up. The 4.1.0 differences are:

- Password hashes are 45 bytes long rather than 41 bytes.
- The `PASSWORD()` function is non-repeatable. That is, with a given argument *X*, successive calls to `PASSWORD(X)` generate different results.

These differences make authentication in 4.1.0 incompatible with that of releases that follow it. If you have upgraded to MySQL 4.1.0, it is recommended that you upgrade to a newer version as soon as possible. After you do, reassign any long passwords in the `user` table so that they are compatible with the 41-byte format.

5.6. MySQL User Account Management

This section describes how to set up accounts for clients of your MySQL server. It discusses the following topics:

- The meaning of account names and passwords as used in MySQL and how that compares to names and passwords used by your operating system
- How to set up new accounts and remove existing accounts
- How to change passwords
- Guidelines for using passwords securely

- How to use secure connections with SSL

5.6.1. MySQL Usernames and Passwords

A MySQL account is defined in terms of a username and the client host or hosts from which the user can connect to the server. The account also has a password. There are several distinctions between the way usernames and passwords are used by MySQL and the way they are used by your operating system:

- Usernames, as used by MySQL for authentication purposes, have nothing to do with usernames (login names) as used by Windows or Unix. On Unix, most MySQL clients by default try to log in using the current Unix username as the MySQL username, but that is for convenience only. The default can be overridden easily, because client programs allow any username to be specified with a `-u` or `--user` option. Because this means that anyone can attempt to connect to the server using any username, you can't make a database secure in any way unless all MySQL accounts have passwords. Anyone who specifies a username for an account that has no password is able to connect successfully to the server.

- MySQL usernames can be up to 16 characters long. Operating system usernames might have a different maximum length. For example, Unix usernames typically are limited to eight characters.

- MySQL passwords have nothing to do with passwords for logging in to your operating system. There is no necessary connection between the password you use to log in to a Windows or Unix machine and the password you use to access the MySQL server on that machine.

- MySQL encrypts passwords using its own algorithm. This encryption is different from that used during the Unix login process. MySQL password encryption is the same as that implemented by the `PASSWORD()` SQL function. Unix password encryption is the same as that implemented by the `ENCRYPT()` SQL function. See the descriptions of the `PASSWORD()` and `ENCRYPT()` functions in Section 12.8.2, “Encryption Functions”. From version 4.1 on, MySQL employs a stronger authentication method that has better password protection during the connection process than in earlier versions. It is secure even if TCP/IP packets are sniffed or the `mysql` database is captured. (In earlier versions, even though passwords are stored in encrypted form in the `user` table, knowledge of the encrypted password value could be used to connect to the MySQL server.)

When you install MySQL, the grant tables are populated with an initial set of accounts. These accounts have names and access privileges that are described in Section 2.9.3, “Securing the Initial MySQL Accounts”, which also discusses how to assign passwords to them. Thereafter, you normally set up, modify, and remove MySQL accounts using the `GRANT` and `REVOKE` statements. See Section 13.5.1.3, “`GRANT` and `REVOKE` Syntax”.

When you connect to a MySQL server with a command-line client, you should specify the username and password for the account that you want to use:

```
shell> mysql --user=monty --password=guess db_name
```

If you prefer short options, the command looks like this:

```
shell> mysql -u monty -pguess db_name
```

There must be *no space* between the `-p` option and the following password value. See Section 5.5.4, “Connecting to the MySQL Server”.

The preceding commands include the password value on the command line, which can be a security risk.

See Section 5.6.6, “Keeping Your Password Secure”. To avoid this, specify the `--password` or `-p` option without any following password value:

```
shell> mysql --user=monty --password db_name
shell> mysql -u monty -p db_name
```

Then the client program prints a prompt and waits for you to enter the password. (In these examples, *db_name* is *not* interpreted as a password, because it is separated from the preceding password option by a space.)

On some systems, the library call that MySQL uses to prompt for a password automatically limits the password to eight characters. That is a problem with the system library, not with MySQL. Internally, MySQL doesn't have any limit for the length of the password. To work around the problem, change your MySQL password to a value that is eight or fewer characters long, or put your password in an option file.

5.6.2. Adding New User Accounts to MySQL

You can create MySQL accounts in two ways:

- By using `GRANT` statements
- By manipulating the MySQL grant tables directly

The preferred method is to use `GRANT` statements, because they are more concise and less error-prone. `GRANT` is available as of MySQL 3.22.11; its syntax is described in Section 13.5.1.3, “`GRANT` and `REVOKE` Syntax”.

Another option for creating accounts is to use one of several available third-party programs that offer capabilities for MySQL account administration. `phpMyAdmin` is one such program.

The following examples show how to use the **mysql** client program to set up new users. These examples assume that privileges are set up according to the defaults described in Section 2.9.3, “Securing the Initial MySQL Accounts”. This means that to make changes, you must connect to the MySQL server as the MySQL `root` user, and the `root` account must have the `INSERT` privilege for the `mysql` database and the `RELOAD` administrative privilege.

First, use the **mysql** program to connect to the server as the MySQL `root` user:

```
shell> mysql --user=root mysql
```

If you have assigned a password to the `root` account, you'll also need to supply a `--password` or `-p` option for this **mysql** command and also for those later in this section.

After connecting to the server as `root`, you can add new accounts. The following statements use `GRANT` to set up four new accounts:

```
mysql> GRANT ALL PRIVILEGES ON *.* TO 'monty'@'localhost'
    ->     IDENTIFIED BY 'some_pass' WITH GRANT OPTION;
mysql> GRANT ALL PRIVILEGES ON *.* TO 'monty'@'%'
    ->     IDENTIFIED BY 'some_pass' WITH GRANT OPTION;
mysql> GRANT RELOAD,PROCESS ON *.* TO 'admin'@'localhost';
mysql> GRANT USAGE ON *.* TO 'dummy'@'localhost';
```

The accounts created by these GRANT statements have the following properties:

- Two of the accounts have a username of monty and a password of some_pass. Both accounts are superuser accounts with full privileges to do anything. One account ('monty'@'localhost') can be used only when connecting from the local host. The other ('monty'@'%') can be used to connect from any other host. Note that it is necessary to have both accounts for monty to be able to connect from anywhere as monty. Without the localhost account, the anonymous-user account for localhost that is created by **mysql_install_db** would take precedence when monty connects from the local host. As a result, monty would be treated as an anonymous user. The reason for this is that the anonymous-user account has a more specific Host column value than the 'monty'@'%' account and thus comes earlier in the user table sort order. (user table sorting is discussed in Section 5.5.5, “Access Control, Stage 1: Connection Verification”.)

- One account has a username of admin and no password. This account can be used only by connecting from the local host. It is granted the RELOAD and PROCESS administrative privileges. These privileges allow the admin user to execute the **mysqladmin reload**, **mysqladmin refresh**, and **mysqladmin flush-*xxx*** commands, as well as **mysqladmin processlist** . No privileges are granted for accessing any databases. You could add such privileges later by issuing additional GRANT statements.

- One account has a username of dummy and no password. This account can be used only by connecting from the local host. No privileges are granted. The USAGE privilege in the GRANT statement allows you to create an account without giving it any privileges. It has the effect of setting all the global privileges to 'N'. It is assumed that you will grant specific privileges to the account later.

As an alternative to GRANT, you can create the same accounts directly by issuing INSERT statements and then telling the server to reload the grant tables:

```
shell> mysql --user=root mysql
mysql> INSERT INTO user
    ->     VALUES('localhost','monty',PASSWORD('some_pass'),
    ->     'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO user
    ->     VALUES('%','monty',PASSWORD('some_pass'),
    ->     'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO user SET Host='localhost',User='admin',
    ->     Reload_priv='Y', Process_priv='Y';
mysql> INSERT INTO user (Host,User,Password)
    ->     VALUES('localhost','dummy','');
mysql> FLUSH PRIVILEGES;
```

The reason for using FLUSH PRIVILEGES when you create accounts with INSERT is to tell the server to re-read the grant tables. Otherwise, the changes go unnoticed until you restart the server. With GRANT, FLUSH PRIVILEGES is unnecessary.

The reason for using the PASSWORD() function with INSERT is to encrypt the password. The GRANT statement encrypts the password for you, so PASSWORD() is unnecessary.

The 'Y' values enable privileges for the accounts. Depending on your MySQL version, you may have to use a different number of 'Y' values in the first two INSERT statements. (Versions prior to 3.22.11 have fewer privilege columns, and versions from 4.0.2 on have more.) For the admin account, the more readable extended INSERT syntax using SET that is available starting with MySQL 3.22.11 is used.

In the INSERT statement for the dummy account, only the Host, User, and Password columns in the user table record are assigned values. None of the privilege columns are set explicitly, so MySQL assigns them all the default value of 'N'. This is equivalent to what GRANT USAGE does.

Note that to set up a superuser account, it is necessary only to create a `user` table entry with the privilege columns set to `'Y'`. `user` table privileges are global, so no entries in any of the other grant tables are needed.

The next examples create three accounts and give them access to specific databases. Each of them has a username of `custom` and password of `obscure`.

To create the accounts with `GRANT`, use the following statements:

```
shell> mysql --user=root mysql
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    ->     ON bankaccount.*
    ->     TO 'custom'@'localhost'
    ->     IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    ->     ON expenses.*
    ->     TO 'custom'@'whitehouse.gov'
    ->     IDENTIFIED BY 'obscure';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
    ->     ON customer.*
    ->     TO 'custom'@'server.domain'
    ->     IDENTIFIED BY 'obscure';
```

The three accounts can be used as follows:

- The first account can access the `bankaccount` database, but only from the local host.
- The second account can access the `expenses` database, but only from the host `whitehouse.gov`.
- The third account can access the `customer` database, but only from the host `server.domain`.

To set up the `custom` accounts without `GRANT`, use `INSERT` statements as follows to modify the grant tables directly:

```
shell> mysql --user=root mysql
mysql> INSERT INTO user (Host,User,Password)
    ->     VALUES('localhost','custom',PASSWORD('obscure'));
mysql> INSERT INTO user (Host,User,Password)
    ->     VALUES('whitehouse.gov','custom',PASSWORD('obscure'));
mysql> INSERT INTO user (Host,User,Password)
    ->     VALUES('server.domain','custom',PASSWORD('obscure'));
mysql> INSERT INTO db
    ->     (Host,Db,User,Select_priv,Insert_priv,
    ->     Update_priv,Delete_priv,Create_priv,Drop_priv)
    ->     VALUES('localhost','bankaccount','custom',
    ->     'Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO db
    ->     (Host,Db,User,Select_priv,Insert_priv,
    ->     Update_priv,Delete_priv,Create_priv,Drop_priv)
    ->     VALUES('whitehouse.gov','expenses','custom',
    ->     'Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO db
    ->     (Host,Db,User,Select_priv,Insert_priv,
    ->     Update_priv,Delete_priv,Create_priv,Drop_priv)
    ->     VALUES('server.domain','customer','custom',
    ->     'Y','Y','Y','Y','Y','Y');
mysql> FLUSH PRIVILEGES;
```

The first three INSERT statements add user table entries that allow the user custom to connect from the various hosts with the given password, but grant no global privileges (all privileges are set to the default value of 'N'). The next three INSERT statements add db table entries that grant privileges to custom for the bankaccount, expenses, and customer databases, but only when accessed from the proper hosts. As usual when you modify the grant tables directly, you tell the server to reload them with FLUSH PRIVILEGES so that the privilege changes take effect.

If you want to give a specific user access from all machines in a given domain (for example, mydomain.com), you can issue a GRANT statement that uses the '%' wildcard character in the host part of the account name:

```
mysql> GRANT ...
    ->     ON *.*
    ->     TO 'myname'@'%.mydomain.com'
    ->     IDENTIFIED BY 'mypass';
```

To do the same thing by modifying the grant tables directly, do this:

```
mysql> INSERT INTO user (Host,User,Password,...)
    ->     VALUES('%.mydomain.com','myname',PASSWORD('mypass'),...);
mysql> FLUSH PRIVILEGES;
```

5.6.3. Removing User Accounts from MySQL

To remove an account, use the DROP USER statement, which was added in MySQL 4.1.1. For older versions of MySQL, use DELETE instead. The account removal procedure is described in Section 13.5.1.2, “DROP USER Syntax”.

5.6.4. Limiting Account Resources

Before MySQL 4.0.2, the only available method for limiting use of MySQL server resources is to set the max_user_connections system variable to a non-zero value. But that method is strictly global. It does not allow for management of individual accounts. Also, it limits only the number of simultaneous connections made using a single account, not what a client can do once connected. Both types of control are interest to many MySQL administrators, particularly those for Internet Service Providers.

Starting from MySQL 4.0.2, you can limit the following server resources for individual accounts:

- The number of queries that an account can issue per hour
- The number of updates that an account can issue per hour
- The number of times an account can connect to the server per hour

Any statement that a client can issue counts against the query limit. Only statements that modify databases or tables count against the update limit.

From MySQL 5.0.3 on, it is also possible to limit the number of simultaneous connection to the server on a per-account basis.

An account in this context is a single record in the user table. Each account is uniquely identified by its User and Host column values.

As a prerequisite for using this feature, the user table in the mysql database must contain the re-

source-related columns. Resource limits are stored in the `max_questions`, `max_updates`, `max_connections`, and `max_user_connections` columns. If your `user` table doesn't have these columns, it must be upgraded; see Section 2.10.7, “Upgrading the Grant Tables”.

To set resource limits with a `GRANT` statement, use a `WITH` clause that names each resource to be limited and a per-hour count indicating the limit value. For example, to create a new account that can access the `customer` database, but only in a limited fashion, issue this statement:

```
mysql> GRANT ALL ON customer.* TO 'francis'@'localhost'
    ->     IDENTIFIED BY 'frank'
    ->     WITH MAX_QUERIES_PER_HOUR 20
    ->          MAX_UPDATES_PER_HOUR 10
    ->          MAX_CONNECTIONS_PER_HOUR 5
    ->          MAX_USER_CONNECTIONS 2;
```

The limit types need not all be named in the `WITH` clause, but those named can be present in any order. The value for each per-hour limit should be an integer representing a count per hour. If the `GRANT` statement has no `WITH` clause, the limits are each set to the default value of zero (that is, no limit). For `MAX_USER_CONNECTIONS`, the limit is an integer indicating the maximum number of simultaneous connections the account can make at any one time. If the limit is set to the default value of zero, the `max_user_connections` system variable determines the number of simultaneous connections for the account.

To set or change limits for an existing account, use a `GRANT USAGE` statement at the global level (`ON *.*`). The following statement changes the query limit for `francis` to 100:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
    ->     WITH MAX_QUERIES_PER_HOUR 100;
```

This statement leaves the account's existing privileges unchanged and modifies only the limit values specified.

To remove an existing limit, set its value to zero. For example, to remove the limit on how many times per hour `francis` can connect, use this statement:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
    ->     WITH MAX_CONNECTIONS_PER_HOUR 0;
```

Resource-use counting takes place when any account has a non-zero limit placed on its use of any of the resources.

As the server runs, it counts the number of times each account uses resources. If an account reaches its limit on number of connections within the last hour, further connections for the account are rejected until that hour is up. Similarly, if the account reaches its limit on the number of queries or updates, further queries or updates are rejected until the hour is up. In all such cases, an appropriate error message is issued.

Resource counting is done per account, not per client. For example, if your account has a query limit of 50, you cannot increase your limit to 100 by making two simultaneous client connections to the server. Queries issued on both connections are counted together.

The current per-hour resource-use counts can be reset globally for all accounts, or individually for a given account:

- To reset the current counts to zero for all accounts, issue a `FLUSH USER_RESOURCES` statement. The counts also can be reset by reloading the grant tables (for example, with a `FLUSH PRIVILEGES` statement or a **mysqladmin reload** command).

- The counts for an individual account can be set to zero by re-granting it any of its limits. To do this, use `GRANT USAGE` as described earlier and specify a limit value equal to the value that the account currently has.

Counter resets do not affect the `MAX_USER_CONNECTIONS` limit.

All counts begin at zero when the server starts; counts are not carried over through a restart.

5.6.5. Assigning Account Passwords

Passwords may be assigned from the command line by using the **mysqladmin** command:

```
shell> mysqladmin -u user_name -h host_name password "newpwd"
```

The account for which this command resets the password is the one with a `user` table record that matches *`user_name`* in the `User` column and the client host *from which you connect* in the `Host` column.

Another way to assign a password to an account is to issue a `SET PASSWORD` statement:

```
mysql> SET PASSWORD FOR 'jeffrey'@'%' = PASSWORD('biscuit');
```

Only users such as `root` with update access to the `mysql` database can change the password for other users. If you are not connected as an anonymous user, you can change your own password by omitting the `FOR` clause:

```
mysql> SET PASSWORD = PASSWORD('biscuit');
```

You can also use a `GRANT USAGE` statement at the global level (`ON *.*`) to assign a password to an account without affecting the account's current privileges:

```
mysql> GRANT USAGE ON *.* TO 'jeffrey'@'%' IDENTIFIED BY 'biscuit';
```

Although it is generally preferable to assign passwords using one of the preceding methods, you can also do so by modifying the `user` table directly:

- To establish a password when creating a new account, provide a value for the `Password` column:

  ```
  shell> mysql -u root mysql
  mysql> INSERT INTO user (Host,User,Password)
      -> VALUES('%','jeffrey',PASSWORD('biscuit'));
  mysql> FLUSH PRIVILEGES;
  ```

- To change the password for an existing account, use `UPDATE` to set the `Password` column value:

  ```
  shell> mysql -u root mysql
  mysql> UPDATE user SET Password = PASSWORD('bagel')
      -> WHERE Host = '%' AND User = 'francis';
  mysql> FLUSH PRIVILEGES;
  ```

When you assign an account a password using `SET PASSWORD`, `INSERT`, or `UPDATE`, you must use

the PASSWORD() function to encrypt it. (The only exception is that you need not use PASSWORD() if the password is empty.) PASSWORD() is necessary because the user table stores passwords in encrypted form, not as plaintext. If you forget that fact, you are likely to set passwords like this:

```
shell> mysql -u root mysql
mysql> INSERT INTO user (Host,User,Password)
    -> VALUES('%','jeffrey','biscuit');
mysql> FLUSH PRIVILEGES;
```

The result is that the literal value 'biscuit' is stored as the password in the user table, not the encrypted value. When jeffrey attempts to connect to the server using this password, the value is encrypted and compared to the value stored in the user table. However, the stored value is the literal string 'biscuit', so the comparison fails and the server rejects the connection:

```
shell> mysql -u jeffrey -pbiscuit test
Access denied
```

If you set passwords using the GRANT ... IDENTIFIED BY statement or the **mysqladmin password** command, they both take care of encrypting the password for you. The PASSWORD() function is unnecessary.

Note: PASSWORD() encryption is different from Unix password encryption. See Section 5.6.1, “MySQL Usernames and Passwords”.

5.6.6. Keeping Your Password Secure

On an administrative level, you should never grant access to the mysql.user table to any non-administrative accounts. Passwords in the user table are stored in encrypted form, but in versions of MySQL earlier than 4.1, knowing the encrypted password for an account makes it possible to connect to the server using that account.

When you run a client program to connect to the MySQL server, it is inadvisable to specify your password in a way that exposes it to discovery by other users. The methods you can use to specify your password when you run client programs are listed here, along with an assessment of the risks of each method:

- Use a -p*your_pass* or --password=*your_pass* option on the command line. For example:

  ```
  shell> mysql -u francis -pfrank db_name
  ```

  This is convenient but insecure, because your password becomes visible to system status programs such as **ps** that may be invoked by other users to display command lines. MySQL clients typically overwrite the command-line password argument with zeros during their initialization sequence, but there is still a brief interval during which the value is visible.

- Use a -p or --password option with no password value specified. In this case, the client program solicits the password from the terminal:

  ```
  shell> mysql -u francis -p db_name
  Enter password: ********
  ```

  The '*' characters indicate where you enter your password. The password is not displayed as you enter it.

  It is more secure to enter your password this way than to specify it on the command line because it is

not visible to other users. However, this method of entering a password is suitable only for programs that you run interactively. If you want to invoke a client from a script that runs non-interactively, there is no opportunity to enter the password from the terminal. On some systems, you may even find that the first line of your script is read and interpreted (incorrectly) as your password!

- Store your password in an option file. For example, on Unix you can list your password in the `[client]` section of the `.my.cnf` file in your home directory:

  ```
  [client]
  password=your_pass
  ```

  If you store your password in `.my.cnf`, the file should not be accessible to anyone but yourself. To ensure this, set the file access mode to `400` or `600`. For example:

  ```
  shell> chmod 600 .my.cnf
  ```

  Section 4.3.2, “Using Option Files” discusses option files in more detail.

- Store your password in the `MYSQL_PWD` environment variable. This method of specifying your MySQL password must be considered extremely insecure and should not be used. Some versions of **ps** include an option to display the environment of running processes. If you set `MYSQL_PWD`, your password is exposed to any other user who runs **ps**. Even on systems without such a version of **ps**, it is unwise to assume that there are no other methods by which users can examine process environments. See Appendix F, *Environment Variables*.

All in all, the safest methods are to have the client program prompt for the password or to specify the password in a properly protected option file.

5.6.7. Using Secure Connections

Beginning with version 4.0.0, MySQL has support for secure (encrypted) connections between MySQL clients and the server using the Secure Sockets Layer (SSL) protocol. This section discusses how to use SSL connections. It also describes a way to set up SSH on Windows.

The standard configuration of MySQL is intended to be as fast as possible, so encrypted connections are not used by default. Doing so would make the client/server protocol much slower. Encrypting data is a CPU-intensive operation that requires the computer to do additional work and can delay other MySQL tasks. For applications that require the security provided by encrypted connections, the extra computation is warranted.

MySQL allows encryption to be enabled on a per-connection basis. You can choose a normal unencrypted connection or a secure encrypted SSL connection according the requirements of individual applications.

5.6.7.1. Basic SSL Concepts

To understand how MySQL uses SSL, it's necessary to explain some basic SSL and X509 concepts. People who are familiar with them can skip this part.

By default, MySQL uses unencrypted connections between the client and the server. This means that someone with access to the network could watch all your traffic and look at the data being sent or received. They could even change the data while it is in transit between client and server. To improve security a little, you can compress client/server traffic by using the `--compress` option when invoking client programs. However, this does not foil a determined attacker.

When you need to move information over a network in a secure fashion, an unencrypted connection is unacceptable. Encryption is the way to make any kind of data unreadable. In fact, today's practice requires many additional security elements from encryption algorithms. They should resist many kind of known attacks such as changing the order of encrypted messages or replaying data twice.

SSL is a protocol that uses different encryption algorithms to ensure that data received over a public network can be trusted. It has mechanisms to detect any data change, loss, or replay. SSL also incorporates algorithms that provide identity verification using the X509 standard.

X509 makes it possible to identify someone on the Internet. It is most commonly used in e-commerce applications. In basic terms, there should be some company called a ``Certificate Authority'' (or CA) that assigns electronic certificates to anyone who needs them. Certificates rely on asymmetric encryption algorithms that have two encryption keys (a public key and a secret key). A certificate owner can show the certificate to another party as proof of identity. A certificate consists of its owner's public key. Any data encrypted with this public key can be decrypted only using the corresponding secret key, which is held by the owner of the certificate.

If you need more information about SSL, X509, or encryption, use your favorite Internet search engine to search for keywords in which you are interested.

5.6.7.2. Requirements

To use SSL connections between the MySQL server and client programs, your system must be able to support OpenSSL and your version of MySQL must be 4.0.0 or newer.

To get secure connections to work with MySQL, you must do the following:

1. Install the OpenSSL library. We have tested MySQL with OpenSSL 0.9.6. If you need OpenSSL, visit http://www.openssl.org.

2. When you configure MySQL, run the **configure** script with the `--with-vio` and `--with-openssl` options.

3. Make sure that you have upgraded your grant tables to include the SSL-related columns in the `mysql.user` table. This is necessary if your grant tables date from a version prior to MySQL 4.0.0. The upgrade procedure is described in Section 2.10.7, “Upgrading the Grant Tables”.

4. To check whether a running **mysqld** server supports OpenSSL, examine the value of the `have_openssl` system variable:

   ```
   mysql> SHOW VARIABLES LIKE 'have_openssl';
   +---------------+-------+
   | Variable_name | Value |
   +---------------+-------+
   | have_openssl  | YES   |
   +---------------+-------+
   ```

   If the value is `YES`, the server supports OpenSSL connections.

5.6.7.3. Setting Up SSL Certificates for MySQL

Here is an example for setting up SSL certificates for MySQL:

```
DIR=`pwd`/openssl
PRIV=$DIR/private
```

```
mkdir $DIR $PRIV $DIR/newcerts
cp /usr/share/ssl/openssl.cnf $DIR
replace ./demoCA $DIR -- $DIR/openssl.cnf

# Create necessary files: $database, $serial and $new_certs_dir
# directory (optional)

touch $DIR/index.txt
echo "01" > $DIR/serial

#
# Generation of Certificate Authority(CA)
#

openssl req -new -x509 -keyout $PRIV/cakey.pem -out $DIR/cacert.pem \
    -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ................++++++
# .........++++++
# writing new private key to '/home/monty/openssl/private/cakey.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL admin
# Email Address []:

#
# Create server request and key
#
openssl req -new -keyout $DIR/server-key.pem -out \
    $DIR/server-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ..++++++
# ..........++++++
# writing new private key to '/home/monty/openssl/server-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
```

```
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL server
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove the passphrase from the key (optional)
#

openssl rsa -in $DIR/server-key.pem -out $DIR/server-key.pem

#
# Sign server cert
#
openssl ca  -policy policy_anything -out $DIR/server-cert.pem \
    -config $DIR/openssl.cnf -infiles $DIR/server-req.pem

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName           :PRINTABLE:'FI'
# organizationName      :PRINTABLE:'MySQL AB'
# commonName            :PRINTABLE:'MySQL admin'
# Certificate is to be certified until Sep 13 14:22:46 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create client request and key
#
openssl req -new -keyout $DIR/client-key.pem -out \
    $DIR/client-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ....................................++++++
# ..............................................++++++
# writing new private key to '/home/monty/openssl/client-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
```

```
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL user
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove a passphrase from the key (optional)
#
openssl rsa -in $DIR/client-key.pem -out $DIR/client-key.pem

#
# Sign client cert
#

openssl ca  -policy policy_anything -out $DIR/client-cert.pem \
    -config $DIR/openssl.cnf -infiles $DIR/client-req.pem

# Sample output:
# Using configuration from /home/monty/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName           :PRINTABLE:'FI'
# organizationName      :PRINTABLE:'MySQL AB'
# commonName            :PRINTABLE:'MySQL user'
# Certificate is to be certified until Sep 13 16:45:17 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create a my.cnf file that you can use to test the certificates
#

cnf=""
cnf="$cnf [client]"
cnf="$cnf ssl-ca=$DIR/cacert.pem"
cnf="$cnf ssl-cert=$DIR/client-cert.pem"
cnf="$cnf ssl-key=$DIR/client-key.pem"
cnf="$cnf [mysqld]"
cnf="$cnf ssl-ca=$DIR/cacert.pem"
cnf="$cnf ssl-cert=$DIR/server-cert.pem"
cnf="$cnf ssl-key=$DIR/server-key.pem"
echo $cnf | replace " " '
' > $DIR/my.cnf
```

To test SSL connections, start the server as follows, where `$DIR` is the pathname to the directory where

the sample my.cnf option file is located:

```
shell> mysqld --defaults-file=$DIR/my.cnf &
```

Then invoke a client program using the same option file:

```
shell> mysql --defaults-file=$DIR/my.cnf
```

If you have a MySQL source distribution, you can also test your setup by modifying the preceding my.cnf file to refer to the demonstration certificate and key files in the SSL directory of the distribution.

5.6.7.4. SSL GRANT Options

MySQL can check X509 certificate attributes in addition to the usual authentication that is based on the username and password. To specify SSL-related options for a MySQL account, use the REQUIRE clause of the GRANT statement. See Section 13.5.1.3, “GRANT and REVOKE Syntax”.

There are different possibilities for limiting connection types for an account:

- If an account has no SSL or X509 requirements, unencrypted connections are allowed if the username and password are valid. However, encrypted connections also can be used at the client's option, if the client has the proper certificate and key files.
- REQUIRE SSL option limits the server to allow only SSL encrypted connections for the account. Note that this option can be omitted if there are any ACL records that allow non-SSL connections.

  ```
  mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
      -> IDENTIFIED BY 'goodsecret' REQUIRE SSL;
  ```

- REQUIRE X509 means that the client must have a valid certificate but that the exact certificate, issuer, and subject do not matter. The only requirement is that it should be possible to verify its signature with one of the CA certificates.

  ```
  mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
      -> IDENTIFIED BY 'goodsecret' REQUIRE X509;
  ```

- REQUIRE ISSUER 'issuer' places the restriction on connection attempts that the client must present a valid X509 certificate issued by CA 'issuer'. If the client presents a certificate that is valid but has a different issuer, the server rejects the connection. Use of X509 certificates always implies encryption, so the SSL option is unnecessary.

  ```
  mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
      -> IDENTIFIED BY 'goodsecret'
      -> REQUIRE ISSUER '/C=FI/ST=Some-State/L=Helsinki/
         O=MySQL Finland AB/CN=Tonu Samuel/Email=tonu@example.com';
  ```

  Note that the ISSUER value should be entered as a single string.

- REQUIRE SUBJECT 'subject' places the restriction on connection attempts that the client must present a valid X509 certificate with subject 'subject' on it. If the client presents a certificate that is valid but has a different subject, the server rejects the connection.

  ```
  mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
      -> IDENTIFIED BY 'goodsecret'
  ```

```
    -> REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/
       O=MySQL demo client certificate/
       CN=Tonu Samuel/Email=tonu@example.com';
```

Note that the `SUBJECT` value should be entered as a single string.

- `REQUIRE CIPHER 'cipher'` is needed to ensure that strong enough ciphers and key lengths are used. SSL itself can be weak if old algorithms with short encryption keys are used. Using this option, we can ask for some exact cipher method to allow a connection.

  ```
  mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
      -> IDENTIFIED BY 'goodsecret'
      -> REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
  ```

The `SUBJECT`, `ISSUER`, and `CIPHER` options can be combined in the `REQUIRE` clause like this:

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    -> IDENTIFIED BY 'goodsecret'
    -> REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/
       O=MySQL demo client certificate/
       CN=Tonu Samuel/Email=tonu@example.com'
    -> AND ISSUER '/C=FI/ST=Some-State/L=Helsinki/
       O=MySQL Finland AB/CN=Tonu Samuel/Email=tonu@example.com'
    -> AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

Note that the `SUBJECT` and `ISSUER` values each should be entered as a single string.

Starting from MySQL 4.0.4, the `AND` keyword is optional between `REQUIRE` options.

The order of the options does not matter, but no option can be specified twice.

5.6.7.5. SSL Command-Line Options

The following list describes options that are used for specifying the use of SSL, certificate files, and key files. These options are available beginning with MySQL 4.0. They may be given on the command line or in an option file.

- `--ssl`

  For the server, this option specifies that the server allows SSL connections. For a client program, it allows the client to connect to the server using SSL. This option is not sufficient in itself to cause an SSL connection to be used. You must also specify the `--ssl-ca`, `--ssl-cert`, and `--ssl-key` options.

  This option is more often used in its opposite form to indicate that SSL should *not* be used. To do this, specify the option as `--skip-ssl` or `--ssl=0`.

  Note that use of `--ssl` doesn't *require* an SSL connection. For example, if the server or client is compiled without SSL support, a normal unencrypted connection is used.

  The secure way to ensure that an SSL connection is used is to create an account on the server that includes a `REQUIRE SSL` clause in the `GRANT` statement. Then use this account to connect to the server, with both a server and client that have SSL support enabled.

- `--ssl-ca=file_name`

  The path to a file with a list of trusted SSL CAs.

- `--ssl-capath=directory_name`

  The path to a directory that contains trusted SSL CA certificates in pem format.

- `--ssl-cert=file_name`

  The name of the SSL certificate file to use for establishing a secure connection.

- `--ssl-cipher=cipher_list`

  A list of allowable ciphers to use for SSL encryption. `cipher_list` has the same format as the `openssl ciphers` command.

  Example: `--ssl-cipher=ALL:-AES:-EXP`

- `--ssl-key=file_name`

  The name of the SSL key file to use for establishing a secure connection.

5.6.7.6. Connecting to MySQL Remotely from Windows with SSH

Here is a note about how to connect to get a secure connection to remote MySQL server with SSH (by David Carlson `<dcarlson@mplcomm.com>`):

1. Install an SSH client on your Windows machine. As a user, the best non-free one I've found is from `SecureCRT` from http://www.vandyke.com/. Another option is `f-secure` from http://www.f-secure.com/. You can also find some free ones on `Google` at http://directory.google.com/Top/Computers/Security/Products_and_Tools/Cryptography/SSH/Clients/Windows/.

2. Start your Windows SSH client. Set `Host_Name = yourmysqlserver_URL_or_IP`. Set `userid=your_userid` to log in to your server. This `userid` value may not be the same as the username of your MySQL account.

3. Set up port forwarding. Either do a remote forward (Set `local_port: 3306`, `remote_host: yourmysqlservername_or_ip`, `remote_port: 3306`) or a local forward (Set `port: 3306`, `host: localhost`, `remote port: 3306`).

4. Save everything, otherwise you'll have to redo it the next time.

5. Log in to your server with the SSH session you just created.

6. On your Windows machine, start some ODBC application (such as Access).

7. Create a new file in Windows and link to MySQL using the ODBC driver the same way you normally do, except type in `localhost` for the MySQL host server, not `yourmysqlservername`.

You should have an ODBC connection to MySQL, encrypted using SSH.

5.7. Disaster Prevention and Recovery

This section discusses how to make database backups (full and incremental) and how to perform table maintenance. The syntax of the SQL statements described here is given in Section 13.5, “Database Administration Statements”. Much of the information here pertains primarily to `MyISAM` tables. `InnoDB` backup procedures are given in Section 15.9, “Backing Up and Recovering an `InnoDB` Database”.

5.7.1. Database Backups

Because MySQL tables are stored as files, it is easy to do a backup. To get a consistent backup, do a `LOCK TABLES` on the relevant tables, followed by `FLUSH TABLES` for the tables. See Section 13.4.5, “`LOCK TABLES` and `UNLOCK TABLES` Syntax” and Section 13.5.5.2, “`FLUSH` Syntax”. You need only a read lock; this allows other clients to continue to query the tables while you are making a copy of the files in the database directory. The `FLUSH TABLES` statement is needed to ensure that the all active index pages are written to disk before you start the backup.

If you want to make an SQL-level backup of a table, you can use `SELECT INTO ... OUTFILE` or `BACKUP TABLE`. For `SELECT INTO ... OUTFILE`, the output file cannot previously exist. For `BACKUP TABLE`, the same is true as of MySQL 3.23.56 and 4.0.12, because this would be a security risk. See Section 13.1.7, “`SELECT` Syntax” and Section 13.5.2.2, “`BACKUP TABLE` Syntax”.

Another way to back up a database is to use the **mysqldump** program or the **mysqlhotcopy script**. See Section 8.8, “The **mysqldump** Database Backup Program” and Section 8.9, “The **mysqlhotcopy** Database Backup Program”.

1. Do a full backup of your database:

   ```
   shell> mysqldump --tab=/path/to/some/dir --opt db_name
   ```

   Or:

   ```
   shell> mysqlhotcopy db_name /path/to/some/dir
   ```

   You can also simply copy all table files (`*.frm`, `*.MYD`, and `*.MYI` files) as long as the server isn't updating anything. The **mysqlhotcopy** script uses this method. (But note that these methods do not work if your database contains `InnoDB` tables. `InnoDB` does not store table contents in database directories, and **mysqlhotcopy** works only for `MyISAM` and `ISAM` tables.)

2. Stop **mysqld** if it's running, then start it with the `--log-bin[=file_name]` option. See Section 5.9.4, “The Binary Log”. The binary log files provide you with the information you need to replicate changes to the database that are made subsequent to the point at which you executed **mysqldump**.

For `InnoDB` tables, it's possible to perform an online backup that takes no locks on tables; see Section 8.8, “The **mysqldump** Database Backup Program”

MySQL supports incremental backups: You need to start the server with the `--log-bin` option to enable binary logging; see Section 5.9.4, “The Binary Log”. At the moment you want to make an incremental backup (containing all changes that happened since the last full or incremental backup), you should rotate the binary log by using `FLUSH LOGS`. This done, you need to copy to the backup location all binary logs which range from the one of the moment of the last full or incremental backup to the last but one. These binary logs are the incremental backup; at restore time, you apply them as explained further below. The next time you do a full backup, you should also rotate the binary log using `FLUSH LOGS`, `mysqldump --flush-logs`, or `mysqlhotcopy --flushlogs`. See Section 8.8, “The **mysqldump** Database Backup Program” and Section 8.9, “The **mysqlhotcopy** Database Backup Program”.

If your MySQL server is a slave replication server, then regardless of the backup method you choose, you should also back up the `master.info` and `relay-log.info` files when you back up your slave's data. These files are always needed to resume replication after you restore the slave's data. If your slave is subject to replicating `LOAD DATA INFILE` commands, you should also back up any `SQL_LOAD-*` files that may exist in the directory specified by the `--slave-load-tmpdir` option. (This location defaults to the value of the `tmpdir` variable if not specified.) The slave needs these files to resume replication of any interrupted `LOAD DATA INFILE` operations.

If you have to restore `MyISAM` tables, try to recover them using `REPAIR TABLE` or **myisamchk -r** first. That should work in 99.9% of all cases. If **myisamchk** fails, try the following procedure. Note that it works only if you have enabled binary logging by starting MySQL with the `--log-bin` option; see Section 5.9.4, “The Binary Log”.

1. Restore the original **mysqldump** backup, or binary backup.
2. Execute the following command to re-run the updates in the binary logs:

   ```
   shell> mysqlbinlog hostname-bin.[0-9]* | mysql
   ```

   In your case, you may want to re-run only certain binary logs, from certain positions (usually you want to re-run all binary logs from the date of the restored backup, excepting possibly some incorrect queries). See Section 8.5, “The **mysqlbinlog** Binary Log Utility” for more information on the **mysqlbinlog** utility and how to use it.

   If you are using the update logs instead (which is a deprecated feature removed in MySQL 5.0), you can process their contents like this:

   ```
   shell> ls -1 -t -r hostname.[0-9]* | xargs cat | mysql
   ```

   `ls` is used to sort the update log filenames into the right order.

You can also do selective backups of individual files:

- To dump the table, use `SELECT * INTO OUTFILE '`*`file_name`*`' FROM `*`tbl_name`*.
- To reload the table, use and restore with `LOAD DATA INFILE 'file_name' REPLACE ...` To avoid duplicate records, the table must have a `PRIMARY KEY` or a `UNIQUE` index. The `REPLACE` keyword causes old records to be replaced with new ones when a new record duplicates an old record on a unique key value.

If you have performance problems with your server while making backups, one strategy that can help is to set up replication and perform backups on the slave rather than on the master. See Section 6.1, “Introduction to Replication”.

If you are using a Veritas filesystem, you can make a backup like this:

1. From a client program, execute `FLUSH TABLES WITH READ LOCK`.
2. From another shell, execute `mount vxfs snapshot`.
3. From the first client, execute `UNLOCK TABLES`.
4. Copy files from the snapshot.

5. Unmount the snapshot.

5.7.2. Example Backup and Recovery Strategy

This section discusses a procedure for performing backups that allows you to recover data after several types of crashes:

- Operating system crash
- Power failure
- Filesystem crash
- Hardware problem (hard drive, motherboard, and so forth)

The following instructions assume a minimum version of MySQL 4.1.8, because some **mysqldump** options used here are not available in earlier versions.

The example commands do not include options such as `--user` and `--password` for the **mysqldump** and **mysql** programs. You should include such options as necessary so that the MySQL server allows you to connect to it.

We'll consider data is stored into the `InnoDB` storage engine of MySQL, which has support for transactions and automatic crash recovery. We'll always assume the MySQL server is under load at the time of crash. If it were not, no recovery would ever be needed.

For the cases of operating system crashes or power failures, we can assume the MySQL disk data is available after a restart. The `InnoDB` data files do not contain consistent data due to the crash, but `InnoDB` reads its logs and finds in them the list of pending committed and non-committed transactions that have not been flushed to the data files. It automatically rolls back those that were not committed, and flushes to the data files those that were committed. Information about this recovery process is conveyed to the user through the MySQL error log. The following is an example log excerpt:

```
InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 13674004
InnoDB: Doing recovery: scanned up to log sequence number 0 13739520
InnoDB: Doing recovery: scanned up to log sequence number 0 13805056
InnoDB: Doing recovery: scanned up to log sequence number 0 13870592
InnoDB: Doing recovery: scanned up to log sequence number 0 13936128
...
InnoDB: Doing recovery: scanned up to log sequence number 0 20555264
InnoDB: Doing recovery: scanned up to log sequence number 0 20620800
InnoDB: Doing recovery: scanned up to log sequence number 0 20664692
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 16745
InnoDB: Rolling back of trx no 16745 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections
```

For the cases of filesystem crashes or hardware problems, we can assume the MySQL disk data is *not*

available after a restart. This means that MySQL fails to start successfully because some blocks of disk data are no longer readable. In this case, it's necessary to reformat the disk, install a new one, or otherwise correct the underlying problem. Then it's necessary to recover our MySQL data from backups---which means that we must already have made backups. To make sure that is the case, let's step back in time and design a backup policy.

5.7.2.1. Backup Policy

We all know that backups must be scheduled periodically. Full backups (a snapshot of the data at a point in time) can be done in MySQL with several tools. For example, `InnoDB Hot Backup` provides online non-blocking physical backup of the InnoDB data file, and **mysqldump** provides online logical backup. This discussion uses **mysqldump**.

Assume that we make a backup on Sunday at 1 PM, when load is low. The following command makes a full backup of all our `InnoDB` tables in all databases:

```
shell> mysqldump --single-transaction --all-databases > backup_sunday_1_PM.sql
```

This is an online, non-blocking backup that does not disturb the reads and writes on the tables. We assumed earlier that our tables are `InnoDB` tables, so `--single-transaction` uses a consistent read and guarantees that data seen by **mysqldump** does not change. (Changes made by other clients to `InnoDB` tables are not seen by the **mysqldump** process.) If we do also have other types of tables, we must assume that they are not changed during the backup. For example, for the `MyISAM` tables in the `mysql` database, we must assume that no administrative changes are being made to MySQL accounts during the backup.

The resulting `.sql` file produced by the **mysqldump** command contains SQL `INSERT` statements that can be used to reload the dumped tables later.

Full backups are necessary, but they are not always convenient. They produce large backup files and take time to generate. They are not optimal in the sense that each successive full backup includes all data, even that part that didn't change since the previous full backup. After we have made the initial full backup, it is more optimal to make incremental backups. They are smaller and take less time to produce. (The tradeoff is that at recovery time, you do not restore your data just by reloading the full backup. You must also process the incremental backups to recover the incremental changes.)

To make incremental backups, we need to save the incremental changes. The MySQL server should always be started with the `--log-bin` option so that it stores these changes in a file while it updates data. This option enables binary logging, so that the server writes each SQL statement that updates data into a file called a MySQL binary log. Let's look at the data directory of a MySQL server that was started with the `--log-bin` option and that has been running for some days. We find these MySQL binary log files:

```
-rw-rw---- 1 guilhem  guilhem   1277324 Nov 10 23:59 gbichot2-bin.000001
-rw-rw---- 1 guilhem  guilhem         4 Nov 10 23:59 gbichot2-bin.000002
-rw-rw---- 1 guilhem  guilhem        79 Nov 11 11:06 gbichot2-bin.000003
-rw-rw---- 1 guilhem  guilhem       508 Nov 11 11:08 gbichot2-bin.000004
-rw-rw---- 1 guilhem  guilhem 220047446 Nov 12 16:47 gbichot2-bin.000005
-rw-rw---- 1 guilhem  guilhem    998412 Nov 14 10:08 gbichot2-bin.000006
-rw-rw---- 1 guilhem  guilhem       361 Nov 14 10:07 gbichot2-bin.index
```

Each time it restarts, the MySQL server creates a new binary log file using the next number in the sequence. While the server is running, you can also tell it to close the current binary log file and begin a new one manually by issuing a `FLUSH LOGS` SQL statement or with a **mysqladmin flush-logs** command. **mysqldump** also has an option to flush the logs. The `.index` file contains the list of all MySQL binary logs in the directory. This file is used for replication.

The MySQL binary logs are important for recovery, because they are incremental backups. If you make

sure to flush the logs when you make your full backup, then any binary log files created afterward contain all the data changes made since the backup. Let's modify the previous **mysqldump** command a bit so that it flushes the MySQL binary logs at the moment of the full backup, and so that the dump file contains the name of the new current binary log:

```
shell> mysqldump --single-transaction --flush-logs --master-data=2
           --all-databases > backup_sunday_1_PM.sql
```

After executing this command, the the data directory contains a new binary log file, `gbichot2-bin.000007`. The resulting `.sql` file contains these lines:

```
-- Position to start replication or point-in-time recovery from
-- CHANGE MASTER TO MASTER_LOG_FILE='gbichot2-bin.000007',MASTER_LOG_POS=4;
```

Because the **mysqldump** command made a full backup, these lines mean two things:

- The `.sql` file contains all changes made before any changes written to the `gbichot2-bin.000007` binary log file or newer.
- All data changes logged after the backup are not present in the `.sql`, but are present in the `gbichot2-bin.000007` binary log file or newer.

On Monday at 1 PM, we can create an incremental backup by flushing the logs to begin a new binary log file. For example, executing a **mysqladmin flush-logs** command creates `gbichot2-bin.000008`. All changes between the Sunday 1 PM full backup and Monday 1 PM are the file `gbichot2-bin.000007`. This incremental backup is important, so it's a good idea to copy it to a safe place. (For example, back it up on tape or DVD, or copy it to another machine.) On Tuesday 1 PM, execute another **mysqladmin flush-logs** command. All changes between Monday 1 PM and Tuesday 1 PM are the file `gbichot2-bin.000008` (which also should be copied somewhere safe).

The MySQL binary logs take up disk space. To free up space, purge them from time to time. One way to do this is by deleting the binary logs that are no longer needed, such as when we make a full backup:

```
shell> mysqldump --single-transaction --flush-logs --master-data=2
           --all-databases --delete-master-logs > backup_sunday_1_PM.sql
```

Note: Deleting the MySQL binary logs with **mysqldump --delete-master-logs** can be dangerous if your server is a replication master server, because slave servers might not yet fully have processed the contents of the binary log.

The description for the `PURGE MASTER LOGS` statement explains what should be verified before deleting the MySQL binary logs. See Section 13.6.1.1, “`PURGE MASTER LOGS` Syntax”.

5.7.2.2. Using Backups for Recovery

Now suppose that we have a catastrophic crash on Wednesday at 8 AM that requires recovery from backups. To recover, first we restore the last full backup we have (the one from Sunday 1 PM). The full backup file is just a set of SQL statements, so restoring it is very easy:

```
shell> mysql < backup_sunday_1_PM.sql
```

At this point, the data is restored to its state as of Sunday 1 PM. To restore the changes made since then, we must use the incremental backups, that is, the `gbichot2-bin.000007` and `gbichot2-bin.000008` binary log files. Fetch them if necessary from where they were backed up, and

then process their contents like this:

```
shell> mysqlbinlog gbichot2-bin.000007 gbichot2-bin.000008 | mysql
```

We now have recovered the data to its state as of Tuesday 1 PM, but still are missing the changes from that date to the date of the crash. To not miss them, we would have needed to have the MySQL server store its MySQL binary logs into a safe location (RAID disks, SAN, ...) different from the place where it stores its data files, so that these logs were not in the destroyed disk. (That is, we can start the server with a `--log-bin` option that specifies a location on a different physical device than the one on which the data directory resides. That way, the logs are not lost even if the device containing the directory is.) If we had done this, we would have the `gbichot2-bin.000009` at hand, and we could apply it to restore the most recent data changes with no loss how it was at the moment of the crash.

5.7.2.3. Backup Strategy Summary

In case of an operating system crash or power failure, `InnoDB` itself does all the job of recovering data. But to make sure that you can sleep well, observe the following guidelines:

- Always run the MySQL server with the `--log-bin` option, or even `--log-bin=`*`log_name`*, where the log file name is located on some safe media different from the drive on which the data directory is located. If you have such safe media, this can also be good for disk load balancing (which results in a performance improvement).
- Make periodic full backups, using the last **mysqldump** command given earlier that makes an online, non-blocking backup.
- Make periodic incremental backups by flushing the logs with `FLUSH LOGS` or **mysqladmin flush-logs**.

5.7.3. Table Maintenance and Crash Recovery

The following text discusses how to use **myisamchk** to check or repair `MyISAM` tables (tables with `.MYI` and `.MYD` files). The same concepts apply to using **isamchk** to check or repair `ISAM` tables (tables with `.ISM` and `.ISD` files). See Chapter 14, *MySQL Storage Engines and Table Types*.

You can use the **myisamchk** utility to get information about your database tables or to check, repair, or optimize them. The following sections describe how to invoke **myisamchk** (including a description of its options), how to set up a table maintenance schedule, and how to use **myisamchk** to perform its various functions.

Even though table repair with **myisamchk** is quite secure, it's always a good idea to make a backup *before* doing a repair (or any maintenance operation that could make a lot of changes to a table)

myisamchk operations that affect indexes can cause `FULLTEXT` indexes to be rebuilt with full-text parameters that are incompatible with the values used by the MySQL server. To avoid this, read the instructions in Section 5.7.3.2, “General Options for **myisamchk**”.

In many cases, you may find it simpler to do `MyISAM` table maintenance using the SQL statements that perform operations that **myisamchk** can do:

- To check or repair `MyISAM` tables, use `CHECK TABLE` or `REPAIR TABLE`.
- To optimize `MyISAM` tables, use `OPTIMIZE TABLE`.
- To analyze `MyISAM` tables, use `ANALYZE TABLE`.

These statements were introduced in different versions, but all are available from MySQL 3.23.14 on. See Section 13.5.2.1, "`ANALYZE TABLE` Syntax", Section 13.5.2.3, "`CHECK TABLE` Syntax", Section 13.5.2.5, "`OPTIMIZE TABLE` Syntax", and Section 13.5.2.6, "`REPAIR TABLE` Syntax". The statements can be used directly, or by means of the **mysqlcheck** client program, which provides a command-line interface to them.

One advantage of these statements over **myisamchk** is that the server does all the work. With **myisamchk**, you must make sure that the server does not use the tables at the same time. Otherwise, there can be unwanted interaction between **myisamchk** and the server.

5.7.3.1. myisamchk Invocation Syntax

Invoke **myisamchk** like this:

```
shell> myisamchk [options] tbl_name
```

The *`options`* specify what you want **myisamchk** to do. They are described in the following sections. You can also get a list of options by invoking **myisamchk --help**.

With no options, **myisamchk** simply checks your table as the default operation. To get more information or to tell **myisamchk** to take corrective action, specify options as described in the following discussion.

`tbl_name` is the database table you want to check or repair. If you run **myisamchk** somewhere other than in the database directory, you must specify the path to the database directory, because **myisamchk** has no idea where the database is located. In fact, **myisamchk** doesn't actually care whether the files you are working on are located in a database directory. You can copy the files that correspond to a database table into some other location and perform recovery operations on them there.

You can name several tables on the **myisamchk** command line if you wish. You can also specify a table by naming its index file (the file with the `.MYI` suffix). This allows you to specify all tables in a directory by using the pattern `*.MYI`. For example, if you are in a database directory, you can check all the `MyISAM` tables in that directory like this:

```
shell> myisamchk *.MYI
```

If you are not in the database directory, you can check all the tables there by specifying the path to the directory:

```
shell> myisamchk /path/to/database_dir/*.MYI
```

You can even check all tables in all databases by specifying a wildcard with the path to the MySQL data directory:

```
shell> myisamchk /path/to/datadir/*/*.MYI
```

The recommended way to quickly check all `MyISAM` and `ISAM` tables is:

```
shell> myisamchk --silent --fast /path/to/datadir/*/*.MYI
shell> isamchk --silent /path/to/datadir/*/*.ISM
```

If you want to check all `MyISAM` and `ISAM` tables and repair any that are corrupted, you can use the following commands:

```
shell> myisamchk --silent --force --fast --update-state \
```

```
          -O key_buffer=64M -O sort_buffer=64M \
          -O read_buffer=1M -O write_buffer=1M \
          /path/to/datadir/*/*.MYI
shell> isamchk --silent --force -O key_buffer=64M \
          -O sort_buffer=64M -O read_buffer=1M -O write_buffer=1M \
          /path/to/datadir/*/*.ISM
```

These commands assume that you have more than 64MB free. For more information about memory allocation with **myisamchk**, see Section 5.7.3.6, "**myisamchk** Memory Usage".

You must ensure that no other program is using the tables while you are running **myisamchk**. Otherwise, when you run **myisamchk**, it may display the following error message:

```
warning: clients are using or haven't closed the table properly
```

This means that you are trying to check a table that has been updated by another program (such as the **mysqld** server) that hasn't yet closed the file or that has died without closing the file properly.

If **mysqld** is running, you must force it to flush any table modifications that are still buffered in memory by using `FLUSH TABLES`. You should then ensure that no one is using the tables while you are running **myisamchk**. The easiest way to avoid this problem is to use `CHECK TABLE` instead of **myisamchk** to check tables.

5.7.3.2. General Options for myisamchk

The options described in this section can be used for any type of table maintenance operation performed by **myisamchk**. The sections following this one describe options that pertain only to specific operations, such as table checking or repairing.

- `--help, -?`

  Display a help message and exit.

- `--debug=`*`debug_options`*`, -# `*`debug_options`*

  Write a debugging log. The *`debug_options`* string often is `'d:t:o,`*`file_name`*`'`.

- `--silent, -s`

  Silent mode. Write output only when errors occur. You can use `-s` twice (`-ss`) to make **myisamchk** very silent.

- `--verbose, -v`

  Verbose mode. Print more information. This can be used with `-d` and `-e`. Use `-v` multiple times (`-vv`, `-vvv`) for even more output.

- `--version, -V`

  Display version information and exit.

- `--wait, -w`

  Instead of terminating with an error if the table is locked, wait until the table is unlocked before continuing. Note that if you are running **mysqld** with the `--skip-external-locking` option, the table can be locked only by another **myisamchk** command.

You can also set the following variables by using `--var_name=value` options:

Variable	Default Value
`decode_bits`	9
`ft_max_word_len`	version-dependent
`ft_min_word_len`	4
`ft_stopword_file`	built-in list
`key_buffer_size`	523264
`myisam_block_size`	1024
`read_buffer_size`	262136
`sort_buffer_size`	2097144
`sort_key_blocks`	16
`write_buffer_size`	262136

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. However, this syntax is deprecated as of MySQL 4.0.

The possible **myisamchk** variables and their default values can be examined with **myisamchk --help**:

`sort_buffer_size` is used when the keys are repaired by sorting keys, which is the normal case when you use `--recover`.

`key_buffer_size` is used when you are checking the table with `--extend-check` or when the keys are repaired by inserting keys row by row into the table (like when doing normal inserts). Repairing through the key buffer is used in the following cases:

- You use `--safe-recover`.
- The temporary files needed to sort the keys would be more than twice as big as when creating the key file directly. This is often the case when you have large key values for `CHAR`, `VARCHAR`, or `TEXT` columns, because the sort operation needs to store the complete key values as it proceeds. If you have lots of temporary space and you can force **myisamchk** to repair by sorting, you can use the `--sort-recover` option.

Repairing through the key buffer takes much less disk space than using sorting, but is also much slower.

If you want a faster repair, set the `key_buffer_size` and `sort_buffer_size` variables to about 25% of your available memory. You can set both variables to large values, because only one of them is used at a time.

`myisam_block_size` is the size used for index blocks. It is available as of MySQL 4.0.0.

The `ft_min_word_len` and `ft_max_word_len` variables are available as of MySQL 4.0.0. `ft_stopword_file` is available as of MySQL 4.0.19.

`ft_min_word_len` and `ft_max_word_len` indicate the minimum and maximum word length for `FULLTEXT` indexes. `ft_stopword_file` names the stopword file. These need to be set under the following circumstances.

If you use **myisamchk** to perform an operation that modifies table indexes (such as repair or analyze), the `FULLTEXT` indexes are rebuilt using the default full-text parameter values for minimum and maximum word length and the stopword file unless you specify otherwise. This can result in queries failing.

The problem occurs because these parameters are known only by the server. They are not stored in `MyISAM` index files. To avoid the problem if you have modified the minimum or maximum word length or the stopword file in the server, specify the same `ft_min_word_len`, `ft_max_word_len`, and `ft_stopword_file` values to **myisamchk** that you use for **mysqld**. For example, if you have set the minimum word length to 3, you can repair a table with **myisamchk** like this:

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

To ensure that **myisamchk** and the server use the same values for full-text parameters, you can place each one in both the `[mysqld]` and `[myisamchk]` sections of an option file:

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

An alternative to using **myisamchk** is to use the `REPAIR TABLE`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, or `ALTER TABLE`. These statements are performed by the server, which knows the proper full-text parameter values to use.

5.7.3.3. Check Options for myisamchk

myisamchk supports the following options for table checking operations:

- `--check, -c`

  Check the table for errors. This is the default operation if you specify no option that selects an operation type explicitly.

- `--check-only-changed, -C`

  Check only tables that have changed since the last check.

- `--extend-check, -e`

  Check the table very thoroughly. This is quite slow if the table has many indexes. This option should only be used in extreme cases. Normally, **myisamchk** or **myisamchk --medium-check** should be able to determine whether there are any errors in the table.

  If you are using `--extend-check` and have plenty of memory, setting the `key_buffer_size` variable to a large value helps the repair operation run faster.

- `--fast, -F`

  Check only tables that haven't been closed properly.

- `--force, -f`

  Do a repair operation automatically if **myisamchk** finds any errors in the table. The repair type is the same as that specified with the `--repair` or `-r` option.

- `--information, -i`

  Print informational statistics about the table that is checked.

- `--medium-check, -m`

Do a check that is faster than an `--extend-check` operation. This finds only 99.99% of all errors, which should be good enough in most cases.

- `--read-only, -T`

  Don't mark the table as checked. This is useful if you use **myisamchk** to check a table that is in use by some other application that doesn't use locking, such as **mysqld** when run with the `--skip-external-locking` option.

- `--update-state, -U`

  Store information in the `.MYI` file to indicate when the table was checked and whether the table crashed. This should be used to get full benefit of the `--check-only-changed` option, but you shouldn't use this option if the **mysqld** server is using the table and you are running it with the `--skip-external-locking` option.

5.7.3.4. Repair Options for myisamchk

myisamchk supports the following options for table repair operations:

- `--backup, -B`

  Make a backup of the `.MYD` file as `file_name-time.BAK`

- `--character-sets-dir=`*`path`*

  The directory where character sets are installed. See Section 5.8.1, “The Character Set Used for Data and Sorting”.

- `--correct-checksum`

  Correct the checksum information for the table.

- `--data-file-length=#, -D #`

  Maximum length of the data file (when re-creating data file when it's ``full'').

- `--extend-check, -e`

  Do a repair that tries to recover every possible row from the data file. Normally this also finds a lot of garbage rows. Don't use this option unless you are totally desperate.

- `--force, -f`

  Overwrite old temporary files (files with names like *`tbl_name`*`.TMD`) instead of aborting.

- `--keys-used=#, -k #`

  For **myisamchk**, the option value indicates which indexes to update. Each binary bit of the option value corresponds to a table index, where the first index is bit 0. For **isamchk**, the option value indicates that only the first # of the table indexes should be updated. In either case, an option value of 0 disables updates to all indexes, which can be used to get faster inserts. Deactivated indexes can be reactivated by using **myisamchk -r** or (**isamchk -r**).

- `--no-symlinks, -l`

Do not follow symbolic links. Normally **myisamchk** repairs the table that a symlink points to. This option doesn't exist as of MySQL 4.0, because versions from 4.0 on do not remove symlinks during repair operations.

- `--parallel-recover, -p`

  Uses the same technique as `-r` and `-n`, but creates all the keys in parallel, using different threads. This option was added in MySQL 4.0.2. *This is alpha code. Use at your own risk!*

- `--quick, -q`

  Achieve a faster repair by not modifying the data file. You can specify this option twice to force **myisamchk** to modify the original data file in case of duplicate keys.

- `--recover, -r`

  Do a repair that can fix almost any problem except unique keys that aren't unique (which is an extremely unlikely error with `ISAM`/`MyISAM` tables). If you want to recover a table, this is the option to try first. You should try `-o` only if **myisamchk** reports that the table can't be recovered by `-r`. (In the unlikely case that `-r` fails, the data file is still intact.)

  If you have lots of memory, you should increase the value of `sort_buffer_size`.

- `--safe-recover, -o`

  Do a repair using an old recovery method that reads through all rows in order and updates all index trees based on the rows found. This is an order of magnitude slower than `-r`, but can handle a couple of very unlikely cases that `-r` cannot. This recovery method also uses much less disk space than `-r`. Normally, you should repair first with `-r`, and then with `-o` only if `-r` fails.

  If you have lots of memory, you should increase the value of `key_buffer_size`.

- `--set-character-set=`*`name`*

  Change the character set used by the table indexes. This option was replaced by `--set-collation` in MySQL 4.1.1/5.0.3.

- `--set-collation=`*`name`*

  Change the collation used to sort table indexes. The character set name is implied by the first part of the collation name. This option was added in MySQL 4.1.11/5.0.3.

- `--sort-recover, -n`

  Force **myisamchk** to use sorting to resolve the keys even if the temporary files should be very big.

- `--tmpdir=`*`path`*`, -t `*`path`*

  Path of the directory to be used for storing temporary files. If this is not set, **myisamchk** uses the value of the `TMPDIR` environment variable. Starting from MySQL 4.1, `tmpdir` can be set to a list of directory paths that are used successively in round-robin fashion for creating temporary files. The separator character between directory names should be colon (`':'`) on Unix and semicolon (`';'`) on Windows, NetWare, and OS/2.

- `--unpack, -u`

  Unpack a table that was packed with **myisampack**.

5.7.3.5. Other Options for myisamchk

myisamchk supports the following options for actions other than table checks and repairs:

- `--analyze, -a`

  Analyze the distribution of keys. This improves join performance by enabling the join optimizer to better choose the order in which to join the tables and which keys it should use. To obtain information about the distribution, use a **myisamchk --description --verbose** ***tbl_name*** command or the `SHOW KEYS FROM` *`tbl_name`* statement.

- `--description, -d`

  Print some descriptive information about the table.

- `--set-auto-increment[=value], -A[value]`

  Force `AUTO_INCREMENT` numbering for new records to start at the given value (or higher, if there are existing records with `AUTO_INCREMENT` values this large). If *`value`* is not specified, `AUTO_INCREMENT` number for new records begins with the largest value currently in the table, plus one.

- `--sort-index, -S`

  Sort the index tree blocks in high-low order. This optimizes seeks and makes table scanning by key faster.

- `--sort-records=#, -R #`

  Sort records according to a particular index. This makes your data much more localized and may speed up range-based `SELECT` and `ORDER BY` operations that use this index. (The first time you use this option to sort a table, it may be very slow.) To determine a table's index numbers, use `SHOW KEYS`, which displays a table's indexes in the same order that **myisamchk** sees them. Indexes are numbered beginning with 1.

5.7.3.6. myisamchk Memory Usage

Memory allocation is important when you run **myisamchk**. **myisamchk** uses no more memory than you specify with the `-O` options. If you are going to use **myisamchk** on very large tables, you should first decide how much memory you want it to use. The default is to use only about 3MB to perform repairs. By using larger values, you can get **myisamchk** to operate faster. For example, if you have more than 32MB RAM, you could use options such as these (in addition to any other options you might specify):

```
shell> myisamchk -O sort=16M -O key=16M -O read=1M -O write=1M ...
```

Using `-O sort=16M` should probably be enough for most cases.

Be aware that **myisamchk** uses temporary files in `TMPDIR`. If `TMPDIR` points to a memory filesystem, you may easily get out of memory errors. If this happens, set `TMPDIR` to point at some directory located on a filesystem with more space and run **myisamchk** again.

When repairing, **myisamchk** also needs a lot of disk space:

- Double the size of the data file (the original one and a copy). This space is not needed if you do a repair with `--quick`; in this case, only the index file is re-created. This space is needed on the same

filesystem as the original data file! (The copy is created in the same directory as the original.)

- Space for the new index file that replaces the old one. The old index file is truncated at the start of the repair operation, so you usually ignore this space. This space is needed on the same filesystem as the original index file!
- When using `--recover` or `--sort-recover` (but not when using `--safe-recover`), you need space for a sort buffer. The amount of space required is:

  ```
  (largest_key + row_pointer_length) * number_of_rows * 2
  ```

  You can check the length of the keys and the `row_pointer_length` with **myisamchk -dv *tbl_name***. This space is allocated in the temporary directory (specified by `TMPDIR` or `--tmpdir=path`).

If you have a problem with disk space during repair, you can try to use `--safe-recover` instead of `--recover`.

5.7.3.7. Using myisamchk for Crash Recovery

If you run **mysqld** with `--skip-external-locking` (which is the default on some systems, such as Linux), you can't reliably use **myisamchk** to check a table when **mysqld** is using the same table. If you can be sure that no one is accessing the tables through **mysqld** while you run **myisamchk**, you only have to do **mysqladmin flush-tables** before you start checking the tables. If you can't guarantee this, then you must stop **mysqld** while you check the tables. If you run **myisamchk** while **mysqld** is updating the tables, you may get a warning that a table is corrupt even when it isn't.

If you are not using `--skip-external-locking`, you can use **myisamchk** to check tables at any time. While you do this, all clients that try to update the table wait until **myisamchk** is ready before continuing.

If you use **myisamchk** to repair or optimize tables, you *must* always ensure that the **mysqld** server is not using the table (this also applies if you are using `--skip-external-locking`). If you don't take down **mysqld**, you should at least do a **mysqladmin flush-tables** before you run **myisamchk**. Your tables *may become corrupted* if the server and **myisamchk** access the tables simultaneously.

This section describes how to check for and deal with data corruption in MySQL databases. If your tables get corrupted frequently you should try to find the reason why. See Section A.4.2, “What to Do If MySQL Keeps Crashing”.

The `MyISAM` table section contains reason for why a table could be corrupted. See Section 14.1.4, “`MyISAM` Table Problems”.

When performing crash recovery, it is important to understand that each `MyISAM` table *`tbl_name`* in a database corresponds to three files in the database directory:

File	Purpose
`tbl_name``.frm`	Definition (format) file
`tbl_name``.MYD`	Data file
`tbl_name``.MYI`	Index file

Each of these three file types is subject to corruption in various ways, but problems occur most often in data files and index files.

myisamchk works by creating a copy of the `.MYD` data file row by row. It ends the repair stage by removing the old `.MYD` file and renaming the new file to the original file name. If you use `--quick`, **myisamchk** does not create a temporary `.MYD` file, but instead assumes that the `.MYD` file is correct and only generates a new index file without touching the `.MYD` file. This is safe, because **myisamchk** automatically detects whether the `.MYD` file is corrupt and aborts the repair if it is. You can also specify the `--quick` option twice to **myisamchk**. In this case, **myisamchk** does not abort on some errors (such as duplicate-key errors) but instead tries to resolve them by modifying the `.MYD` file. Normally the use of two `--quick` options is useful only if you have too little free disk space to perform a normal repair. In this case, you should at least make a backup before running **myisamchk**.

5.7.3.8. How to Check `MyISAM` Tables for Errors

To check a `MyISAM` table, use the following commands:

- `myisamchk` *`tbl_name`*

  This finds 99.99% of all errors. What it can't find is corruption that involves *only* the data file (which is very unusual). If you want to check a table, you should normally run **myisamchk** without options or with either the `-s` or `--silent` option.

- `myisamchk -m` *`tbl_name`*

  This finds 99.999% of all errors. It first checks all index entries for errors and then reads through all rows. It calculates a checksum for all keys in the rows and verifies that the checksum matches the checksum for the keys in the index tree.

- `myisamchk -e` *`tbl_name`*

  This does a complete and thorough check of all data (`-e` means ``extended check''). It does a check-read of every key for each row to verify that they indeed point to the correct row. This may take a long time for a large table that has many indexes. Normally, **myisamchk** stops after the first error it finds. If you want to obtain more information, you can add the `--verbose` (`-v`) option. This causes **myisamchk** to keep going, up through a maximum of 20 errors.

- `myisamchk -e -i` *`tbl_name`*

  Like the previous command, but the `-i` option tells **myisamchk** to print some informational statistics, too.

In most cases, a simple **myisamchk** with no arguments other than the table name is sufficient to check a table.

5.7.3.9. How to Repair Tables

The discussion in this section describes how to use **myisamchk** on `MyISAM` tables (extensions `.MYI` and `.MYD`). If you are using `ISAM` tables (extensions `.ISM` and `.ISD`), you should use **isamchk** instead; the concepts are similar.

If you are using MySQL 3.23.16 and above, you can (and should) use the `CHECK TABLE` and `REPAIR TABLE` statements to check and repair `MyISAM` tables. See Section 13.5.2.3, “`CHECK TABLE` Syntax” and Section 13.5.2.6, “`REPAIR TABLE` Syntax”.

The symptoms of a corrupted table include queries that abort unexpectedly and observable errors such as these:

- *tbl_name*.frm is locked against change
- Can't find file *tbl_name*.MYI (Errcode: *###*)
- Unexpected end of file
- Record file is crashed
- Got error *###* from table handler

To get more information about the error you can run **perror** *###*, where *###* is the error number. The following example shows how to use **perror** to find the meanings for the most common error numbers that indicate a problem with a table:

```
shell> perror 126 127 132 134 135 136 141 144 145
126 = Index file is crashed / Wrong file format
127 = Record-file is crashed
132 = Old database file
134 = Record was already deleted (or record file crashed)
135 = No more room in record file
136 = No more room in index file
141 = Duplicate unique key or constraint on write or update
144 = Table is crashed and last repair failed
145 = Table was marked as crashed and should be repaired
```

Note that error 135 (no more room in record file) and error 136 (no more room in index file) are not errors that can be fixed by a simple repair. In this case, you have to use ALTER TABLE to increase the MAX_ROWS and AVG_ROW_LENGTH table option values:

```
ALTER TABLE tbl_name MAX_ROWS=xxx AVG_ROW_LENGTH=yyy;
```

If you don't know the current table option values, use SHOW CREATE TABLE tbl_name.

For the other errors, you must repair your tables. **myisamchk** can usually detect and fix most problems that occur.

The repair process involves up to four stages, described here. Before you begin, you should change location to the database directory and check the permissions of the table files. On Unix, make sure that they are readable by the user that **mysqld** runs as (and to you, because you need to access the files you are checking). If it turns out you need to modify files, they must also be writable by you.

The options that you can use for table maintenance with **myisamchk** and **isamchk** are described in several of the earlier subsections of Section 5.7.3, “Table Maintenance and Crash Recovery”.

The following section is for the cases where the above command fails or if you want to use the extended features that **myisamchk** and **isamchk** provide.

If you are going to repair a table from the command line, you must first stop the **mysqld** server. Note that when you do **mysqladmin shutdown** on a remote server, the **mysqld** server is still alive for a while after **mysqladmin** returns, until all queries are stopped and all keys have been flushed to disk.

Stage 1: Checking your tables

Run **myisamchk *.MYI** or **myisamchk -e *.MYI** if you have more time. Use the -s (silent) option to suppress unnecessary information.

If the **mysqld** server is down, you should use the --update-state option to tell **myisamchk** to

mark the table as 'checked'.

You have to repair only those tables for which **myisamchk** announces an error. For such tables, proceed to Stage 2.

If you get weird errors when checking (such as `out of memory` errors), or if **myisamchk** crashes, go to Stage 3.

Stage 2: Easy safe repair

Note: If you want a repair operation to go much faster, you should set the values of the `sort_buffer_size` and `key_buffer_size` variables each to about 25% of your available memory when running **myisamchk** or **isamchk**.

First, try **myisamchk -r -q** ***tbl_name*** (`-r -q` means ``quick recovery mode''). This attempts to repair the index file without touching the data file. If the data file contains everything that it should and the delete links point at the correct locations within the data file, this should work, and the table is fixed. Start repairing the next table. Otherwise, use the following procedure:

1. Make a backup of the data file before continuing.
2. Use **myisamchk -r** ***tbl_name*** (`-r` means ``recovery mode''). This removes incorrect records and deleted records from the data file and reconstructs the index file.
3. If the preceding step fails, use **myisamchk --safe-recover** ***tbl_name***. Safe recovery mode uses an old recovery method that handles a few cases that regular recovery mode doesn't (but is slower).

If you get weird errors when repairing (such as `out of memory` errors), or if **myisamchk** crashes, go to Stage 3.

Stage 3: Difficult repair

You should reach this stage only if the first 16KB block in the index file is destroyed or contains incorrect information, or if the index file is missing. In this case, it's necessary to create a new index file. Do so as follows:

1. Move the data file to some safe place.
2. Use the table description file to create new (empty) data and index files:

   ```
   shell> mysql db_name
   mysql> SET AUTOCOMMIT=1;
   mysql> TRUNCATE TABLE tbl_name;
   mysql> quit
   ```

   If your version of MySQL doesn't have `TRUNCATE TABLE`, use `DELETE FROM tbl_name` instead.
3. Copy the old data file back onto the newly created data file. (Don't just move the old file back onto the new file; you want to retain a copy in case something goes wrong.)

Go back to Stage 2. **myisamchk -r -q** should work. (This shouldn't be an endless loop.)

As of MySQL 4.0.2, you can also use `REPAIR TABLE tbl_name USE_FRM`, which performs the whole procedure automatically.

Stage 4: Very difficult repair

You should reach this stage only if the `.frm` description file has also crashed. That should never happen, because the description file isn't changed after the table is created:

1. Restore the description file from a backup and go back to Stage 3. You can also restore the index file and go back to Stage 2. In the latter case, you should start with **myisamchk -r**.
2. If you don't have a backup but know exactly how the table was created, create a copy of the table in another database. Remove the new data file, then move the `.frm` description and `.MYI` index files from the other database to your crashed database. This gives you new description and index files, but leaves the `.MYD` data file alone. Go back to Stage 2 and attempt to reconstruct the index file.

5.7.3.10. Table Optimization

To coalesce fragmented records and eliminate wasted space resulting from deleting or updating records, run **myisamchk** in recovery mode:

```
shell> myisamchk -r tbl_name
```

You can optimize a table in the same way by using the SQL `OPTIMIZE TABLE` statement. `OPTIMIZE TABLE` does a repair of the table and a key analysis, and also sorts the index tree to give faster key lookups. There is also no possibility of unwanted interaction between a utility and the server, because the server does all the work when you use `OPTIMIZE TABLE`. See Section 13.5.2.5, “`OPTIMIZE TABLE` Syntax”.

myisamchk also has a number of other options you can use to improve the performance of a table:

- `-S`, `--sort-index`
- `-R` *`index_num`*, `--sort-records=`*`index_num`*
- `-a`, `--analyze`

For a full description of the options, see Section 5.7.3.1, “**myisamchk** Invocation Syntax”.

5.7.4. Setting Up a Table Maintenance Schedule

It is a good idea to perform table checks on a regular basis rather than waiting for problems to occur. One way to check and repair `MyISAM` tables is with the `CHECK TABLE` and `REPAIR TABLE` statements. These are available starting with MySQL 3.23.16. See Section 13.5.2.3, “`CHECK TABLE` Syntax” and Section 13.5.2.6, “`REPAIR TABLE` Syntax”.

Another way to check tables is to use **myisamchk**. For maintenance purposes, you can use **myisamchk -s**. The `-s` option (short for `--silent`) causes **myisamchk** to run in silent mode, printing messages only when errors occur.

It's also a good idea to check tables when the server starts. For example, whenever the machine has done a restart in the middle of an update, you usually need to check all the tables that could have been affected. (These are ``expected crashed tables.'') To check `MyISAM` tables automatically, start the server with the `--myisam-recover` option, available as of MySQL 3.23.25. If your server is too old to support this option, you could add a test to **mysqld_safe** that runs **myisamchk** to check all tables that have been modified during the last 24 hours if there is an old `.pid` (process ID) file left after a restart. (The

`.pid` file is created by **mysqld** when it starts and removed when it terminates normally. The presence of a `.pid` file at system startup time indicates that **mysqld** terminated abnormally.)

An even better test would be to check any table whose last-modified time is more recent than that of the `.pid` file.

You should also check your tables regularly during normal system operation. At MySQL AB, we run a **cron** job to check all our important tables once a week, using a line like this in a `crontab` file:

```
35 0 * * 0 /path/to/myisamchk --fast --silent /path/to/datadir/*/*.MYI
```

This prints out information about crashed tables so that we can examine and repair them when needed.

Because we haven't had any unexpectedly crashed tables (tables that become corrupted for reasons other than hardware trouble) for a couple of years (this is really true), once a week is more than enough for us.

We recommend that to start with, you execute **myisamchk -s** each night on all tables that have been updated during the last 24 hours, until you come to trust MySQL as much as we do.

Normally, MySQL tables need little maintenance. If you are changing `MyISAM` tables with dynamic-sized rows (tables with `VARCHAR`, `BLOB`, or `TEXT` columns) or have tables with many deleted rows you may want to defragment/reclaim space from the tables from time to time (once a month?).

You can do this by using `OPTIMIZE TABLE` on the tables in question. Or, if you can stop the **mysqld** server for a while, change location into the data directory and use this command while the server is stopped:

```
shell> myisamchk -r -s --sort-index -O sort_buffer_size=16M */*.MYI
```

For `ISAM` tables, the command is similar:

```
shell> isamchk -r -s --sort-index -O sort_buffer_size=16M */*.ISM
```

5.7.5. Getting Information About a Table

To obtain a description of a table or statistics about it, use the commands shown here. We explain some of the information in more detail later:

- **myisamchk -d *tbl_name***

  Runs **myisamchk** in ``describe mode'' to produce a description of your table. If you start the MySQL server using the `--skip-external-locking` option, **myisamchk** may report an error for a table that is updated while it runs. However, because **myisamchk** doesn't change the table in describe mode, there is no risk of destroying data.

- **myisamchk -d -v *tbl_name***

  Adding `-v` runs **myisamchk** in verbose mode so that it produces more information about what it is doing.

- **myisamchk -eis *tbl_name***

  Shows only the most important information from a table. This operation is slow because it must read the entire table.

- **myisamchk -eiv *tbl_name***

This is like `-eis`, but tells you what is being done.

Sample output for some of these commands follows. They are based on a table with these data and index file sizes:

```
-rw-rw-r--   1 monty    tcx     317235748 Jan 12 17:30 company.MYD
-rw-rw-r--   1 davida   tcx      96482304 Jan 12 18:35 company.MYM
```

Example of **myisamchk -d** output:

```
MyISAM file:     company.MYI
Record format:   Fixed length
Data records:    1403698  Deleted blocks:         0
Recordlength:    226

table description:
Key Start Len Index   Type
1   2     8   unique  double
2   15    10  multip. text packed stripped
3   219   8   multip. double
4   63    10  multip. text packed stripped
5   167   2   multip. unsigned short
6   177   4   multip. unsigned long
7   155   4   multip. text
8   138   4   multip. unsigned long
9   177   4   multip. unsigned long
    193   1           text
```

Example of **myisamchk -d -v** output:

```
MyISAM file:         company
Record format:       Fixed length
File-version:        1
Creation time:       1999-10-30 12:12:51
Recover time:        1999-10-31 19:13:01
Status:              checked
Data records:            1403698  Deleted blocks:              0
Datafile parts:          1403698  Deleted data:                0
Datafile pointer (bytes):      3  Keyfile pointer (bytes):     3
Max datafile length:  3791650815  Max keyfile length: 4294967294
Recordlength:                226

table description:
Key Start Len Index   Type                     Rec/key     Root Blocksize
1   2     8   unique  double                         1 15845376      1024
2   15    10  multip. text packed stripped           2 25062400      1024
3   219   8   multip. double                        73 40907776      1024
4   63    10  multip. text packed stripped           5 48097280      1024
5   167   2   multip. unsigned short              4840 55200768      1024
6   177   4   multip. unsigned long               1346 65145856      1024
7   155   4   multip. text                        4995 75090944      1024
8   138   4   multip. unsigned long                 87 85036032      1024
9   177   4   multip. unsigned long                178 96481280      1024
    193   1           text
```

Example of **myisamchk -eis** output:

```
Checking MyISAM file: company
```

```
Key:  1:  Keyblocks used:  97%  Packed:    0%  Max levels:  4
Key:  2:  Keyblocks used:  98%  Packed:   50%  Max levels:  4
Key:  3:  Keyblocks used:  97%  Packed:    0%  Max levels:  4
Key:  4:  Keyblocks used:  99%  Packed:   60%  Max levels:  3
Key:  5:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
Key:  6:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
Key:  7:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
Key:  8:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
Key:  9:  Keyblocks used:  98%  Packed:    0%  Max levels:  4
Total:    Keyblocks used:  98%  Packed:   17%

Records:          1403698    M.recordlength:     226
Packed:             0%
Recordspace used:     100%   Empty space:          0%
Blocks/Record:   1.00
Record blocks:    1403698    Delete blocks:        0
Recorddata:     317235748    Deleted data:         0
Lost space:             0    Linkdata:             0

User time 1626.51, System time 232.36
Maximum resident set size 0, Integral resident set size 0
Non physical pagefaults 0, Physical pagefaults 627, Swaps 0
Blocks in 0 out 0, Messages in 0 out 0, Signals 0
Voluntary context switches 639, Involuntary context switches 28966
```

Example of **myisamchk -eiv** output:

```
Checking MyISAM file: company
Data records: 1403698   Deleted blocks:       0
- check file-size
- check delete-chain
block_size 1024:
index  1:
index  2:
index  3:
index  4:
index  5:
index  6:
index  7:
index  8:
index  9:
No recordlinks
- check index reference
- check data record references index: 1
Key:  1:  Keyblocks used:  97%  Packed:    0%  Max levels:  4
- check data record references index: 2
Key:  2:  Keyblocks used:  98%  Packed:   50%  Max levels:  4
- check data record references index: 3
Key:  3:  Keyblocks used:  97%  Packed:    0%  Max levels:  4
- check data record references index: 4
Key:  4:  Keyblocks used:  99%  Packed:   60%  Max levels:  3
- check data record references index: 5
Key:  5:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
- check data record references index: 6
Key:  6:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
- check data record references index: 7
Key:  7:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
- check data record references index: 8
Key:  8:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
- check data record references index: 9
Key:  9:  Keyblocks used:  98%  Packed:    0%  Max levels:  4
Total:    Keyblocks used:   9%  Packed:   17%
```

```
- check records and index references
[LOTS OF ROW NUMBERS DELETED]

Records:         1403698   M.recordlength:   226   Packed:           0%
Recordspace used:    100%  Empty space:        0%  Blocks/Record: 1.00
Record blocks:   1403698   Delete blocks:      0
Recorddata:    317235748   Deleted data:       0
Lost space:            0   Linkdata:           0

User time 1639.63, System time 251.61
Maximum resident set size 0, Integral resident set size 0
Non physical pagefaults 0, Physical pagefaults 10580, Swaps 0
Blocks in 4 out 0, Messages in 0 out 0, Signals 0
Voluntary context switches 10604, Involuntary context switches 122798
```

Explanations for the types of information **myisamchk** produces are given here. ``Keyfile'' refers to the index file. ``Record'' and ``row'' are synonymous.

- `MyISAM file`

  Name of the `MyISAM` (index) file.

- `File-version`

  Version of `MyISAM` format. Currently always 2.

- `Creation time`

  When the data file was created.

- `Recover time`

  When the index/data file was last reconstructed.

- `Data records`

  How many records are in the table.

- `Deleted blocks`

  How many deleted blocks still have reserved space. You can optimize your table to minimize this space. See Section 5.7.3.10, “Table Optimization”.

- `Datafile parts`

  For dynamic record format, this indicates how many data blocks there are. For an optimized table without fragmented records, this is the same as `Data records`.

- `Deleted data`

  How many bytes of unreclaimed deleted data there are. You can optimize your table to minimize this space. See Section 5.7.3.10, “Table Optimization”.

- `Datafile pointer`

  The size of the data file pointer, in bytes. It is usually 2, 3, 4, or 5 bytes. Most tables manage with 2 bytes, but this cannot be controlled from MySQL yet. For fixed tables, this is a record address. For dynamic tables, this is a byte address.

- `Keyfile pointer`

  The size of the index file pointer, in bytes. It is usually 1, 2, or 3 bytes. Most tables manage with 2 bytes, but this is calculated automatically by MySQL. It is always a block address.

- `Max datafile length`

  How long the table data file can become, in bytes.

- `Max keyfile length`

  How long the table index file can become, in bytes.

- `Recordlength`

  How much space each record takes, in bytes.

- `Record format`

  The format used to store table rows. The preceding examples use `Fixed length`. Other possible values are `Compressed` and `Packed`.

- `table description`

  A list of all keys in the table. For each key, **myisamchk** displays some low-level information:

  - `Key`

    This key's number.

  - `Start`

    Where in the record this index part starts.

  - `Len`

    How long this index part is. For packed numbers, this should always be the full length of the column. For strings, it may be shorter than the full length of the indexed column, because you can index a prefix of a string column.

  - `Index`

    Whether a key value can exist multiple times in the index. Values are `unique` or `multip.` (multiple).

  - `Type`

    What data type this index part has. This is a `MyISAM` data type with the options `packed`, `stripped`, or `empty`.

  - `Root`

    Address of the root index block.

  - `Blocksize`

    The size of each index block. By default this is 1024, but the value may be changed at compile time when MySQL is built from source.

  - `Rec/key`

This is a statistical value used by the optimizer. It tells how many records there are per value for this key. A unique key always has a value of 1. This may be updated after a table is loaded (or greatly changed) with **myisamchk -a**. If this is not updated at all, a default value of 30 is given.

For the table shown in the examples, there are two `table description` lines for the ninth index. This indicates that it is a multiple-part index with two parts.

- `Keyblocks used`

  What percentage of the keyblocks are used. When a table has just been reorganized with **myisamchk**, as for the table in the examples, the values are very high (very near the theoretical maximum).

- `Packed`

  MySQL tries to pack keys with a common suffix. This can only be used for indexes on `CHAR`, `VARCHAR`, or `DECIMAL` columns. For long indexed strings that have similar leftmost parts, this can significantly reduce the space used. In the third example above, the fourth key is 10 characters long and a 60% reduction in space is achieved.

- `Max levels`

  How deep the B-tree for this key is. Large tables with long key values get high values.

- `Records`

  How many rows are in the table.

- `M.recordlength`

  The average record length. This is the exact record length for tables with fixed-length records, because all records have the same length.

- `Packed`

  MySQL strips spaces from the end of strings. The `Packed` value indicates the percentage of savings achieved by doing this.

- `Recordspace used`

  What percentage of the data file is used.

- `Empty space`

  What percentage of the data file is unused.

- `Blocks/Record`

  Average number of blocks per record (that is, how many links a fragmented record is composed of). This is always 1.0 for fixed-format tables. This value should stay as close to 1.0 as possible. If it gets too big, you can reorganize the table. See Section 5.7.3.10, “Table Optimization”.

- `Recordblocks`

  How many blocks (links) are used. For fixed format, this is the same as the number of records.

- `Deleteblocks`

How many blocks (links) are deleted.

- `Recorddata`

  How many bytes in the data file are used.

- `Deleted data`

  How many bytes in the data file are deleted (unused).

- `Lost space`

  If a record is updated to a shorter length, some space is lost. This is the sum of all such losses, in bytes.

- `Linkdata`

  When the dynamic table format is used, record fragments are linked with pointers (4 to 7 bytes each). `Linkdata` is the sum of the amount of storage used by all such pointers.

If a table has been compressed with **myisampack**, **myisamchk -d** prints additional information about each table column. See Section 8.2, "**myisampack**, the MySQL Compressed Read-only Table Generator", for an example of this information and a description of what it means.

5.8. MySQL Localization and International Usage

This section describes how to configure the server to use different character sets. It also discusses how to set the server's time zone and enable per-connection time zone support.

5.8.1. The Character Set Used for Data and Sorting

By default, MySQL uses the ISO-8859-1 (Latin1) character set with sorting according to Swedish/Finnish rules. These defaults are suitable for the United States and most of western Europe.

All MySQL binary distributions are compiled with `--with-extra-charsets=complex`. This adds code to all standard programs that enables them to handle `latin1` and all multi-byte character sets within the binary. Other character sets are loaded from a character-set definition file when needed.

The character set determines what characters are allowed in names. It also determines how strings are sorted by the `ORDER BY` and `GROUP BY` clauses of the `SELECT` statement.

You can change the character set with the `--default-character-set` option when you start the server. The character sets available depend on the `--with-charset=`*`charset`* and `--with-extra-charsets=` *`list-of-charsets`* `| complex | all | none` options to **configure**, and the character set configuration files listed in `SHAREDIR/charsets/Index`. See Section 2.8.2, "Typical **configure** Options".

As of MySQL 4.1.1, you can also change the character set collation with the `--default-collation` option when you start the server. The collation must be a legal collation for the default character set. (Use the `SHOW COLLATION` statement to determine which collations are available for each character set.) See Section 2.8.2, "Typical **configure** Options".

If you change the character set when running MySQL, that may also change the sort order. Con-

sequently, you must run **myisamchk -r -q --set-character-set=*charset*** on all tables, or your indexes may not be ordered correctly.

When a client connects to a MySQL server, the server indicates to the client what the server's default character set is. The client switches to use this character set for this connection.

You should use `mysql_real_escape_string()` when escaping strings for an SQL query. `mysql_real_escape_string()` is identical to the old `mysql_escape_string()` function, except that it takes the `MYSQL` connection handle as the first parameter so that the appropriate character set can be taken into account when escaping characters.

If the client is compiled with different paths than where the server is installed and the user who configured MySQL didn't include all character sets in the MySQL binary, you must tell the client where it can find the additional character sets it needs if the server runs with a different character set than the client.

You can do this by specifying a `--character-sets-dir` option to indicate the path to the directory in which the dynamic MySQL character sets are stored. For example, you can put the following in an option file:

```
[client]
character-sets-dir=/usr/local/mysql/share/mysql/charsets
```

You can force the client to use specific character set as follows:

```
[client]
default-character-set=charset
```

This is normally unnecessary, however.

5.8.1.1. Using the German Character Set

In MySQL 4.0, to get German sorting order, you should start **mysqld** with a `--default-character-set=latin1_de` option. This affects server behavior in several ways:

- When sorting and comparing strings, the following mapping is performed on the strings before doing the comparison:

  ```
  ä  ->  ae
  ö  ->  oe
  ü  ->  ue
  ß  ->  ss
  ```

- All accented characters are converted to their unaccented uppercase counterpart. All letters are converted to uppercase.

- When comparing strings with `LIKE`, the one-character to two-character mapping is not done. All letters are converted to uppercase. Accents are removed from all letters except `Ü`, `ü`, `Ö`, `ö`, `Ä`, and `ä`.

In MySQL 4.1 and up, character set and collation are specified separately. You should select the `latin1` character set and either the `latin1_german1_ci` or `latin1_german2_ci` collation. For example, to start the server with the `latin1_german1_ci` collation, use the `--character-set-server=latin1` and `--collation-server=latin1_german1_ci` options.

For information on the differences between these two collations, see Section 10.11.2, “West European Character Sets”.

5.8.2. Setting the Error Message Language

By default, **mysqld** produces error messages in English, but they can also be displayed in any of these other languages: Czech, Danish, Dutch, Estonian, French, German, Greek, Hungarian, Italian, Japanese, Korean, Norwegian, Norwegian-ny, Polish, Portuguese, Romanian, Russian, Slovak, Spanish, or Swedish.

To start **mysqld** with a particular language for error messages, use the `--language` or `-L` option. The option value can be a language name or the full path to the error message file. For example:

```
shell> mysqld --language=swedish
```

Or:

```
shell> mysqld --language=/usr/local/share/swedish
```

The language name should be specified in lowercase.

The language files are located (by default) in the `share/`*`LANGUAGE`* directory under the MySQL base directory.

To change the error message file, you should edit the `errmsg.txt` file, and then execute the following command to generate the `errmsg.sys` file:

```
shell> comp_err errmsg.txt errmsg.sys
```

If you upgrade to a newer version of MySQL, remember to repeat your changes with the new `errmsg.txt` file.

5.8.3. Adding a New Character Set

This section discusses the procedure for adding add another character set to MySQL. You must have a MySQL source distribution to use these instructions.

To choose the proper procedure, decide whether the character set is simple or complex:

- If the character set does not need to use special string collating routines for sorting and does not need multi-byte character support, it is simple.
- If it needs either of those features, it is complex.

For example, `latin1` and `danish` are simple character sets, whereas `big5` and `czech` are complex character sets.

In the following procedures, the name of your character set is represented by *`MYSET`*.

For a simple character set, do the following:

1. Add *`MYSET`* to the end of the `sql/share/charsets/Index` file. Assign a unique number to it.

2. Create the file `sql/share/charsets/MYSET.conf`. (You can use a copy of `sql/share/charsets/latin1.conf` as the basis for this file.)

   The syntax for the file is very simple:

   - Comments start with a '#' character and proceed to the end of the line.
   - Words are separated by arbitrary amounts of whitespace.
   - When defining the character set, every word must be a number in hexadecimal format.
   - The `ctype` array takes up the first 257 words. The `to_lower[]`, `to_upper[]` and `sort_order[]` arrays take up 256 words each after that.

   See Section 5.8.4, “The Character Definition Arrays”.

3. Add the character set name to the `CHARSETS_AVAILABLE` and `COMPILED_CHARSETS` lists in `configure.in`.

4. Reconfigure, recompile, and test.

For a complex character set, do the following:

1. Create the file `strings/ctype-MYSET.c` in the MySQL source distribution.

2. Add *MYSET* to the end of the `sql/share/charsets/Index` file. Assign a unique number to it.

3. Look at one of the existing `ctype-*.c` files (such as `strings/ctype-big5.c`) to see what needs to be defined. Note that the arrays in your file must have names like `ctype_MYSET`, `to_lower_MYSET`, and so on. These correspond to the arrays for a simple character set. See Section 5.8.4, “The Character Definition Arrays”.

4. Near the top of the file, place a special comment like this:

   ```
   /*
    * This comment is parsed by configure to create ctype.c,
    * so don't change it unless you know what you are doing.
    *
    * .configure. number_MYSET=MYNUMBER
    * .configure. strxfrm_multiply_MYSET=N
    * .configure. mbmaxlen_MYSET=N
    */
   ```

   The **configure** program uses this comment to include the character set into the MySQL library automatically.

   The `strxfrm_multiply` and `mbmaxlen` lines are explained in the following sections. You need include them only if you need the string collating functions or the multi-byte character set functions, respectively.

5. You should then create some of the following functions:

   - `my_strncoll_MYSET()`
   - `my_strcoll_MYSET()`

- `my_strxfrm_MYSET()`
- `my_like_range_MYSET()`

See Section 5.8.5, “String Collating Support”.

6. Add the character set name to the `CHARSETS_AVAILABLE` and `COMPILED_CHARSETS` lists in `configure.in`.
7. Reconfigure, recompile, and test.

The `sql/share/charsets/README` file includes additional instructions.

If you want to have the character set included in the MySQL distribution, mail a patch to the MySQL `internals` mailing list. See Section 1.4.1.1, “The MySQL Mailing Lists”.

5.8.4. The Character Definition Arrays

`to_lower[]` and `to_upper[]` are simple arrays that hold the lowercase and uppercase characters corresponding to each member of the character set. For example:

```
to_lower['A'] should contain 'a'
to_upper['a'] should contain 'A'
```

`sort_order[]` is a map indicating how characters should be ordered for comparison and sorting purposes. Quite often (but not for all character sets) this is the same as `to_upper[]`, which means that sorting is case-insensitive. MySQL sorts characters based on the values of `sort_order[]` elements. For more complicated sorting rules, see the discussion of string collating in Section 5.8.5, “String Collating Support”.

`ctype[]` is an array of bit values, with one element for one character. (Note that `to_lower[]`, `to_upper[]`, and `sort_order[]` are indexed by character value, but `ctype[]` is indexed by character value + 1. This is an old legacy convention to be able to handle `EOF`.)

You can find the following bitmask definitions in `m_ctype.h`:

```
#define _U      01      /* Uppercase */
#define _L      02      /* Lowercase */
#define _N      04      /* Numeral (digit) */
#define _S      010     /* Spacing character */
#define _P      020     /* Punctuation */
#define _C      040     /* Control character */
#define _B      0100    /* Blank */
#define _X      0200    /* heXadecimal digit */
```

The `ctype[]` entry for each character should be the union of the applicable bitmask values that describe the character. For example, `'A'` is an uppercase character (`_U`) as well as a hexadecimal digit (`_X`), so `ctype['A'+1]` should contain the value:

```
_U + _X = 01 + 0200 = 0201
```

5.8.5. String Collating Support

If the sorting rules for your language are too complex to be handled with the simple `sort_order[]`

table, you need to use the string collating functions.

The best documentation for this is the existing character sets. Look at the `big5`, `czech`, `gbk`, `sjis`, and `tis160` character sets for examples.

You must specify the `strxfrm_multiply_MYSET=N` value in the special comment at the top of the file. *N* should be set to the maximum ratio the strings may grow during `my_strxfrm_MYSET` (it must be a positive integer).

5.8.6. Multi-Byte Character Support

If you want to add support for a new character set that includes multi-byte characters, you need to use the multi-byte character functions.

The best documentation for this is the existing character sets. Look at the `euc_kr`, `gb2312`, `gbk`, `sjis`, and `ujis` character sets for examples. These are implemented in the `ctype-charset.c` files in the `strings` directory.

You must specify the `mbmaxlen_MYSET=N` value in the special comment at the top of the source file. *N* should be set to the size in bytes of the largest character in the set.

5.8.7. Problems With Character Sets

If you try to use a character set that is not compiled into your binary, you might run into the following problems:

- Your program has an incorrect path to where the character sets are stored. (Default `/usr/local/mysql/share/mysql/charsets`). This can be fixed by using the `--character-sets-dir` option when you run the program in question.
- The character set is a multi-byte character set that can't be loaded dynamically. In this case, you must recompile the program with support for the character set.
- The character set is a dynamic character set, but you don't have a configure file for it. In this case, you should install the configure file for the character set from a new MySQL distribution.
- If your `Index` file doesn't contain the name for the character set, your program displays the following error message:

  ```
  ERROR 1105: File '/usr/local/share/mysql/charsets/?.conf'
  not found (Errcode: 2)
  ```

  In this case, you should either get a new `Index` file or manually add the name of any missing character sets to the current file.

For `MyISAM` tables, you can check the character set name and number for a table with **myisamchk -dvv *tbl_name***.

5.8.8. MySQL Server Time Zone Support

Before MySQL 4.1.3, you can set the time zone for the server with the `--timezone=timezone_name` option to **mysqld_safe**. You can also set it by setting the `TZ` environment variable before you start **mysqld**.

The allowable values for `--timezone` or `TZ` are system-dependent. Consult your operating system

documentation to see what values are acceptable.

Beginning with MySQL 4.1.3, the server maintains several time zone settings:

- The system time zone. When the server starts, it attempts to determine the time zone of the host machine and uses it to set the `system_time_zone` system variable.
- The server's current time zone. The global `time_zone` system variable indicates the time zone the server currently is operating in. The initial value is `'SYSTEM'`, which indicates that the server time zone is the same as the system time zone. The initial value can be specified explicitly with the `--default-time-zone=timezone` option. If you have the `SUPER` privilege, you can set the global value at runtime with this statement:

  ```
  mysql> SET GLOBAL time_zone = timezone;
  ```

- Per-connection time zones. Each client that connects has its own time zone setting, given by the session `time_zone` variable. Initially this is the same as the global `time_zone` variable, but can be reset with this statement:

  ```
  mysql> SET time_zone = timezone;
  ```

The current values of the global and per-connection time zones can be retrieved like this:

```
mysql> SELECT @@global.time_zone, @@session.time_zone;
```

`timezone` values can be given as strings indicating an offset from UTC, such as `'+10:00'` or `'-6:00'`. If the time zone-related tables in the `mysql` database have been created and populated, you can also used named time zones, such as `'Europe/Helsinki'`, `'US/Eastern'`, or `'MET'`. The value `'SYSTEM'` indicates that the time zone should be the same as the system time zone. Time zone names are not case sensitive.

The MySQL installation procedure creates the time zone tables in the `mysql` database, but does not load them. You must do so manually. (If you are upgrading to MySQL 4.1.3 or later from an earlier version, you should create the tables by upgrading your `mysql` database. Use the instructions in Section 2.10.7, “Upgrading the Grant Tables”.)

If your system has its own zoneinfo database (the set of files describing time zones), you should use the **mysql_tzinfo_to_sql** program for filling the time zone tables. Examples of such systems are Linux, FreeBSD, Sun Solaris, and Mac OS X. One likely location for these files is the `/usr/share/zoneinfo` directory. If your system does not have a zoneinfo database, you can use the downloadable package described later in this section.

The **mysql_tzinfo_to_sql** program is used to load the time zone tables. On the command line, pass the zoneinfo directory pathname to **mysql_tzinfo_to_sql** and send the output into the **mysql** program. For example:

```
shell> mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

mysql_tzinfo_to_sql reads your system's time zone files and generates SQL statements from them. **mysql** processes those statements to load the time zone tables.

mysql_tzinfo_to_sql also can be used to load a single time zone file, and to generate leap second information.

To load a single time zone file *`tz_file`* that corresponds to a time zone name *`tz_name`*, invoke

mysql_tzinfo_to_sql like this:

```
shell> mysql_tzinfo_to_sql tz_file tz_name | mysql -u root mysql
```

If your time zone needs to account for leap seconds, initialize the leap second information like this, where *tz_file* is the name of your time zone file:

```
shell> mysql_tzinfo_to_sql --leap tz_file | mysql -u root mysql
```

If your system doesn't have a zoneinfo database (for example, Windows or HP-UX), you can use the package of pre-built time zone tables that is available for download at http://dev.mysql.com/downloads/timezones.html. This package contains `.frm`, `.MYD`, and `.MYI` files for the `MyISAM` time zone tables. These tables should belong to the `mysql` database, so you should place the files in the `mysql` subdirectory of your MySQL server's data directory. The server should be shut down while you do this.

Warning! Please don't use the downloadable package if your system has a zoneinfo database. Use the **mysql_tzinfo_to_sql** utility instead! Otherwise, you may cause a difference in datetime handling between MySQL and other applications on your system.

For information about time zone settings in replication setup please look into Section 6.7, “Replication Features and Known Problems”.

5.9. The MySQL Log Files

MySQL has several different log files that can help you find out what's going on inside **mysqld**:

Log File	Types of Information Logged to File
The error log	Logs problems encountered starting, running, or stopping **mysqld**.
The isam log	Logs all changes to the `ISAM` tables. Used only for debugging the isam code.
The query log	Logs established client connections and executed statements.
The update log	Logs statements that change data. This log is deprecated.
The binary log	Logs all statements that change data. Also used for replication.
The slow log	Logs all queries that took more than `long_query_time` seconds to execute or didn't use indexes.

By default, all logs are created in the **mysqld** data directory. You can force **mysqld** to close and reopen the log files (or in some cases switch to a new log) by flushing the logs. Log flushing occurs when you issue a `FLUSH LOGS` statement or execute **mysqladmin flush-logs** or **mysqladmin refresh**. See Section 13.5.5.2, “`FLUSH` Syntax”.

If you are using MySQL replication capabilities, slave replication servers maintain additional log files called relay logs. These are discussed in Chapter 6, *Replication in MySQL*.

5.9.1. The Error Log

The error log file contains information indicating when **mysqld** was started and stopped and also any critical errors that occur while the server is running.

If **mysqld** dies unexpectedly and **mysqld_safe** needs to restart it, **mysqld_safe** writes a `restarted mysqld` message to the error log. If **mysqld** notices a table that needs to be automatically checked or repaired, it writes a message to the error log.

On some operating systems, the error log contains a stack trace if **mysqld** dies. The trace can be used to determine where **mysqld** died. See Section E.1.4, “Using a Stack Trace”.

Beginning with MySQL 4.0.10, you can specify where **mysqld** stores the error log file with the `--log-error[=file_name]` option. If no `file_name` value is given, **mysqld** uses the name `host_name.err` and writes the file in the data directory. (Prior to MySQL 4.0.10, the Windows error log name is `mysql.err`.) If you execute `FLUSH LOGS`, the error log is renamed with a suffix of `-old` and **mysqld** creates a new empty log file.

In older MySQL versions on Unix, error log handling was done by **mysqld_safe** which redirected the error file to `host_name.err`. You could change this filename by specifying a `--err-log=file_name` option to **mysqld_safe**.

If you don't specify `--log-error`, or (on Windows) if you use the `--console` option, errors are written to stderr, the standard error output. Usually this is your terminal.

On Windows, error output is always written to the `.err` file if `--console` is not given.

5.9.2. The General Query Log

If you want to know what happens within **mysqld**, you should start it with the `--log[=file_name]` or `-l [file_name]` option. If no `file_name` value is given, the default name is `host_name.log` This logs all connections and statements to the log file. This log can be very useful when you suspect an error in a client and want to know exactly what the client sent to **mysqld**.

Older versions of the **mysql.server** script (from MySQL 3.23.4 to 3.23.8) pass a `--log` option to **safe_mysqld** to enable the general query log. If you need better performance when you start using MySQL in a production environment, you can remove the `--log` option from **mysql.server** or change it to `--log-bin`. See Section 5.9.4, “The Binary Log”.

mysqld writes statements to the query log in the order that it receives them. This may be different from the order in which they are executed. This is in contrast to the update log and the binary log, which are written after the query is executed, but before any locks are released. (The query log also contains all statements, whereas the update and binary logs do not contain statements that only select data.)

Server restarts and log flushing do not cause a new general query log file to be generated (although flushing closes and reopens it). On Unix, you can rename the file and create a new one by using the following commands:

```
shell> mv hostname.log hostname-old.log
shell> mysqladmin flush-logs
shell> cp hostname-old.log to-backup-directory
shell> rm hostname-old.log
```

On Windows, you cannot rename the log file while the server has it open. You must stop the server and rename the log. Then restart the server to create a new log.

5.9.3. The Update Log

Note: The update log has been deprecated and replaced by the binary log. See Section 5.9.4, “The Binary Log”. The binary log can do anything the old update log could do, and more. *The update log is unavailable as of MySQL 5.0.0*.

When started with the `--log-update[=file_name]` option, **mysqld** writes a log file containing all SQL statements that update data. If no `file_name` value is given, the default name is name of the host machine. If a filename is given, but it doesn't contain a leading path, the file is written in the data directory. If `file_name` doesn't have an extension, **mysqld** creates log files with names of the form

file_name.###, where *###* is a number that is incremented each time you start the server or flush the logs.

Note: For this naming scheme to work, you must not create your own files with the same names as those that might be used for the log file sequence.

Update logging is smart because it logs only statements that really update data. So, an `UPDATE` or a `DELETE` with a `WHERE` that finds no rows is not written to the log. It even skips `UPDATE` statements that set a column to its existing value.

The update logging is done immediately after a query completes but before any locks are released or any commit is done. This ensures that statements are logged in execution order.

If you want to update a database from update log files, you could do the following (assuming that your update logs have names of the form `file_name.###`):

```
shell> ls -1 -t -r file_name.[0-9]* | xargs cat | mysql
```

`ls` is used to sort the update log filenames into the right order.

This can be useful if you have to revert to backup files after a crash and you want to redo the updates that occurred between the time of the backup and the crash.

5.9.4. The Binary Log

The binary log has replaced the old update log, which is unavailable starting from MySQL 5.0. The binary log contains all information that is available in the update log in a more efficient format and in a manner that is transactionally safe.

The binary log contains all statements which updated data or (starting from MySQL 4.1.3) could potentially have updated it (for example, a `DELETE` which matched no rows).

The binary log also contains information about how long each statement took that updated the database. It doesn't contain statements that don't modify any data. If you want to log all statements (for example, to identify a problem query) you should use the general query log. See Section 5.9.2, “The General Query Log”.

The primary purpose of the binary log is to be able to update the database during a restore operation as fully as possible, because the binary log contains all updates done after a backup was made.

The binary log is also used on master replication servers as a record of the statements to be sent to slave servers. See Chapter 6, *Replication in MySQL*.

Running the server with the binary log enabled makes performance about 1% slower. However, the benefits of the binary log for restore operations and in allowing you to set up replication generally outweigh this minor performance decrement.

When started with the `--log-bin[=file_name]` option, **mysqld** writes a log file containing all SQL commands that update data. If no *file_name* value is given, the default name is the name of the host machine followed by `-bin`. If file name is given, but it doesn't contain a path, the file is written in the data directory. It is recommended to specify a filename, see Section 1.5.7.3, “Open Bugs and Design Deficiencies in MySQL” for the reason.

If you supply an extension in the log name (for example, `--log-bin=file_name.extension`), the extension is silently removed and ignored.

mysqld appends a numeric extension to the binary log name. The number is incremented each time you start the server or flush the logs. A new binary log also is created automatically when the current log's

size reaches `max_binlog_size`. A binary log may become larger than `max_binlog_size` if you are using large transactions: A transaction is written to the binary log in one piece, never split between binary logs.

To be able to know which different binary log files have been used, **mysqld** also creates a binary log index file that contains the name of all used binary log files. By default this has the same name as the binary log file, with the extension `'.index'`. You can change the name of the binary log index file with the `--log-bin-index[=file_name]` option. You should not manually edit this file while **mysqld** is running; doing so would confuse **mysqld**.

You can delete all binary log files with the `RESET MASTER` statement, or only some of them with `PURGE MASTER LOGS`. See Section 13.5.5.5, "`RESET` Syntax" and Section 13.6.1, "SQL Statements for Controlling Master Servers".

The binary log format has some known limitations which can affect recovery from backups, especially in old versions. These caveats which also affect replication are listed at Section 6.7, "Replication Features and Known Problems". One caveat which does not affect replication but only recovery with `mysqlbinlog`: before MySQL 4.1, **mysqlbinlog** could not prepare output suitable for **mysql** if the binary log contained interlaced statements originating from different clients that used temporary tables of the same name. This is fixed in MySQL 4.1. However, the problem still existed for `LOAD DATA INFILE` statements until it was fixed in MySQL 4.1.8.

You can use the following options to **mysqld** to affect what is logged to the binary log. See also the discussion that follows this option list.

- `--binlog-do-db=db_name`

  Tells the master that it should log updates to the binary log if the current database (that is, the one selected by `USE`) is *`db_name`*. All other databases that are not explicitly mentioned are ignored. If you use this, you should ensure that you only do updates in the current database.

  Observe that there is an exception to the `CREATE/ALTER/DROP DATABASE` statements, which use the database manipulated to decide if it should log the statement rather than the current database.

  An example of what does not work as you might expect: If the server is started with `binlog-do-db=sales`, and you do `USE prices; UPDATE sales.january SET amount=amount+1000;`, this statement does not get written into the binary log.

- `--binlog-ignore-db=db_name`

  Tells the master that updates where the current database (that is, the one selected by `USE`) is *`db_name`* should not be stored in the binary log. If you use this, you should ensure that you only do updates in the current database.

  An example of what does not work as you might expect: If the server is started with `binlog-ignore-db=sales`, and you do `USE prices; UPDATE sales.january SET amount=amount+1000;`, this statement is not written into the binary log.

  Similar to the case for `--binlog-do-db`, there is an exception to the `CREATE/ALTER/DROP DATABASE` statements, which use the database manipulated to decide if it should log the statement rather than the current database.

To log or ignore multiple databases, specify the appropriate option multiple times, once for each database.

The rules for logging or ignoring updates to the binary log are evaluated according to the following rules. Observe that there is an exception for `CREATE/ALTER/DROP DATABASE` statements. In those cases, the database being *created/altered/dropped* replace the current database in the rules below.

1. Are there `binlog-do-db` or `binlog-ignore-db` rules?
   - No: Write the statement to the binary log and exit.
   - Yes: Go to the next step.
2. There are some rules (`binlog-do-db` or `binlog-ignore-db` or both). Is there a current database (has any database been selected by `USE`?)?
   - No: Do *not* write the statement, and exit.
   - Yes: Go to the next step.
3. There is a current database. Are there some `binlog-do-db` rules?
   - Yes: Does the current database match any of the `binlog-do-db` rules?
     - Yes: Write the statement and exit.
     - No: Do *not* write the statement, and exit.
   - No: Go to the next step.
4. There are some `binlog-ignore-db` rules. Does the current database match any of the `binlog-ignore-db` rules?
   - Yes: Do not write the statement, and exit.
   - No: Write the query and exit.

For example, a slave running with only `binlog-do-db=sales` does not write to the binary log any statement whose current database is different from `sales` (in other words, `binlog-do-db` can sometimes mean ``ignore other databases").

If you are using replication, you should not delete old binary log files until you are sure that no slave still needs to use them. One way to do this is to do **mysqladmin flush-logs** once a day and then remove any logs that are more than three days old. You can remove them manually, or preferably using `PURGE MASTER LOGS` (see Section 13.6.1, “SQL Statements for Controlling Master Servers”), which also safely updates the binary log index file for you (and which can take a date argument since MySQL 4.1)

A client with the `SUPER` privilege can disable binary logging of its own statements by using a `SET SQL_LOG_BIN=0` statement. See Section 13.5.3, “`SET` Syntax”.

You can examine the binary log file with the **mysqlbinlog** utility. This can be useful when you want to reprocess statements in the log. For example, you can update a MySQL server from the binary log as follows:

```
shell> mysqlbinlog log-file | mysql -h server_name
```

See Section 8.5, “The **mysqlbinlog** Binary Log Utility” for more information on the **mysqlbinlog** utility and how to use it.

If you are using transactions, you must use the MySQL binary log for backups instead of the old update log.

The binary logging is done immediately after a query completes but before any locks are released or any commit is done. This ensures that the log is logged in the execution order.

Updates to non-transactional tables are stored in the binary log immediately after execution. For transactional tables such as BDB or InnoDB tables, all updates (UPDATE, DELETE, or INSERT) that change tables are cached until a COMMIT statement is received by the server. At that point, **mysqld** writes the whole transaction to the binary log before the COMMIT is executed. When the thread that handles the transaction starts, it allocates a buffer of binlog_cache_size to buffer queries. If a statement is bigger than this, the thread opens a temporary file to store the transaction. The temporary file is deleted when the thread ends.

The Binlog_cache_use status variable shows the number of transactions that used this buffer (and possibly a temporary file) for storing statements. The Binlog_cache_disk_use status variable shows how many of those transactions actually did have to use a temporary file. These two variables can be used for tuning binlog_cache_size to a large enough value that avoids the use of temporary files.

The max_binlog_cache_size (default 4GB) can be used to restrict the total size used to cache a multiple-statement transaction. If a transaction is larger than this, it fails and rolls back.

If you are using the update log or binary log, concurrent inserts are converted to normal inserts when using CREATE ... SELECT or INSERT ... SELECT. This is to ensure that you can re-create an exact copy of your tables by applying the log on a backup.

The binary log format is different in versions 3.23, 4.0, and 5.0.0. Those format changes were required to implement enhancements to replication. MySQL 4.1 has the same binary log format as 4.0. See Section 6.5, “Replication Compatibility Between MySQL Versions”.

By default, the binary log is not synchronized to disk at each write. So if the operating system or machine (not only the MySQL server) crashes there is a chance that the last statements of the binary log are lost. To prevent this, you can make the binary log be synchronized to disk after every Nth binary log write, with the sync_binlog global variable (1 being the safest value, but also the slowest). See Section 5.2.3, “Server System Variables”. Even with this set to 1, there is still the chance of an inconsistency between the tables content and the binary log content in case of crash. For example, if using InnoDB tables, and the MySQL server processes a COMMIT statement, it writes the whole transaction to the binary log and then commits this transaction into InnoDB. If it crashes between those two operations, at restart the transaction is rolled back by InnoDB but still exist in the binary log. This problem can be solved with the --innodb-safe-binlog option (available starting from MySQL 4.1.3), which adds consistency between the content of InnoDB tables and the binary log. For this option to really bring safety to you, the MySQL server should also be configured to synchronize to disk, at every transaction, the binary log (sync_binlog=1) and (which is true by default) the InnoDB logs. The effect of this option is that at restart after a crash, after doing a rollback of transactions, the MySQL server cuts rolled back InnoDB transactions from the binary log. This ensures that the binary log reflects the exact data of InnoDB tables, and so, that the slave remains in sync with the master (not receiving a statement which has been rolled back). Note that --innodb-safe-binlog can be used even if the MySQL server updates other storage engines than InnoDB. Only statements/transactions affecting InnoDB tables are subject to being removed from the binary log at InnoDB's crash recovery. If at crash recovery the MySQL server discovers that the binary log is shorter than it should have been (that is, it lacks at least one successfully committed InnoDB transaction), which should not happen if sync_binlog=1 and the disk/filesystem do an actual sync when they are requested to (some don't), it prints an error message ("The binary log <name> is shorter than its expected size"). In this case, this binary log is not correct, replication should be restarted from a fresh master's data snapshot.

Before MySQL 4.1.9, a write to a binary log file or binary log index file that failed due to a full disk or an exceeded quota resulted in corruption of the file. Starting from MySQL 4.1.9, writes to the binary log file and binary log index file are handled the same way as writes to MyISAM tables. See Section A.4.3, “How MySQL Handles a Full Disk”.

5.9.5. The Slow Query Log

When started with the `--log-slow-queries[=file_name]` option, **mysqld** writes a log file containing all SQL statements that took more than `long_query_time` seconds to execute. The time to acquire the initial table locks are not counted as execution time.

If no *`file_name`* value is given, the default is the name of the host machine with a suffix of `-slow.log`. If a filename is given, but doesn't contain a path, the file is written in the data directory.

A statement is logged to the slow query log after it has been executed and after all locks have been released. Log order may be different from execution order.

The slow query log can be used to find queries that take a long time to execute and are therefore candidates for optimization. However, examining a long slow query log can become a difficult task. To make this easier, you can pipe the slow query log through the **mysqldumpslow** command to get a summary of the queries that appear in the log.

Before MySQL 4.1, if you also use `--log-long-format` when logging slow queries, then queries that are not using indexes are logged as well. queries that are not using indexes also are logged to the slow query log. `--log-long-format` is deprecated as of MySQL version 4.1, when `--log-short-format` was introduced. (Long log format is the default setting since version 4.1.) Also note that starting with MySQL 4.1, the `--log-queries-not-using-indexes` option is available for the purpose of logging queries that do not use indexes to the slow query log. See Section 5.2.1, "**mysqld** Command-Line Options".

Queries handled by the query cache are not added to the slow query log, nor are queries that would not benefit from the presence of an index because the table has zero rows or one row.

5.9.6. Log File Maintenance

The MySQL Server can create a number of different log files that make it easy to see what is going on. See Section 5.9, "The MySQL Log Files". However, you must clean up these files regularly to ensure that the logs don't take up too much disk space.

When using MySQL with logging enabled, you may want to back up and remove old log files from time to time and tell MySQL to start logging to new files. See Section 5.7.1, "Database Backups".

On a Linux (Red Hat) installation, you can use the `mysql-log-rotate` script for this. If you installed MySQL from an RPM distribution, the script should have been installed automatically. You should be careful with this script if you are using the binary log for replication! (You should not remove binary logs until you are certain that their contents have been processed by all slaves.)

On other systems, you must install a short script yourself that you start from **cron** to handle log files.

You can force MySQL to start using new log files by using **mysqladmin flush-logs** or by using the SQL statement `FLUSH LOGS`. If you are using MySQL 3.21, you must use **mysqladmin refresh**.

A log flushing operation does the following:

- If standard logging (`--log`) or slow query logging (`--log-slow-queries`) is used, closes and reopens the log file (`mysql.log` and `` `hostname`-slow.log `` as default).
- If update logging (`--log-update`) or binary logging (`--log-bin`) is used, closes the log and opens a new log file with a higher sequence number.

If you are using only an update log, you only have to rename the log file and then flush the logs before making a backup. For example, you can do something like this:

```
shell> cd mysql-data-directory
shell> mv mysql.log mysql.old
shell> mysqladmin flush-logs
```

Then make a backup and remove `mysql.old`.

5.10. Running Multiple MySQL Servers on the Same Machine

In some cases, you might want to run multiple **mysqld** servers on the same machine. You might want to test a new MySQL release while leaving your existing production setup undisturbed. Or you may want to give different users access to different **mysqld** servers that they manage themselves. (For example, you might be an Internet Service Provider that wants to provide independent MySQL installations for different customers.)

To run multiple servers on a single machine, each server must have unique values for several operating parameters. These can be set on the command line or in option files. See Section 4.3, “Specifying Program Options”.

At least the following options must be different for each server:

- `--port=port_num`

  `--port` controls the port number for TCP/IP connections.

- `--socket=path`

  `--socket` controls the Unix socket file path on Unix and the name of the named pipe on Windows. On Windows, it's necessary to specify distinct pipe names only for those servers that support named pipe connections.

- `--shared-memory-base-name=name`

  This option currently is used only on Windows. It designates the shared memory name used by a Windows server to allow clients to connect via shared memory. This option is new in MySQL 4.1.

- `--pid-file=path`

  This option is used only on Unix. It indicates the name of the file in which the server writes its process ID.

If you use the following log file options, they must be different for each server:

- `--log=path`
- `--log-bin=path`
- `--log-update=path`
- `--log-error=path`
- `--log-isam=path`
- `--bdb-logdir=path`

Log file options are described in Section 5.9.6, “Log File Maintenance”.

If you want more performance, you can also specify the following options differently for each server, to spread the load between several physical disks:

- `--tmpdir=path`
- `--bdb-tmpdir=path`

Having different temporary directories is also recommended, to make it easier to determine which MySQL server created any given temporary file.

Generally, each server should also use a different data directory, which is specified using the `--datadir=path` option.

Warning: Normally you should never have two servers that update data in the same databases! This may lead to unpleasant surprises if your operating system doesn't support fault-free system locking! If (despite this warning) you run multiple servers using the same data directory and they have logging enabled, you must use the appropriate options to specify log filenames that are unique to each server. Otherwise, the servers try to log to the same files. Please note that this kind of setup only works with `ISAM`, `MyISAM` and `MERGE` tables, and not with any of the other storage engines.

The warning against sharing a data directory among servers also applies in an NFS environment. Allowing multiple MySQL servers to access a common data directory over NFS is a *bad idea*!

- The primary problem is that NFS is the speed bottleneck. It is not meant for such use.
- Another risk with NFS is that you has to come up with a way to make sure that two or more servers do not interfere with each other. Usually NFS file locking is handled by the `lockd` daemon, but at the moment there is no platform that performs locking 100% reliably in every situation.

Make it easy for yourself: Forget about sharing a data directory among servers over NFS. A better solution is to have one computer that contains several CPUs and use an operating system that handles threads efficiently.

If you have multiple MySQL installations in different locations, normally you can specify the base installation directory for each server with the `--basedir=path` option to cause each server to use a different data directory, log files, and PID file. (The defaults for all these values are determined relative to the base directory). In that case, the only other options you need to specify are the `--socket` and `--port` options. For example, suppose that you install different versions of MySQL using `tar` file binary distributions. These install in different locations, so you can start the server for each installation using the command **bin/mysqld_safe** under its corresponding base directory. **mysqld_safe** determines the proper `--basedir` option to pass to **mysqld**, and you need specify only the `--socket` and `--port` options to **mysqld_safe**. (For versions of MySQL older than 4.0, use **safe_mysqld** rather than **mysqld_safe**.)

As discussed in the following sections, it is possible to start additional servers by setting environment variables or by specifying appropriate command-line options. However, if you need to run multiple servers on a more permanent basis, it is more convenient to use option files to specify for each server those option values that must be unique to it.

5.10.1. Running Multiple Servers on Windows

You can run multiple servers on Windows by starting them manually from the command line, each with

appropriate operating parameters. On Windows NT-based systems, you also have the option of installing several servers as Windows services and running them that way. General instructions for running MySQL servers from the command line or as services are given in Section 2.3, “Installing MySQL on Windows”. This section describes how to make sure that you start each server with different values for those startup options that must be unique per server, such as the data directory. These options are described in Section 5.10, “Running Multiple MySQL Servers on the Same Machine”.

5.10.1.1. Starting Multiple Windows Servers at the Command Line

To start multiple servers manually from the command line, you can specify the appropriate options on the command line or in an option file. It's more convenient to place the options in an option file, but it's necessary to make sure that each server gets its own set of options. To do this, create an option file for each server and tell the server the filename with a `--defaults-file` option when you run it.

Suppose that you want to run **mysqld** on port 3307 with a data directory of `C:\mydata1`, and **mysqld-max** on port 3308 with a data directory of `C:\mydata2`. (To do this, make sure that before you start the servers, each data directory exists and has its own copy of the `mysql` database that contains the grant tables.)

Then create two option files. For example, create one file named `C:\my-opts1.cnf` that looks like this:

```
[mysqld]
datadir = C:/mydata1
port = 3307
```

Create a second file named `C:\my-opts2.cnf` that looks like this:

```
[mysqld]
datadir = C:/mydata2
port = 3308
```

Then start each server with its own option file:

```
C:\> C:\mysql\bin\mysqld --defaults-file=C:\my-opts1.cnf
C:\> C:\mysql\bin\mysqld-max --defaults-file=C:\my-opts2.cnf
```

On NT, each server starts in the foreground (no new prompt appears until the server exits later); you'll need to issue those two commands in separate console windows.

To shut down the servers, you must connect to the appropriate port number:

```
C:\> C:\mysql\bin\mysqladmin --port=3307 shutdown
C:\> C:\mysql\bin\mysqladmin --port=3308 shutdown
```

Servers configured as just described allow clients to connect over TCP/IP. If your version of Windows supports named pipes and you also want to allow named pipe connections, use the **mysqld-nt** or **mysqld-max-nt** servers and specify options that enable the named pipe and specify its name. Each server that supports named pipe connections must use a unique pipe name. For example, the `C:\my-opts1.cnf` file might be written like this:

```
[mysqld]
datadir = C:/mydata1
port = 3307
enable-named-pipe
socket = mypipe1
```

Then start the server this way:

```
C:\> C:\mysql\bin\mysqld-nt --defaults-file=C:\my-opts1.cnf
```

Modify `C:\my-opts2.cnf` similarly for use by the second server.

5.10.1.2. Starting Multiple Windows Servers as Services

On NT-based systems, a MySQL server can be run as a Windows service. The procedures for installing, controlling, and removing a single MySQL service are described in Section 2.3.12, “Starting MySQL as a Windows Service”.

As of MySQL 4.0.2, you can install multiple servers as services. In this case, you must make sure that each server uses a different service name in addition to all the other parameters that must be unique per server.

For the following instructions, assume that you want to run the **mysqld-nt** server from two different versions of MySQL that are installed at `C:\mysql-4.0.8` and `C:\mysql-4.0.17`, respectively. (This might be the case if you're running 4.0.8 as your production server, but want to test 4.0.17 before upgrading to it.)

The following principles apply when installing a MySQL service with the `--install` or `--install-manual` option:

- If you specify no service name, the server uses the default service name of `MySQL` and the server reads options from the `[mysqld]` group in the standard option files.
- If you specify a service name after the `--install` option, the server ignores the `[mysqld]` option group and instead reads options from the group that has the same name as the service. The server reads options from the standard option files.
- If you specify a `--defaults-file` option after the service name, the server ignores the standard option files and reads options only from the `[mysqld]` group of the named file.

Note: Before MySQL 4.0.17, only a server installed using the default service name (`MySQL`) or one installed explicitly with a service name of **mysqld** read the `[mysqld]` group in the standard option files. As of 4.0.17, all servers read the `[mysqld]` group if they read the standard option files, even if they are installed using another service name. This allows you to use the `[mysqld]` group for options that should be used by all MySQL services, and an option group named after each service for use by the server installed with that service name.

Based on the preceding information, you have several ways to set up multiple services. The following instructions describe some examples. Before trying any of them, be sure that you shut down and remove any existing MySQL services first.

- **Approach 1:** Specify the options for all services in one of the standard option files. To do this, use a different service name for each server. Suppose that you want to run the 4.0.8 **mysqld-nt** using the service name of `mysqld1` and the 4.0.17 **mysqld-nt** using the service name `mysqld2`. In this case, you can use the `[mysqld1]` group for 4.0.8 and the `[mysqld2]` group for 4.0.17. For example, you can set up `C:\my.cnf` like this:

  ```
  # options for mysqld1 service
  [mysqld1]
  ```

```
basedir = C:/mysql-4.0.8
port = 3307
enable-named-pipe
socket = mypipe1

# options for mysqld2 service
[mysqld2]
basedir = C:/mysql-4.0.17
port = 3308
enable-named-pipe
socket = mypipe2
```

Install the services as follows, using the full server pathnames to ensure that Windows registers the correct executable program for each service:

```
C:\> C:\mysql-4.0.8\bin\mysqld-nt --install mysqld1
C:\> C:\mysql-4.0.17\bin\mysqld-nt --install mysqld2
```

To start the services, use the services manager, or use **NET START** with the appropriate service names:

```
C:\> NET START mysqld1
C:\> NET START mysqld2
```

To stop the services, use the services manager, or use **NET STOP** with the appropriate service names:

```
C:\> NET STOP mysqld1
C:\> NET STOP mysqld2
```

- **Approach 2:** Specify options for each server in separate files and use `--defaults-file` when you install the services to tell each server what file to use. In this case, each file should list options using a `[mysqld]` group.

  With this approach, to specify options for the 4.0.8 **mysqld-nt**, create a file `C:\my-opts1.cnf` that looks like this:

  ```
  [mysqld]
  basedir = C:/mysql-4.0.8
  port = 3307
  enable-named-pipe
  socket = mypipe1
  ```

  For the 4.0.17 **mysqld-nt**, create a file `C:\my-opts2.cnf` that looks like this:

  ```
  [mysqld]
  basedir = C:/mysql-4.0.17
  port = 3308
  enable-named-pipe
  socket = mypipe2
  ```

  Install the services as follows (enter each command on a single line):

  ```
  C:\> C:\mysql-4.0.8\bin\mysqld-nt --install mysqld1
              --defaults-file=C:\my-opts1.cnf
  ```

```
C:\> C:\mysql-4.0.17\bin\mysqld-nt --install mysqld2
           --defaults-file=C:\my-opts2.cnf
```

To use a `--defaults-file` option when you install a MySQL server as a service, you must precede the option with the service name.

After installing the services, start and stop them the same way as in the preceding example.

To remove multiple services, use **mysqld --remove** for each one, specifying a service name following the `--remove` option. If the service name is the default (`MySQL`), you can omit it.

5.10.2. Running Multiple Servers on Unix

The easiest way is to run multiple servers on Unix is to compile them with different TCP/IP ports and Unix socket files so that each one is listening on different network interfaces. Also, by compiling in different base directories for each installation, that automatically results in different compiled-in data directory, log file, and PID file locations for each of your servers.

Assume that an existing server is configured for the default TCP/IP port number (3306) and Unix socket file (`/tmp/mysql.sock`). To configure a new server to have different operating parameters, use a **configure** command something like this:

```
shell> ./configure --with-tcp-port=port_number \
             --with-unix-socket-path=file_name \
             --prefix=/usr/local/mysql-4.0.17
```

Here, *`port_number`* and *`file_name`* must be different from the default TCP/IP port number and Unix socket file pathname, and the `--prefix` value should specify an installation directory different than the one under which the existing MySQL installation is located.

If you have a MySQL server listening on a given port number, you can use the following command to find out what operating parameters it is using for several important configurable variables, including the base directory and Unix socket filename:

```
shell> mysqladmin --host=host_name --port=port_number variables
```

With the information displayed by that command, you can tell what option values *not* to use when configuring an additional server.

Note that if you specify `localhost` as a hostname, **mysqladmin** defaults to using a Unix socket file connection rather than TCP/IP. In MySQL 4.1, you can explicitly specify the connection protocol to use by using the `--protocol={TCP | SOCKET | PIPE | MEMORY}` option.

You don't have to compile a new MySQL server just to start with a different Unix socket file and TCP/IP port number. It is also possible to specify those values at runtime. One way to do so is by using command-line options:

```
shell> mysqld_safe --socket=file_name --port=port_number
```

To start a second server, provide different `--socket` and `--port` option values, and pass a `--datadir=path` option to **mysqld_safe** so that the server uses a different data directory.

Another way to achieve a similar effect is to use environment variables to set the Unix socket filename

and TCP/IP port number:

```
shell> MYSQL_UNIX_PORT=/tmp/mysqld-new.sock
shell> MYSQL_TCP_PORT=3307
shell> export MYSQL_UNIX_PORT MYSQL_TCP_PORT
shell> mysql_install_db --user=mysql
shell> mysqld_safe --datadir=/path/to/datadir &
```

This is a quick way of starting a second server to use for testing. The nice thing about this method is that the environment variable settings apply to any client programs that you invoke from the same shell. Thus, connections for those clients are automatically directed to the second server!

Appendix F, *Environment Variables* includes a list of other environment variables you can use to affect **mysqld**.

For automatic server execution, your startup script that is executed at boot time should execute the following command once for each server with an appropriate option file path for each command:

```
mysqld_safe --defaults-file=path
```

Each option file should contain option values specific to a given server.

On Unix, the **mysqld_multi** script is another way to start multiple servers. See Section 5.1.5, “The **mysqld_multi** Program for Managing Multiple MySQL Servers”.

5.10.3. Using Client Programs in a Multiple-Server Environment

When you want to connect with a client program to a MySQL server that is listening to different network interfaces than those compiled into your client, you can use one of the following methods:

- Start the client with `--host=host_name --port=port_number` to connect via TCP/IP to a remote server, with `--host=127.0.0.1 --port=port_number` to connect via TCP/IP to a local server, or with `--host=localhost --socket=file_name` to connect to a local server via a Unix socket file or a Windows named pipe.
- As of MySQL 4.1, start the client with `--protocol=tcp` to connect via TCP/IP, `--protocol=socket` to connect via a Unix socket file, `--protocol=pipe` to connect via a named pipe, or `--protocol=memory` to connect via shared memory. For TCP/IP connections, you may also need to specify `--host` and `--port` options. For the other types of connections, you may need to specify a `--socket` option to specify a Unix socket file or named pipe name, or a `--shared-memory-base-name` option to specify the shared memory name. Shared memory connections are supported only on Windows.
- On Unix, set the `MYSQL_UNIX_PORT` and `MYSQL_TCP_PORT` environment variables to point to the Unix socket file and TCP/IP port number before you start your clients. If you normally use a specific socket file or port number, you can place commands to set these environment variables in your `.login` file so that they apply each time you log in. See Appendix F, *Environment Variables*.
- Specify the default Unix socket file and TCP/IP port number in the `[client]` group of an option file. For example, you can use `C:\my.cnf` on Windows, or the `.my.cnf` file in your home directory on Unix. See Section 4.3.2, “Using Option Files”.
- In a C program, you can specify the socket file or port number arguments in the `mysql_real_connect()` call. You can also have the program read option files by calling

`mysql_options()`. See Section 22.2.3, “C API Function Descriptions”.

- If you are using the Perl `DBD::mysql` module, you can read options from MySQL option files. For example:

  ```
  $dsn = "DBI:mysql:test;mysql_read_default_group=client;"
          . "mysql_read_default_file=/usr/local/mysql/data/my.cnf";
  $dbh = DBI->connect($dsn, $user, $password);
  ```

  See Section 22.4, “MySQL Perl API”.

  Other programming interfaces may provide similar capabilities for reading option files.

5.11. The MySQL Query Cache

From version 4.0.1 on, MySQL Server features a query cache. When in use, the query cache stores the text of a `SELECT` query together with the corresponding result that was sent to the client. If the identical query is received later, the server retrieves the results from the query cache rather than parsing and executing the query again.

The query cache is extremely useful in an environment where (some) tables don't change very often and you have a lot of identical queries. This is a typical situation for many Web servers that generate a lot of dynamic pages based on database content.

Note: The query cache does not return stale data. When tables are modified, any relevant entries in the query cache are flushed.

Note: The query cache does not work in an environment where you have many **mysqld** servers updating the same `MyISAM` tables.

Some performance data for the query cache follow. These results were generated by running the MySQL benchmark suite on a Linux Alpha 2 x 500MHz system with 2GB RAM and a 64MB query cache.

- If all the queries you're performing are simple (such as selecting a row from a table with one row), but still differ so that the queries cannot be cached, the overhead for having the query cache active is 13%. This could be regarded as the worst case scenario. In real life, queries tend to be much more complicated, so the overhead normally is significantly lower.
- Searches for a single row in a single-row table are 238% faster with the query cache than without it. This can be regarded as close to the minimum speedup to be expected for a query that is cached.

To disable the query cache at server startup, set the `query_cache_size` system variable to 0. By disabling the query cache code, there is no noticeable overhead. Query cache capabilities can be excluded from the server entirely by using the `--without-query-cache` option to **configure** when compiling MySQL.

5.11.1. How the Query Cache Operates

This section describes how the query cache works when it is operational. Section 5.11.3, “Query Cache Configuration” describes how to control whether or not it is operational.

Queries are compared before parsing, so the following two queries are regarded as different by the query cache:

```
SELECT * FROM tbl_name
Select * from tbl_name
```

Queries must be exactly the same (byte for byte) to be seen as identical. In addition, query strings that are identical may be treated as different for other reasons. Queries that use different databases, different protocol versions, or different default character sets are considered different queries and are cached separately.

If a query result is returned from query cache, the server increments the `Qcache_hits` status variable, not `Com_select`. See Section 5.11.4, “Query Cache Status and Maintenance”.

If a table changes, then all cached queries that use the table become invalid and are removed from the cache. This includes queries that use `MERGE` tables that map to the changed table. A table can be changed by many types of statements, such as `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE`, `ALTER TABLE`, `DROP TABLE`, or `DROP DATABASE`.

Transactional `InnoDB` tables that have been changed are invalidated when a `COMMIT` is performed.

In MySQL 4.0, the query cache is disabled within transactions (it does not return results). Beginning with MySQL 4.1.1, the query cache also works within transactions when using `InnoDB` tables (it uses the table version number to detect whether or not its contents are still current).

Before MySQL 5.0, a query that begins with a leading comment might be cached, but could not be fetched from the cache. This problem is fixed in MySQL 5.0.

The query cache works for `SELECT SQL_CALC_FOUND_ROWS ...` and `SELECT FOUND_ROWS()` type queries. `FOUND_ROWS()` returns the correct value even if the preceding query was fetched from the cache because the number of found rows is also stored in the cache.

A query cannot be cached if it contains any of the following functions:

`BENCHMARK()`	`CONNECTION_ID()`	`CURDATE()`
`CURRENT_DATE()`	`CURRENT_TIME()`	`CURRENT_TIMESTAMP()`
`CURTIME()`	`DATABASE()`	`ENCRYPT()` with one parameter
`FOUND_ROWS()`	`GET_LOCK()`	`LAST_INSERT_ID()`
`LOAD_FILE()`	`MASTER_POS_WAIT()`	`NOW()`
`RAND()`	`RELEASE_LOCK()`	`SYSDATE()`
`UNIX_TIMESTAMP()` with no parameters	`USER()`	

A query also is not cached under these conditions:

- It contains user-defined functions (UDFs).
- It contains user variables.
- It refers to the tables in the `mysql` system database.
- It is of any of the following forms:

  ```
  SELECT ... IN SHARE MODE
  SELECT ... INTO OUTFILE ...
  SELECT ... INTO DUMPFILE ...
  SELECT * FROM ... WHERE autoincrement_col IS NULL
  ```

The last form is not cached because it is used as the ODBC workaround for obtaining the last insert ID value. See Section 23.1.14.1, “How to Get the Value of an `AUTO_INCREMENT` Column in ODBC”.

- It uses `TEMPORARY` tables.
- It does not use any tables.
- The user has a column-level privilege for any of the involved tables.
- Before a query is fetched from the query cache, MySQL checks that the user has `SELECT` privilege for all the involved databases and tables. If this is not the case, the cached result is not used.

5.11.2. Query Cache `SELECT` Options

There are two query cache-related options that may be specified in a `SELECT` statement:

- `SQL_CACHE`

  The query result is cached if the value of the `query_cache_type` system variable is `ON` or `DEMAND`.

- `SQL_NO_CACHE`

  The query result is not cached.

Examples:

```
SELECT SQL_CACHE id, name FROM customer;
SELECT SQL_NO_CACHE id, name FROM customer;
```

5.11.3. Query Cache Configuration

The `have_query_cache` server system variable indicates whether the query cache is available:

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | YES   |
+------------------+-------+
```

Several other system variables control query cache operation. These can be set in an option file or on the command line when starting **mysqld**. The query cache-related system variables all have names that begin with `query_cache_`. They are described briefly in Section 5.2.3, “Server System Variables”, with additional configuration information given here.

To set the size of the query cache, set the `query_cache_size` system variable. Setting it to 0 disables the query cache. The default cache size is 0; that is, the query cache is disabled.

If the query cache is enabled, the `query_cache_type` variable influences how it works. This vari-

able can be set to the following values:

- A value of `0` or `OFF` prevents caching or retrieval of cached results.
- A value of `1` or `ON` allows caching except of those statements that begin with `SELECT SQL_NO_CACHE`.
- A value of `2` or `DEMAND` causes caching of only those statements that begin with `SELECT SQL_CACHE`.

Setting the `GLOBAL` value of `query_cache_type` determines query cache behavior for all clients that connect after the change is made. Individual clients can control cache behavior for their own connection by setting the `SESSION` value of `query_cache_type`. For example, a client can disable use of the query cache for its own queries like this:

```
mysql> SET SESSION query_cache_type = OFF;
```

To control the maximum size of individual query results that can be cached, set the `query_cache_limit` variable. The default value is 1MB.

The result of a query (the data sent to the client) is stored in the query cache during result retrieval. Therefore the data usually is not handled in one big chunk. The query cache allocates blocks for storing this data on demand, so when one block is filled, a new block is allocated. Because memory allocation operation is costly (timewise), the query cache allocates blocks with a minimum size given by the `query_cache_min_res_unit` system variable. When a query is executed, the last result block is trimmed to the actual data size so that unused memory is freed. Depending on the types of queries your server executes, you might find it helpful to tune the value of `query_cache_min_res_unit`:

- The default value of `query_cache_min_res_unit` is 4KB. This should be adequate for most cases.
- If you have a lot of queries with small results, the default block size may lead to memory fragmentation, as indicated by a large number of free blocks. Fragmentation can force the query cache to prune (delete) queries from the cache due to lack of memory. In this case, you should decrease the value of `query_cache_min_res_unit`. The number of free blocks and queries removed due to pruning are given by the values of the `Qcache_free_blocks` and `Qcache_lowmem_prunes` status variables.
- If most of your queries have large results (check the `Qcache_total_blocks` and `Qcache_queries_in_cache` status variables), you can increase performance by increasing `query_cache_min_res_unit`. However, be careful to not make it too large (see the previous item).

`query_cache_min_res_unit` is present from MySQL 4.1.

5.11.4. Query Cache Status and Maintenance

You can check whether the query cache is present in your MySQL server using the following statement:

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | YES   |
+------------------+-------+
```

You can defragment the query cache to better utilize its memory with the `FLUSH QUERY CACHE` statement. The statement does not remove any queries from the cache.

The `RESET QUERY CACHE` statement removes all query results from the query cache. The `FLUSH TABLES` statement also does this.

To monitor query cache performance, use `SHOW STATUS` to view the cache status variables:

```
mysql> SHOW STATUS LIKE 'Qcache%';
+-------------------------+--------+
| Variable_name           | Value  |
+-------------------------+--------+
| Qcache_free_blocks      | 36     |
| Qcache_free_memory      | 138488 |
| Qcache_hits             | 79570  |
| Qcache_inserts          | 27087  |
| Qcache_lowmem_prunes    | 3114   |
| Qcache_not_cached       | 22989  |
| Qcache_queries_in_cache | 415    |
| Qcache_total_blocks     | 912    |
+-------------------------+--------+
```

Descriptions of each of these variables are given in Section 5.2.4, “Server Status Variables”. Some uses for them are described here.

The total number of `SELECT` queries is equal to:

```
  Com_select
+ Qcache_hits
+ queries with errors found by parser
```

The `Com_select` value is equal to:

```
  Qcache_inserts
+ Qcache_not_cached
+ queries with errors found during columns/rights check
```

The query cache uses variable-length blocks, so `Qcache_total_blocks` and `Qcache_free_blocks` may indicate query cache memory fragmentation. After `FLUSH QUERY CACHE`, only a single free block remains.

Every cached query requires a minimum of two blocks (one for the query text and one or more for the query results). Also, every table that is used by a query requires one block. However, if two or more queries use the same table, only one block needs to be allocated.

The information provided by the `Qcache_lowmem_prunes` status variable can help you tune the query cache size. It counts the number of queries that have been removed from the cache to free up memory for caching new queries. The query cache uses a least recently used (LRU) strategy to decide which queries to remove from the cache. Tuning information is given in Section 5.11.3, “Query Cache Configuration”.

Chapter 6. Replication in MySQL

Replication capabilities allowing the databases on one MySQL server to be duplicated on another were introduced in MySQL 3.23.15. This chapter describes the various replication features provided by MySQL. It introduces replication concepts, shows how to set up replication servers, and serves as a reference to the available replication options. It also provides a list of frequently asked questions (with answers), and troubleshooting advice for solving problems.

For a description of the syntax of replication-related SQL statements, see Section 13.6, “Replication Statements”.

We suggest that you visit our Web site at http://www.mysql.com often and read updates to this chapter. Replication is constantly being improved, and we update the manual frequently with the most current information.

6.1. Introduction to Replication

MySQL 3.23.15 and up features support for one-way replication. One server acts as the master, while one or more other servers act as slaves. The master server writes updates to its binary log files, and maintains an index of the files to keep track of log rotation. These logs serve as a record of updates to be sent to slave servers. When a slave server connects to the master server, it informs the master of its last position within the logs since the last successfully propagated update. The slave catches up any updates that have occurred since then, and then blocks and waits for the master to notify it of new updates.

A slave server can also serve as a master if you want to set up chained replication servers.

Note that when you are using replication, all updates to the tables that are replicated should be performed on the master server. Otherwise, you must always be careful to avoid conflicts between updates that users make to tables on the master and updates that they make to tables on the slave.

One-way replication has benefits for robustness, speed, and system administration:

- Robustness is increased with a master/slave setup. In the event of problems with the master, you can switch to the slave as a backup.
- Better response time for clients can be achieved by splitting the load for processing client queries between the master and slave servers. `SELECT` queries may be sent to the slave to reduce the query processing load of the master. Statements that modify data should still be sent to the master so that the master and slave do not get out of sync. This load-balancing strategy is effective if non-updating queries dominate, but that is the normal case.
- Another benefit of using replication is that you can perform backups using a slave server without disturbing the master. The master continues to process updates while the backup is being made. See Section 5.7.1, “Database Backups”.

6.2. Replication Implementation Overview

MySQL replication is based on the master server keeping track of all changes to your databases (updates, deletes, and so on) in the binary logs. Therefore, to use replication, you must enable binary logging on the master server. See Section 5.9.4, “The Binary Log”.

Each slave server receives from the master the saved updates that the master has recorded in its binary log, so that the slave can execute the same updates on its copy of the data.

It is **very important** to realize that the binary log is simply a record starting from the fixed point in time at which you enable binary logging. Any slaves that you set up need copies of the databases on your master as they existed at the moment you enabled binary logging on the master. If you start your slaves with databases that are not the same as what was on the master **when the binary log was started**, your slaves may fail.

One way to copy the master's data to the slave is to use the `LOAD DATA FROM MASTER` statement. Be aware that `LOAD DATA FROM MASTER` is available only as of MySQL 4.0.0 and currently works only if all the tables on the master are `MyISAM` type. Also, this statement acquires a global read lock, so no updates on the master are possible while the tables are being transferred to the slave. When we implement lock-free hot table backup (in MySQL 5.0), this global read lock will no longer be necessary.

Due to these limitations, we recommend that at this point you use `LOAD DATA FROM MASTER` only if the dataset on the master is relatively small, or if a prolonged read lock on the master is acceptable. While the actual speed of `LOAD DATA FROM MASTER` may vary from system to system, a good rule of thumb for how long it takes is 1 second per 1MB of data. That is only a rough estimate, but you should get close to it if both master and slave are equivalent to 700MHz Pentium performance and are connected through a 100MBit/s network.

After the slave has been set up with a copy of the master's data, it connects to the master and waits for updates to process. If the master goes away or the slave loses connectivity with your master, it keeps trying to connect periodically until it is able to reconnect and resume listening for updates. The retry interval is controlled by the `--master-connect-retry` option. The default is 60 seconds.

Each slave keeps track of where it left off. The master server has no knowledge of how many slaves there are or which ones are up to date at any given time.

6.3. Replication Implementation Details

MySQL replication capabilities are implemented using three threads (one on the master server and two on the slave). When `START SLAVE` is issued, the slave creates an I/O thread. The I/O thread connects to the master and asks it to send the statements recorded in its binary logs. The master creates a thread to send the binary log contents to the slave. This thread can be identified as the `Binlog Dump` thread in the output of `SHOW PROCESSLIST` on the master. The slave I/O thread reads what the master `Binlog Dump` thread sends and simply copies it to some local files in the slave's data directory called relay logs. The third thread is the SQL thread, which the slave creates to read the relay logs and execute the updates they contain.

In the preceding description, there are three threads per slave. For a master that has multiple slaves, it creates one thread for each currently connected slave, and each slave has its own I/O and SQL threads.

For versions of MySQL before 4.0.2, replication involves only two threads (one on the master and one on the slave). The slave I/O and SQL threads are combined as a single thread, and no relay log files are used.

The advantage of using two slave threads is that statement reading and execution are separated into two independent tasks. The task of reading statements is not slowed down if statement execution is slow. For example, if the slave server has not been running for a while, its I/O thread can quickly fetch all the binary log contents from the master when the slave starts, even if the SQL thread lags far behind and may take hours to catch up. If the slave stops before the SQL thread has executed all the fetched statements, the I/O thread has at least fetched everything so that a safe copy of the statements is locally stored in the slave's relay logs for execution when next the slave starts. This allows the binary logs to be purged on the master, because it no longer need wait for the slave to fetch their contents.

The `SHOW PROCESSLIST` statement provides information that tells you what is happening on the master and on the slave regarding replication.

The following example illustrates how the three threads show up in `SHOW PROCESSLIST`. The output format is that used by `SHOW PROCESSLIST` as of MySQL version 4.0.15, when the content of the `State` column was changed to be more meaningful compared to earlier versions.

On the master server, the output from `SHOW PROCESSLIST` looks like this:

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 2
   User: root
   Host: localhost:32931
     db: NULL
Command: Binlog Dump
   Time: 94
  State: Has sent all binlog to slave; waiting for binlog to
         be updated
   Info: NULL
```

Here, thread 2 is a replication thread for a connected slave. The information indicates that all outstanding updates have been sent to the slave and that the master is waiting for more updates to occur.

On the slave server, the output from `SHOW PROCESSLIST` looks like this:

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 10
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 11
  State: Waiting for master to send event
   Info: NULL
*************************** 2. row ***************************
     Id: 11
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 11
  State: Has read all relay log; waiting for the slave I/O
         thread to update it
   Info: NULL
```

This information indicates that thread 10 is the I/O thread that is communicating with the master server, and thread 11 is the SQL thread that is processing the updates stored in the relay logs. Currently, both threads are idle, waiting for further updates.

Note that the value in the `Time` column can tell how late the slave is compared to the master. See Section 6.9, “Replication FAQ”.

6.3.1. Replication Master Thread States

The following list shows the most common states you see in the `State` column for the master's `Binlog Dump` thread. If you don't see any `Binlog Dump` threads on a master server, replication is not running. That is, no slaves currently are connected.

- `Sending binlog event to slave`

Binary logs consist of events, where an event is usually an update statement plus some other information. The thread has read an event from the binary log and is sending it to the slave.

- `Finished reading one binlog; switching to next binlog`

  The thread has finished reading a binary log file and is opening the next one to send to the slave.

- `Has sent all binlog to slave; waiting for binlog to be updated`

  The thread has read all outstanding updates from the binary logs and sent them to the slave. It is idle, waiting for new events to appear in the binary log resulting from new update statements being executed on the master.

- `Waiting to finalize termination`

  A very brief state that occurs as the thread is stopping.

6.3.2. Replication Slave I/O Thread States

The following list shows the most common states you see in the `State` column for a slave server I/O thread. Beginning with MySQL 4.1.1, this state also appears in the `Slave_IO_State` column displayed by the `SHOW SLAVE STATUS` statement. This means that you can get a good view of what is happening by using only `SHOW SLAVE STATUS`.

- `Connecting to master`

  The thread is attempting to connect to the master.

- `Checking master version`

  A very brief state that occurs just after the connection to the master is established.

- `Registering slave on master`

  A very brief state that occurs just after the connection to the master is established.

- `Requesting binlog dump`

  A very brief state that occurs just after the connection to the master is established. The thread sends to the master a request for the contents of its binary logs, starting from the requested binary log filename and position.

- `Waiting to reconnect after a failed binlog dump request`

  If the binary log dump request failed (due to disconnection), the thread goes into this state while it sleeps, then tries to reconnect periodically. The interval between retries can be specified using the `--master-connect-retry` option.

- `Reconnecting after a failed binlog dump request`

  The thread is trying to reconnect to the master.

- `Waiting for master to send event`

  The thread has connected to the master and is waiting for binary log events to arrive. This can last for a long time if the master is idle. If the wait lasts for `slave_read_timeout` seconds, a

timeout occurs. At that point, the thread considers the connection to be broken and make an attempt to reconnect.

- `Queueing master event to the relay log`

  The thread has read an event and is copying it to the relay log so that the SQL thread can process it.

- `Waiting to reconnect after a failed master event read`

  An error occurred while reading (due to disconnection). The thread is sleeping for `master-connect-retry` seconds before attempting to reconnect.

- `Reconnecting after a failed master event read`

  The thread is trying to reconnect to the master. When connection is established again, the state becomes `Waiting for master to send event`.

- `Waiting for the slave SQL thread to free enough relay log space`

  You are using a non-zero `relay_log_space_limit` value, and the relay logs have grown so much that their combined size exceeds this value. The I/O thread is waiting until the SQL thread frees enough space by processing relay log contents so that it can delete some relay log files.

- `Waiting for slave mutex on exit`

  A very brief state that occurs as the thread is stopping.

6.3.3. Replication Slave SQL Thread States

The following list shows the most common states you see in the `State` column for a slave server SQL thread:

- `Reading event from the relay log`

  The thread has read an event from the relay log so that it can process it.

- `Has read all relay log; waiting for the slave I/O thread to update it`

  The thread has processed all events in the relay log files and is waiting for the I/O thread to write new events to the relay log.

- `Waiting for slave mutex on exit`

  A very brief state that occurs as the thread is stopping.

The `State` column for the I/O thread may also show the text of a statement. This indicates that the thread has read an event from the relay log, extracted the statement from it, and is executing it.

6.3.4. Replication Relay and Status Files

By default, relay logs are named using filenames of the form *`host_name`*`-relay-bin.`*`nnnnnn`*, where *`host_name`* is the name of the slave server host and *`nnnnnn`* is a sequence number. Successive relay log files are created using successive sequence numbers, beginning with `000001` (`001` in MySQL 4.0 or older). The slave keeps track of relay logs currently in use in an index file. The default relay log

index filename is *host_name*-relay-bin.index. By default, these files are created in the slave's data directory. The default filenames may be overridden with the --relay-log and --relay-log-index server options. See Section 6.8, “Replication Startup Options”.

Relay logs have the same format as binary logs, so you can use **mysqlbinlog** to read them. A relay log is automatically deleted by the SQL thread as soon as it has executed all its events and no longer needs it). There is no explicit mechanism for deleting relay logs, because the SQL thread takes care of doing so. However, from MySQL 4.0.14, FLUSH LOGS rotates relay logs, which influences when the SQL thread deletes them.

A new relay log is created under the following conditions:

- When the I/O thread starts for the first time after the slave server starts. (In MySQL 5.0, a new relay log is created each time the I/O thread starts, not just the first time.)
- When the logs are flushed; for example, with FLUSH LOGS or **mysqladmin flush-logs**. (This creates a new relay log only as of MySQL 4.0.14.)
- When the size of the current relay log file becomes too large. The meaning of ``too large'' is determined as follows:
  - max_relay_log_size, if max_relay_log_size > 0
  - max_binlog_size, if max_relay_log_size = 0 or MySQL is older than 4.0.14

A slave replication server creates two additional small files in the data directory. These are status files and are named master.info and relay-log.info by default. They contain information like that shown in the output of the SHOW SLAVE STATUS statement (see Section 13.6.2, “SQL Statements for Controlling Slave Servers” for a description of this statement). As disk files, they survive a slave server's shutdown. The next time the slave starts up, it reads these files to determine how far it has proceeded in reading binary logs from the master and in processing its own relay logs.

The master.info file is updated by the I/O thread. Before MySQL 4.1, the correspondence between the lines in the file and the columns displayed by SHOW SLAVE STATUS is as follows:

Line	Description
1	Master_Log_File
2	Read_Master_Log_Pos
3	Master_Host
4	Master_User
5	Password (not shown by SHOW SLAVE STATUS)
6	Master_Port
7	Connect_Retry

As of MySQL 4.1, the file includes a line count and information about SSL options:

Line	Description
1	Number of lines in the file
2	Master_Log_File
3	Read_Master_Log_Pos
4	Master_Host
5	Master_User

6	Password (not shown by `SHOW SLAVE STATUS`)
7	`Master_Port`
8	`Connect_Retry`
9	`Master_SSL_Allowed`
10	`Master_SSL_CA_File`
11	`Master_SSL_CA_Path`
12	`Master_SSL_Cert`
13	`Master_SSL_Cipher`
14	`Master_SSL_Key`

The `relay-log.info` file is updated by the SQL thread. The correspondence between the lines in the file and the columns displayed by `SHOW SLAVE STATUS` is as follows:

Line	**Description**
1	`Relay_Log_File`
2	`Relay_Log_Pos`
3	`Relay_Master_Log_File`
4	`Exec_Master_Log_Pos`

When you back up your slave's data, you should back up these two small files as well, along with the relay log files. They are needed to resume replication after you restore the slave's data. If you lose the relay logs but still have the `relay-log.info` file, you can check it to determine how far the SQL thread has executed in the master binary logs. Then you can use `CHANGE MASTER TO` with the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options to tell the slave to re-read the binary logs from that point. This requires that the binary logs still exist on the master server.

If your slave is subject to replicating `LOAD DATA INFILE` statements, you should also back up any `SQL_LOAD-*` files that exist in the directory that the slave uses for this purpose. The slave needs these files to resume replication of any interrupted `LOAD DATA INFILE` operations. The directory location is specified using the `--slave-load-tmpdir` option. Its default value, if not specified, is the value of the `tmpdir` variable.

6.4. How to Set Up Replication

Here is a quick description of how to set up complete replication of your current MySQL server. It assumes that you want to replicate all your databases and have not configured replication before. You need to shut down your master server briefly to complete the steps outlined here.

The procedure is written in terms of setting up a single slave, but you can use it to set up multiple slaves.

While this method is the most straightforward way to set up a slave, it is not the only one. For example, if you have a snapshot of the master's data, and the master has its server ID set and binary logging enabled, you can set up a slave without shutting down the master or even blocking updates to it. For more details, please see Section 6.9, “Replication FAQ”.

If you want to administer a MySQL replication setup, we suggest that you read this entire chapter through and try all statements mentioned in Section 13.6.1, “SQL Statements for Controlling Master Servers” and Section 13.6.2, “SQL Statements for Controlling Slave Servers”. You should also familiarize yourself with replication startup options described in Section 6.8, “Replication Startup Options”.

Note that this procedure and some of the replication SQL statements in later sections refer to the SUPER privilege. Prior to MySQL 4.0.2, use the PROCESS privilege instead.

1. Make sure that you have a recent version of MySQL installed on the master and slaves, and that these versions are compatible according to the table shown in Section 6.5, “Replication Compatibility Between MySQL Versions”.

   Please do not report bugs until you have verified that the problem is present in the latest release.

2. Set up an account on the master server that the slave server can use to connect. This account must be given the REPLICATION SLAVE privilege. If the account is used only for replication (which is recommended), you don't need to grant any additional privileges.

   Suppose that your domain is mydomain.com and you want to create an account with a username of repl such that slave servers can use the account to access the master server from any host in your domain using a password of slavepass. To create the account, this use GRANT statement:

   ```
   mysql> GRANT REPLICATION SLAVE ON *.*
       -> TO 'repl'@'%.mydomain.com' IDENTIFIED BY 'slavepass';
   ```

   For MySQL versions older than 4.0.2, the REPLICATION SLAVE privilege does not exist. Grant the FILE privilege instead:

   ```
   mysql> GRANT FILE ON *.*
       -> TO 'repl'@'%.mydomain.com' IDENTIFIED BY 'slavepass';
   ```

   If you plan to use the LOAD TABLE FROM MASTER or LOAD DATA FROM MASTER statements from the slave host, you need to grant this account additional privileges:

   - Grant the account the SUPER and RELOAD global privileges.
   - Grant the SELECT privilege for all tables that you want to load. Any master tables from which the account cannot SELECT are ignored by LOAD DATA FROM MASTER.

3. If you are using only MyISAM tables, flush all the tables and block write statements by executing a FLUSH TABLES WITH READ LOCK statement.

   ```
   mysql> FLUSH TABLES WITH READ LOCK;
   ```

   Leave the client running from which you issue the FLUSH TABLES statement so that the read lock remains in effect. (If you exit the client, the lock is released.) Then take a snapshot of the data on your master server.

   The easiest way to create a snapshot is to use an archiving program to make a binary backup of the databases in your master's data directory. For example, use **tar** on Unix, or **PowerArchiver**, **WinRAR**, **WinZip**, or any similar software on Windows. To use **tar** to create an archive that includes all databases, change location into the master server's data directory, then execute this command:

   ```
   shell> tar -cvf /tmp/mysql-snapshot.tar .
   ```

   If you want the archive to include only a database called this_db, use this command instead:

   ```
   shell> tar -cvf /tmp/mysql-snapshot.tar ./this_db
   ```

Then copy the archive file to the `/tmp` directory on the slave server host. On that machine, change location into the slave's data directory, and unpack the archive file using this command:

```
shell> tar -xvf /tmp/mysql-snapshot.tar
```

You may not want to replicate the `mysql` database if the slave server has a different set of user accounts from those that exist on the master. In this case, you should exclude it from the archive. You also need not include any log files in the archive, or the `master.info` or `relay-log.info` files.

While the read lock placed by `FLUSH TABLES WITH READ LOCK` is in effect, read the value of the current binary log name and offset on the master:

```
mysql > SHOW MASTER STATUS;
+---------------+----------+--------------+------------------+
| File          | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+---------------+----------+--------------+------------------+
| mysql-bin.003 | 73       | test         | manual,mysql     |
+---------------+----------+--------------+------------------+
```

The `File` column shows the name of the log, while `Position` shows the offset. In this example, the binary log value is `mysql-bin.003` and the offset is 73. Record the values. You need to use them later when you are setting up the slave. They represent the replication coordinates at which the slave should begin processing new updates from the master.

After you have taken the snapshot and recorded the log name and offset, you can re-enable write activity on the master:

```
mysql> UNLOCK TABLES;
```

If you are using `InnoDB` tables, ideally you should use the `InnoDB Hot Backup` tool. It takes a consistent snapshot without acquiring any locks on the master server, and records the log name and offset corresponding to the snapshot to be later used on the slave. `InnoDB Hot Backup` is a non-free (commercial) additional tool that is not included in the standard MySQL distribution. See the `InnoDB Hot Backup` home page at http://www.innodb.com/manual.php for detailed information and screenshots.

Without the `Hot Backup` tool, the quickest way to take a binary snapshot of `InnoDB` tables is to shut down the master server and copy the `InnoDB` data files, log files, and table definition files (`.frm` files). To record the current log file name and offset, you should issue the following statements before you shut down the server:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SHOW MASTER STATUS;
```

Then record the log name and the offset from the output of `SHOW MASTER STATUS` as was shown earlier. After recording the log name and the offset, shut down the server *without* unlocking the tables to make sure that the server goes down with the snapshot corresponding to the current log file and offset:

```
shell> mysqladmin -u root shutdown
```

An alternative that works for both `MyISAM` and `InnoDB` tables is to take an SQL dump of the master instead of a binary copy as described in the preceding discussion. For this, you can use

mysqldump --master-data on your master and later load the SQL dump file into your slave. However, this is slower than doing a binary copy.

If the master has been previously running without `--log-bin` enabled, the log name and position values displayed by `SHOW MASTER STATUS` or **mysqldump --master-data** are empty. In that case, the values that you need to use later when specifying the slave's log file and position are the empty string (`''`) and `4`.

4. Make sure that the `[mysqld]` section of the `my.cnf` file on the master host includes a `log-bin` option. The section should also have a `server-id=master_id` option, where `master_id` must be a positive integer value from 1 to 2^32 # 1. For example:

   ```
   [mysqld]
   log-bin=mysql-bin
   server-id=1
   ```

   If those options are not present, add them and restart the server.

5. Stop the server that is to be used as a slave server and add the following to its `my.cnf` file:

   ```
   [mysqld]
   server-id=slave_id
   ```

   The `slave_id` value, like the `master_id` value, must be a positive integer value from 1 to 2^32 # 1. In addition, it is very important that the ID of the slave be different from the ID of the master. For example:

   ```
   [mysqld]
   server-id=2
   ```

   If you are setting up multiple slaves, each one must have a unique `server-id` value that differs from that of the master and from each of the other slaves. Think of `server-id` values as something similar to IP addresses: These IDs uniquely identify each server instance in the community of replication partners.

   If you don't specify a `server-id` value, it is set to 1 if you have not defined `master-host`, otherwise it is set to 2. Note that in the case of `server-id` omission, a master refuses connections from all slaves, and a slave refuses to connect to a master. Thus, omitting `server-id` is good only for backup with a binary log.

6. If you made a binary backup of the master server's data, copy it to the slave server's data directory before starting the slave. Make sure that the privileges on the files and directories are correct. The user that the server MySQL runs as must able to read and write the files, just as on the master.

   If you made a backup using **mysqldump**, start the slave first (see next step).

7. Start the slave server. If it has been replicating previously, start the slave server with the `--skip-slave-start` option so that it doesn't immediately try to connect to its master. You also may want to start the slave server with the `--log-warnings` option (enabled by default as of MySQL 4.0.19 and 4.1.2), to get more messages in the error log about problems (for example, network or connection problems). As of MySQL 4.0.21 and 4.1.3, aborted connections are not logged to the error log unless the value is greater than 1.

8. If you made a backup of the master server's data using **mysqldump**, load the dump file into the slave server:

   ```
   shell> mysql -u root -p < dump_file.sql
   ```

9. Execute the following statement on the slave, replacing the option values with the actual values relevant to your system:

```
mysql> CHANGE MASTER TO
    ->     MASTER_HOST='master_host_name',
    ->     MASTER_USER='replication_user_name',
    ->     MASTER_PASSWORD='replication_password',
    ->     MASTER_LOG_FILE='recorded_log_file_name',
    ->     MASTER_LOG_POS=recorded_log_position;
```

   The following table shows the maximum length for the string options:

MASTER_HOST	60
MASTER_USER	16
MASTER_PASSWORD	32
MASTER_LOG_FILE	255

10. Start the slave threads:

```
mysql> START SLAVE;
```

After you have performed this procedure, the slave should connect to the master and catch up on any updates that have occurred since the snapshot was taken.

If you have forgotten to set the `server-id` value for the master, slaves are not able to connect to it.

If you have forgotten to set the `server-id` value for the slave, you get the following error in its error log:

```
Warning: You should set server-id to a non-0 value if master_host is set;
we will force server id to 2, but this MySQL server will not act as a slave.
```

You also find error messages in the slave's error log if it is not able to replicate for any other reason.

Once a slave is replicating, you can find in its data directory one file named `master.info` and another named `relay-log.info`. The slave uses these two files to keep track of how much of the master's binary log it has processed. **Do not** remove or edit these files, unless you really know what you are doing and understand the implications. Even in that case, it is preferred that you use the `CHANGE MASTER TO` statement.

Note: The content of `master.info` overrides some options specified on the command line or in `my.cnf`. See Section 6.8, “Replication Startup Options” for more details.

Once you have a snapshot, you can use it to set up other slaves by following the slave portion of the procedure just described. You do not need to take another snapshot of the master; you can use the same one for each slave.

6.5. Replication Compatibility Between MySQL

Versions

The original binary log format was developed in MySQL 3.23. It changed in MySQL 4.0, and again in MySQL 5.0. This has consequences when you upgrade servers in a replication setup, as described in Section 6.6, “Upgrading a Replication Setup”.

As far as replication is concerned, any MySQL 4.1.x version and any 4.0.x version are identical, because they all use the same binary log format. Thus, any servers from these versions are compatible, and replication between them should work seamlessly. The exceptions to this compatibility is that versions from MySQL 4.0.0 to 4.0.2 were very early development versions that should not be used anymore. (These were the alpha versions in the 4.0 release series. Compatibility for them is still documented in the manual included with their distributions.)

The following table indicates master/slave replication compatibility between different versions of MySQL.

		Master	**Master**	**Master**
		3.23.33 and up	**4.0.3 and up or any 4.1.x**	**5.0.0**
Slave	**3.23.33 and up**	yes	no	no
Slave	**4.0.3 and up**	yes	yes	no
Slave	**5.0.0**	yes	yes	yes

As a general rule, we recommended using recent MySQL versions, because replication capabilities are continually being improved. We also recommend using the same version for both the master and the slave.

The preceding information pertains the replication compatibility at the protocol level. There can also be SQL-level compatibility constraints, as discussed in Section 6.7, “Replication Features and Known Problems”.

6.6. Upgrading a Replication Setup

When you upgrade servers that participate in a replication setup, the procedure for upgrading depends on the current server versions and the version to which you are upgrading.

6.6.1. Upgrading Replication to 4.0 or 4.1

This section applies to upgrading replication from MySQL 3.23 to 4.0 or 4.1. A 4.0 server should be 4.0.3 or newer, as mentioned in Section 6.5, “Replication Compatibility Between MySQL Versions”.

When you upgrade a master from MySQL 3.23 to MySQL 4.0 or 4.1, you should first ensure that all the slaves of this master are at 4.0 or 4.1. If that is not the case, you should first upgrade your slaves: Shut down each one, upgrade it, restart it, and restart replication.

The upgrade can safely be done using the following procedure, assuming that you have a 3.23 master to upgrade and the slaves are 4.0 or 4.1. Note that after the master has been upgraded, you should not restart replication using any old 3.23 binary logs, because this unfortunately confuses the 4.0 or 4.1 slaves.

1. Block all updates on the master by issuing a `FLUSH TABLES WITH READ LOCK` statement.

2. Wait until all the slaves have caught up with all changes from the master server. Use `SHOW MASTER STATUS` on the master to obtain its current binary log file and position. Then, for each slave,

use those values with a `SELECT MASTER_POS_WAIT()` statement. The statement blocks on the slave and returns when the slave has caught up. Then run `STOP SLAVE` on the slave.

3. Stop the master server and upgrade it to MySQL 4.0 or 4.1.

4. Restart the master server and record the name of its newly created binary log. You can obtain the name of the file by issuing a `SHOW MASTER STATUS` statement on the master. Then issue these statements on each slave:

```
mysql> CHANGE MASTER TO MASTER_LOG_FILE='binary_log_name',
    ->     MASTER_LOG_POS=4;
mysql> START SLAVE;
```

6.6.2. Upgrading Replication to 5.0

This section applies to upgrading replication from MySQL 3.23, 4.0, or 4.1 to 5.0.0. A 4.0 server should be 4.0.3 or newer, as mentioned in Section 6.5, “Replication Compatibility Between MySQL Versions”.

First, note that MySQL 5.0.0 is an alpha release. It is intended to work better than older versions (easier upgrade, replication of some important session variables such as `sql_mode`; see Section D.1.5, “Changes in release 5.0.0 (22 Dec 2003: Alpha)”). However it has not yet been extensively tested. As with any alpha release, we recommend that you not use it in critical production environments yet.

When you upgrade a master from MySQL 3.23, 4.0, or 4.1 to 5.0.0, you should first ensure that all the slaves of this master are 5.0.0. If that's not the case, you should first upgrade your slaves. To upgrade each slave, just shut it down, upgrade it to 5.0.0, restart it, and restart replication. The 5.0.0 slave is able to read its old relay logs that were written before the upgrade and execute the statements they contain. Relay logs created by the slave after the upgrade are in 5.0.0 format.

After the slaves have been upgraded, shut down your master, upgrade it to 5.0.0, and restart it. The 5.0.0 master is able to read its old binary logs that were written before the upgrade and send them to the 5.0.0 slaves. The slaves recognize the old format and handle it properly. Binary logs created by master after the upgrade are in 5.0.0 format. These too are recognized by the 5.0.0 slaves.

In other words, there are no measures to take when upgrading to 5.0.0, except that slaves must be 5.0.0 before you can upgrade the master to 5.0.0. Note that downgrading from 5.0.0 to older versions does not work so automatically: You must ensure that any 5.0.0 binary logs or relay logs have been fully processed, so that you can remove them before proceeding with the downgrade.

6.7. Replication Features and Known Problems

In general, replication compatibility at the SQL level requires that any features used be supported by both the master and the slave servers. For example, the `GROUP_CONCAT()` function is available in MySQL 4.1 and up. If you use this function on the master server, you cannot replicate to a slave server that is older than MySQL 4.1.

The following list provides details about what is supported and what is not. Additional `InnoDB`-specific information about replication is given in Section 15.7.5, “`InnoDB` and MySQL Replication”.

- Replication is done correctly with `AUTO_INCREMENT`, `LAST_INSERT_ID()`, and `TIMESTAMP` values.

- The `USER()`, `UUID()`, and `LOAD_FILE()` functions are replicated without changes and thus do not work reliably on the slave. This is also true for `CONNECTION_ID()` in slave versions older

than 4.1.1. The **new** PASSWORD() function in MySQL 4.1 is well replicated in masters from 4.1.1 and up; your slaves also must be 4.1.1 or above to replicate it. If you have older slaves and need to replicate PASSWORD() from your 4.1.x master, you must start your master with the --old-password option, so that it uses the old implementation of PASSWORD(). (Note that the PASSWORD() implementation in MySQL 4.1.0 differs from every other version of MySQL. It is best to avoid 4.1.0 in a replication situation.)

- The functions handling user-level locks: GET_LOCK(), RELEASE_LOCK(), IS_FREE_LOCK(), IS_USED_LOCK() are replicated without the slave knowing the concurrency context on master; so these functions should not be used to insert into a master's table as the content on slave would differ (i.e. do not do INSERT INTO mytable VALUES(GET_LOCK(...))).

- The FOREIGN_KEY_CHECKS variable is replicated as of MySQL 4.0.14. The SQL_MODE, UNIQUE_CHECKS, and SQL_AUTO_IS_NULL variables are replicated as of 5.0.0. The TABLE_TYPE, also known as STORAGE_ENGINE variable is not yet replicated, which is a good thing for replication between different storage engines.

- Starting from MySQL 5.0.3 (master and slave), replication works fine even if the master and slave have different global character set variables; with the exception of the LOAD DATA INFILE statement, whose replication will be made charset-aware in later 5.0 releases.

- This paragraph applies to replication between MySQL servers using different character sets where the master runs MySQL 4.1 (and slave runs 4.1 or newer). First, you must ALWAYS use the same **global** character set and collation (--default-character-set, --default-collation) on the master and the slave. Otherwise, you may get duplicate-key errors on the slave, because a key that is regarded as unique in the master's character set may not be unique in the slave's character set. Second, if the master is strictly older than MySQL 4.1.3, the character set of the session should never be made different from its global value (in other words, don't use SET NAMES, SET CHARACTER SET etc) because this character set change is not known to the slave. If the master is 4.1.3 or newer, and the slave too, the session can freely set its local value of character set variables (NAMES, CHARACTER SET, COLLATION_CLIENT, COLLATION_SERVER etc) as these settings are written to the binary log and then known to the slave. The session however is prevented from changing the **global** value of these; as said previously the master and slave must always have identical global character set values. There also is one last limitation: if on the master you have databases with different character sets from the global collation_server value, you should design your CREATE TABLE statements so that they don't implicitly rely on the default database's character set, because there currently is a bug (Bug #2326 [http://bugs.mysql.com/2326]); a good workaround is to explicitly state the character set and collation in a clause of the CREATE TABLE.

- For both master and slave same system time zone should be set (otherwise some statements, for example statements using NOW() or FROM_UNIXTIME() functions, won't be replicated properly). One could set time zone in which MySQL server runs by using --timezone=timezone_name option of mysqld_safe script or by setting TZ environment variable. Also starting from version 4.1.3 both master and slave should have same default connection time zone set, i.e. --default-time-zone parameter should have the same value for both master and slave. CONVERT_TZ(...,...,@[global|session].time_zone) is not properly replicated.

- Session variables are not replicated properly when used in statements which update tables; for example: SET MAX_JOIN_SIZE=1000; INSERT INTO mytable VALUES(@MAX_JOIN_SIZE); will not insert the same data on master and on slave. This also applies to the common SET TIME_ZONE=...; INSERT INTO mytable VALUES(CONVERT_TZ(...,...,@time_zone)), which will be fixed in MySQL 5.0.x.

- It is possible to replicate transactional tables on the master using non-transactional tables on the slave. For example, you can replicate an InnoDB master table as a MyISAM slave table. However, if you do this, there are problems if the slave is stopped in the middle of a BEGIN/COMMIT block, because the slave restarts at the beginning of the BEGIN block. This issue is on our TODO and will be fixed in the near future.

- Update statements that refer to user variables (that is, variables of the form `@var_name`) are badly replicated in 3.23 and 4.0. This is fixed in 4.1. Note that user variable names are case insensitive starting from MySQL 5.0. You should take this into account when setting up replication between 5.0 and an older version.

- The slave can connect to the master using SSL if both are 4.1.1 or newer.

- Starting from MySQL 4.1.11 and 5.0.3, there is a global system variable `slave_transaction_retries`: If the replication slave SQL thread fails to execute a transaction because of an `InnoDB` deadlock or exceeded `innodb_lock_wait_timeout`, it automatically retries `slave_transaction_retries` times before stopping with an error. The default value is 0 in MySQL 4.1 and 10 in MySQL 5.0.

- If a `DATA DIRECTORY` or `INDEX DIRECTORY` clause is used in a `CREATE TABLE` statement on the master server, the clause is also used on the slave. This can cause problems if no corresponding directory exists in the slave host filesystem or exists but is not accessible to the slave server. Starting from MySQL 4.0.15, there is a `sql_mode` option called `NO_DIR_IN_CREATE`. If the slave server is run with its SQL mode set to include this option, it ignores the clauses before replicating the `CREATE TABLE` statement. The result is that the `MyISAM` data and index files are created in the table's database directory.

- It is possible for the data on the master and slave to become different if a query is designed in such a way that the data modification is non-deterministic; that is, left to the will of the query optimizer. (That generally is not a good practice anyway, even outside of replication!). For a detailed explanation of this issue, see Section 1.5.7.3, “Open Bugs and Design Deficiencies in MySQL”.

- If on master a `LOAD DATA INFILE` is interrupted in the middle (integrity constraint violation, killed connection...), the slave skips this `LOAD DATA INFILE` entirely. It means that if this command permanently inserted/updated some table records before being interrupted, these modifications won't be replicated to the slave. This will be fixed when MySQL features a record-level binary log format, in development.

- Before MySQL 4.1.1, `FLUSH`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements are not written to the binary log and thus are not replicated to the slaves. This is not normally a problem because these statements do not modify table data. However, it can cause difficulties under certain circumstances. If you replicate the privilege tables in the `mysql` database and update those tables directly without using the `GRANT` statement, you must issue a `FLUSH PRIVILEGES` statement on your slaves to put the new privileges into effect. Also if you use `FLUSH TABLES` when renaming a `MyISAM` table that is part of a `MERGE` table, you have to issue `FLUSH TABLES` manually on the slaves. As of MySQL 4.1.1, these statements are written to the binary log (unless you specify `NO_WRITE_TO_BINLOG`, or its alias `LOCAL`). Exceptions are that `FLUSH LOGS`, `FLUSH MASTER`, `FLUSH SLAVE`, and `FLUSH TABLES WITH READ LOCK` are not logged in any case. (Any of them may cause problems if replicated to a slave.) For a syntax example, see Section 13.5.5.2, “`FLUSH` Syntax”.

- MySQL only supports one master and many slaves. Later we will add a voting algorithm to automatically change master if something goes wrong with the current master. We will also introduce ``agent'' processes to help do load balancing by sending `SELECT` queries to different slaves.

- When a server shuts down and restarts, its `MEMORY` (`HEAP`) tables become empty. As of MySQL 4.0.18, the master replicates this effect as follows: The first time that the master uses each `MEMORY` table after startup, it notifies slaves that the table needs to be emptied by writing a `DELETE FROM` statement for the table to its binary log. See Section 14.3, “The `MEMORY` (`HEAP`) Storage Engine” for more details.

- Temporary tables are replicated with the exception of the case that you shut down the slave server (not just the slave threads) and you have some replicated temporary tables that are used in update statements that have not yet been executed on the slave. If you shut down the slave server, the tem-

porary tables needed by those updates no longer are available when the slave starts again. To avoid this problem, do not shut down the slave while it has temporary tables open. Instead, use this procedure:

1. Issue a `STOP SLAVE` statement.
2. Use `SHOW STATUS` to check the value of the `Slave_open_temp_tables` variable.
3. If the value is 0, issue a **mysqladmin shutdown** command to shut down the slave.
4. If the value is not 0, restart the slave threads with `START SLAVE`.
5. Repeat the procedure later to see if you have better luck next time.

We have plans to fix this problem in the near future.

- It is safe to connect servers in a circular master/slave relationship with the `--log-slave-updates` option specified. Note, however, that many statements do not work correctly in this kind of setup unless your client code is written to take care of the potential problems that can occur from updates that occur in different sequence on different servers.

  This means that you can create a setup such as this:

  ```
  A -> B -> C -> A
  ```

  Server IDs are encoded in the binary log events, so server A knows when an event that it reads was originally created by itself and does not execute the event (unless server A was started with the `--replicate-same-server-id` option, which is meaningful only in rare setups). Thus, there are no infinite loops. But this circular setup works only if you perform no conflicting updates between the tables. In other words, if you insert data in both A and C, you should never insert a row in A that may have a key that conflicts with a row inserted in C. You should also not update the same rows on two servers if the order in which the updates are applied is significant.

- If a statement on the slave produces an error, the slave SQL thread terminates, and the slave writes a message to its error log. You should then connect to the slave manually, fix the problem (for example, a non-existent table), and then run `START SLAVE`.

- It is safe to shut down a master server and restart it later. If a slave loses its connection to the master, the slave tries to reconnect immediately. If that fails, the slave retries periodically. (The default is to retry every 60 seconds. This may be changed with the `--master-connect-retry` option.) The slave also is able to deal with network connectivity outages. However, the slave does notice the network outage only after receiving no data from the master for `slave_net_timeout` seconds. If your outages are short, you may want to decrease `slave_net_timeout`. See Section 5.2.3, “Server System Variables”.

- Shutting down the slave (cleanly) is also safe, as it keeps track of where it left off. Unclean shutdowns might produce problems, especially if disk cache was not flushed to disk before the system went down. Your system fault tolerance is greatly increased if you have a good uninterruptible power supply. Unclean shutdowns of the master may cause inconsistencies between the content of tables and the binary log in master; this can be avoided by using `InnoDB` tables and the `--innodb-safe-binlog` option on the master. See Section 5.9.4, “The Binary Log”.

- Due to the non-transactional nature of `MyISAM` tables, it is possible to have a statement that only partially updates a table and returns an error code. This can happen, for example, on a multiple-row insert that has one row violating a key constraint, or if a long update statement is killed after updating some of the rows. If that happens on the master, the slave thread exits and waits for the database administrator to decide what to do about it unless the error code is legitimate and the statement execution results in the same error code. If this error code validation behavior is not desirable, some or

all errors can be masked out (ignored) with the `--slave-skip-errors` option. This option is available starting with MySQL 3.23.47.

- If you update transactional tables from non-transactional tables inside a `BEGIN/COMMIT` segment, updates to the binary log may be out of sync if some thread changes the non-transactional table before the transaction commits. This is because the transaction is written to the binary log only when it is committed.

- Before version 4.0.15, any update to a non-transactional table is written to the binary log at once when the update is made, whereas transactional updates are written on `COMMIT` or not written at all if you use `ROLLBACK`. You must take this into account when updating both transactional tables and non-transactional tables within the same transaction. (This is true not only for replication, but also if you are using binary logging for backups.) In version 4.0.15, we changed the logging behavior for transactions that mix updates to transactional and non-transactional tables, which solves the problems (order of statements is good in the binary log, and all needed statements are written to the binary log even in case of `ROLLBACK`). The problem that remains is when a second connection updates the non-transactional table while the first connection's transaction is not finished yet; wrong order can still occur, because the second connection's update is written immediately after it is done.

- When a 4.x slave replicates a `LOAD DATA INFILE` from a 3.23 master, the values of the `Exec_Master_Log_Pos` and `Relay_Log_Space` columns of `SHOW SLAVE STATUS` become incorrect. The incorrectness of `Exec_Master_Log_Pos` causes a problem when you stop and restart replication; so it is a good idea to correct the value before this, by doing `FLUSH LOGS` on the master. These bugs are fixed in MySQL 5.0.0 slaves.

The following table lists replication problems in MySQL 3.23 that are fixed in MySQL 4.0:

- `LOAD DATA INFILE` is handled properly, as long as the data file still resides on the master server at the time of update propagation.

- `LOAD DATA LOCAL INFILE` is no longer skipped on the slave as it was in 3.23.

- In 3.23, `RAND()` in updates does not replicate properly. Use `RAND(some_non_rand_expr)` if you are replicating updates with `RAND()`. You can, for example, use `UNIX_TIMESTAMP()` as the argument to `RAND()`.

6.8. Replication Startup Options

On both the master and the slave, you must use the `server-id` option to establish a unique replication ID for each server. You should pick a unique positive integer in the range from 1 to 2^32 # 1 for each master and slave. Example: `server-id=3`

The options that you can use on the master server for controlling binary logging are described in Section 5.9.4, “The Binary Log”.

The following table describes the options you can use on slave replication servers. You can specify them on the command line or in an option file.

Some slave server replication options are handled in a special way, in the sense that they are ignored if a `master.info` file exists when the slave starts and contains values for the options. The following options are handled this way:

- `--master-host`

- `--master-user`
- `--master-password`
- `--master-port`
- `--master-connect-retry`

As of MySQL 4.1.1, the following options also are handled specially:

- `--master-ssl`
- `--master-ssl-ca`
- `--master-ssl-capath`
- `--master-ssl-cert`
- `--master-ssl-cipher`
- `--master-ssl-key`

The `master.info` file format in 4.1.1 changed to include values corresponding to the SSL options. In addition, the 4.1.1 file format includes as its first line the number of lines in the file. If you upgrade an older server to 4.1.1, the new server upgrades the `master.info` file to the new format automatically when it starts. However, if you downgrade a 4.1.1 or newer server to a version older than 4.1.1, you should manually remove the first line before starting the older server for the first time. Note that, in this case, the downgraded server no longer can use an SSL connection to communicate with the master.

If no `master.info` file exists when the slave server starts, it uses values for those options that are specified in option files or on the command line. This occurs when you start the server as a replication slave for the very first time, or when you have run `RESET SLAVE` and shut down and restarted the slave server.

If the `master.info` file exists when the slave server starts, the server ignores those options. Instead, it uses the values found in the `master.info` file.

If you restart the slave server with different values of the startup options that correspond to values in the `master.info` file, the different values have no effect, because the server continues to use the `master.info` file. To use different values, you must either restart after removing the `master.info` file or (preferably) use the `CHANGE MASTER TO` statement to reset the values while the slave is running.

Suppose that you specify this option in your `my.cnf` file:

```
[mysqld]
master-host=some_host
```

The first time you start the server as a replication slave, it reads and uses that option from the `my.cnf` file. The server then records the value in the `master.info` file. The next time you start the server, it reads the master host value from the `master.info` file only and ignores the value in the option file. If you modify the `my.cnf` file to specify a different master host of `some_other_host`, the change still has no effect. You should use `CHANGE MASTER TO` instead.

Because the server gives an existing `master.info` file precedence over the startup options just described, you might prefer not to use startup options for these values at all, and instead specify them by using the `CHANGE MASTER TO` statement. See Section 13.6.2.1, “`CHANGE MASTER TO` Syntax”.

This example shows a more extensive use of startup options to configure a slave server:

```
[mysqld]
server-id=2
master-host=db-master.mycompany.com
master-port=3306
master-user=pertinax
master-password=freitag
master-connect-retry=60
report-host=db-slave.mycompany.com
```

The following list describes startup options for controlling replication: Many of these options can be reset while the server is running by using the `CHANGE MASTER TO` statement. Others, such as the `--replicate-*` options, can be set only when the slave server starts. We plan to fix this.

- `--log-slave-updates`

  Normally, updates received from a master server by a slave are not logged to its binary log. This option tells the slave to log the updates performed by its SQL thread to the slave's own binary log. For this option to have any effect, the slave must also be started with the `--log-bin` option to enable binary logging. `--log-slave-updates` is used when you want to chain replication servers. For example, you might want a setup like this:

  ```
  A -> B -> C
  ```

  That is, A serves as the master for the slave B, and B serves as the master for the slave C. For this to work, B must be both a master and a slave. You must start both A and B with `--log-bin` to enable binary logging, and B with the `--log-slave-updates` option.

- `--log-warnings`

  Makes the slave print more messages to the error log about what it is doing. For example, it warns you that it succeeded in reconnecting after a network/connection failure, and warns you about how each slave thread started. This option is enabled by default as of MySQL 4.0.19 and 4.1.2; to disable it, use `--skip-log-warnings`. As of MySQL 4.0.21 and 4.1.3, aborted connections are not logged to the error log unless the value is greater than 1.

  This option is not limited to replication use only. It produces warnings across a spectrum of server activities.

- `--master-connect-retry=`*`seconds`*

  The number of seconds the slave thread sleeps before retrying to connect to the master in case the master goes down or the connection is lost. The value in the `master.info` file takes precedence if it can be read. If not set, the default is 60.

- `--master-host=`*`host`*

  The hostname or IP number of the master replication server. If this option is not given, the slave thread does not start. The value in `master.info` takes precedence if it can be read.

- `--master-info-file=`*`file_name`*

  The name to use for the file in which the slave records information about the master. The default name is `mysql.info` in the data directory.

- `--master-password=password`

  The password of the account that the slave thread uses for authentication when connecting to the master. The value in the `master.info` file takes precedence if it can be read. If not set, an empty password is assumed.

- `--master-port=port_number`

  The TCP/IP port the master is listening on. The value in the `master.info` file takes precedence if it can be read. If not set, the compiled-in setting is assumed. If you have not tinkered with **configure** options, this should be 3306.

- `--master-ssl`, `--master-ssl-ca=file_name`, `--master-ssl-capath=directory_name`, `--master-ssl-cert=file_name`, `--master-ssl-cipher=cipher_list`, `--master-ssl-key=file_name`

  These options are used for setting up a secure replication connection to the master server using SSL. Their meanings are the same as the corresponding `--ssl`, `--ssl-ca`, `--ssl-capath`, `--ssl-cert`, `--ssl-cipher`, `--ssl-key` options described in Section 5.6.7.5, “SSL Command-Line Options”. The values in the `master.info` file take precedence if they can be read.

  These options are operational as of MySQL 4.1.1.

- `--master-user=username`

  The username of the account that the slave thread uses for authentication when connecting to the master. The account must have the `REPLICATION SLAVE` privilege. (Prior to MySQL 4.0.2, it must have the `FILE` privilege instead.) The value in the `master.info` file takes precedence if it can be read. If the master user is not set, user `test` is assumed.

- `--max-relay-log-size=#`

  To rotate the relay log automatically. See Section 5.2.3, “Server System Variables”.

  This option is available as of MySQL 4.0.14.

- `--read-only`

  This option causes the slave to allow no updates except from slave threads or from users with the `SUPER` privilege. This can be useful to ensure that a slave server accepts no updates from clients.

  This option is available as of MySQL 4.0.14.

- `--relay-log=file_name`

  The name for the relay log. The default name is *`host_name`*`-relay-bin.`*`nnnnnn`*, where *`host_name`* is the name of the slave server host and *`nnnnnn`* indicates that relay logs are created in numbered sequence. You can specify the option to create hostname-independent relay log names, or if your relay logs tend to be big (and you don't want to decrease `max_relay_log_size`) and you need to put them in some area different from the data directory, or if you want to increase speed by balancing load between disks.

- `--relay-log-index=file_name`

  The location and name that should be used for the relay log index file. The default name is *`host_name`*`-relay-bin.index`, where *`host_name`* is the name of the slave server.

- `--relay-log-info-file=file_name`

The name to use for the file in which the slave records information about the relay logs. The default name is `relay-log.info` in the data directory.

- `--relay-log-purge={0|1}`

  Disables or enables automatic purging of relay logs as soon as they are not needed any more. The default value is 1 (enabled). This is a global variable that can be changed dynamically with `SET GLOBAL relay_log_purge`.

  This option is available as of MySQL 4.1.1.

- `--relay-log-space-limit=#`

  Places an upper limit on the total size of all relay logs on the slave (a value of 0 means ``unlimited''). This is useful for a slave server host that has limited disk space. When the limit is reached, the I/O thread stops reading binary log events from the master server until the SQL thread has caught up and deleted some unused relay logs. Note that this limit is not absolute: There are cases where the SQL thread needs more events before it can delete relay logs. In that case, the I/O thread exceeds the limit until it becomes possible for the SQL thread to delete some relay logs. Not doing so would cause a deadlock (which is what happens before MySQL 4.0.13). You should not set `--relay-log-space-limit` to less than twice the value of `--max-relay-log-size` (or `--max-binlog-size` if `--max-relay-log-size` is 0). In that case, there is a chance that the I/O thread waits for free space because `--relay-log-space-limit` is exceeded, but the SQL thread has no relay log to purge and is unable to satisfy the I/O thread. This forces the I/O thread to temporarily ignore `--relay-log-space-limit`.

- `--replicate-do-db=`*`db_name`*

  Tells the slave to restrict replication to statements where the default database (that is, the one selected by `USE`) is *`db_name`*. To specify more than one database, use this option multiple times, once for each database. Note that this does not replicate cross-database statements such as `UPDATE` *`some_db.some_table`* `SET foo='bar'` while having selected a different database or no database. If you need cross-database updates to work, make sure that you have MySQL 3.23.28 or later, and use `--replicate-wild-do-table=`*`db_name`*`.%`. Please read the notes that follow this option list.

  An example of what does not work as you might expect: If the slave is started with `--replicate-do-db=sales` and you issue the following statements on the master, the `UPDATE` statement is not replicated:

  ```
  USE prices;
  UPDATE sales.january SET amount=amount+1000;
  ```

  If you need cross-database updates to work, use `--replicate-wild-do-table=`*`db_name`*`.%` instead.

  The main reason for this ``just-check-the-default-database'' behavior is that it's difficult from the statement alone to know whether or not it should be replicated (for example, if you are using multiple-table `DELETE` or multiple-table `UPDATE` statements that go across multiple databases). It's also very fast to just check the default database.

- `--replicate-do-table=`*`db_name.tbl_name`*

  Tells the slave thread to restrict replication to the specified table. To specify more than one table, use this option multiple times, once for each table. This works for cross-database updates, in contrast to `--replicate-do-db`. Please read the notes that follow this option list.

- `--replicate-ignore-db=db_name`

  Tells the slave to not replicate any statement where the default database (that is, the one selected by `USE`) is *`db_name`*. To specify more than one database to ignore, use this option multiple times, once for each database. You should not use this option if you are using cross-database updates and you don't want these updates to be replicated. Please read the notes that follow this option list.

  An example of what does not work as you might expect: If the slave is started with `--replicate-ignore-db=sales` and you issue the following statements on the master, the `UPDATE` statement is not replicated:

  ```
  USE prices;
  UPDATE sales.january SET amount=amount+1000;
  ```

  If you need cross-database updates to work, use `--replicate-wild-ignore-table=db_name.%` instead.

- `--replicate-ignore-table=db_name.tbl_name`

  Tells the slave thread to not replicate any statement that updates the specified table (even if any other tables might be updated by the same statement). To specify more than one table to ignore, use this option multiple times, once for each table. This works for cross-database updates, in contrast to `--replicate-ignore-db`. Please read the notes that follow this option list.

- `--replicate-wild-do-table=db_name.tbl_name`

  Tells the slave thread to restrict replication to statements where any of the updated tables match the specified database and table name patterns. Patterns can contain the '%' and '_' wildcard characters, which have the same meaning as for the `LIKE` pattern-matching operator. To specify more than one table, use this option multiple times, once for each table. This works for cross-database updates. Please read the notes that follow this option list.

  Example: `--replicate-wild-do-table=foo%.bar%` replicates only updates that use a table where the database name starts with `foo` and the table name starts with `bar`.

  If the table name pattern is `%`, it matches any table name and the option also applies to database-level statements (`CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE`). For example, if you use `--replicate-wild-do-table=foo%.%`, database-level statements are replicated if the database name matches the pattern `foo%`.

  To include literal wildcard characters in the database or table name patterns, escape them with a backslash. For example, to replicate all tables of a database that is named `my_own%db`, but not replicate tables from the `my1ownAABCdb` database, you should escape the '_' and '%' characters like this: `--replicate-wild-do-table=my\_own\%db`. If you're using the option on the command line, you might need to double the backslashes or quote the option value, depending on your command interpreter. For example, with the **bash** shell, you would need to type `--replicate-wild-do-table=my\\_own\\%db`.

- `--replicate-wild-ignore-table=db_name.tbl_name`

  Tells the slave thread to not replicate a statement where any table matches the given wildcard pattern. To specify more than one table to ignore, use this option multiple times, once for each table. This works for cross-database updates. Please read the notes that follow this option list.

  Example: `--replicate-wild-ignore-table=foo%.bar%` does not replicate updates that use a table where the database name starts with `foo` and the table name starts with `bar`.

For information about how matching works, see the description of the `--replicate-wild-do-table` option. The rules for including literal wildcard characters in the option value are the same as for `--replicate-wild-ignore-table` as well.

- `--replicate-rewrite-db=from_name->to_name`

  Tells the slave to translate the default database (that is, the one selected by `USE`) to *`to_name`* if it was *`from_name`* on the master. Only statements involving tables are affected (not statements such as `CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE`), and only if *`from_name`* was the default database on the master. This does not work for cross-database updates. Note that the database name translation is done before `--replicate-*` rules are tested.

  If you use this option on the command line and the '`>`' character is special to your command interpreter, quote the option value. For example:

  ```
  shell> mysqld --replicate-rewrite-db="olddb->newdb"
  ```

- `--replicate-same-server-id`

  To be used on slave servers. Usually you can should the default setting of 0, to prevent infinite loops in circular replication. If set to 1, this slave does not skip events having its own server id; normally this is useful only in rare configurations. Cannot be set to 1 if `--log-slave-updates` is used. Be careful that starting from MySQL 4.1, by default the slave I/O thread does not even write binary log events to the relay log if they have the slave's server id (this optimization helps save disk usage compared to 4.0). So if you want to use `--replicate-same-server-id` in 4.1 versions, be sure to start the slave with this option before you make the slave read its own events which you want the slave SQL thread to execute.

- `--report-host=host`

  The hostname or IP number of the slave to be reported to the master during slave registration. This value appears in the output of `SHOW SLAVE HOSTS` on the master server. Leave the value unset if you do not want the slave to register itself with the master. Note that it is not sufficient for the master to simply read the IP number of the slave from the TCP/IP socket after the slave connects. Due to `NAT` and other routing issues, that IP may not be valid for connecting to the slave from the master or other hosts.

  This option is available as of MySQL 4.0.0.

- `--report-port=port_number`

  The TCP/IP port for connecting to the slave, to be reported to the master during slave registration. Set it only if the slave is listening on a non-default port or if you have a special tunnel from the master or other clients to the slave. If you are not sure, leave this option unset.

  This option is available as of MySQL 4.0.0.

- `--skip-slave-start`

  Tells the slave server not to start the slave threads when the server starts. To start the threads later, use a `START SLAVE` statement.

- `--slave_compressed_protocol={0|1}`

If this option is set to 1, use compression of the slave/master protocol if both the slave and the master support it.

- `--slave-load-tmpdir=file_name`

  The name of the directory where the slave creates temporary files. This option is by default equal to the value of the `tmpdir` system variable. When the slave SQL thread replicates a `LOAD DATA INFILE` statement, it extracts the to-be-loaded file from the relay log into temporary files, then loads these into the table. If the file loaded on the master was huge, the temporary files on the slave are huge, too. Therefore, it might be advisable to use this option to tell the slave to put temporary files in a directory located in some filesystem that has a lot of available space. In that case, you may also use the `--relay-log` option to place the relay logs in that filesystem, because the relay logs are huge as well. `--slave-load-tmpdir` should point to a disk-based filesystem, not a memory-based one: The slave needs the temporary files used to replicate `LOAD DATA INFILE` to survive a machine's restart. The directory also should not be one that is cleared by the operating system during the system startup process.

- `--slave-net-timeout=seconds`

  The number of seconds to wait for more data from the master before aborting the read, considering the connection broken, and trying to reconnect. The first retry occurs immediately after the timeout. The interval between retries is controlled by the `--master-connect-retry` option.

- `--slave-skip-errors= [err_code1,err_code2,... | all]`

  Normally, replication stops when an error occurs, which gives you the opportunity to resolve the inconsistency in the data manually. This option tells the slave SQL thread to continue replication when a statement returns any of the errors listed in the option value.

  Do not use this option unless you fully understand why you are getting the errors. If there are no bugs in your replication setup and client programs, and no bugs in MySQL itself, an error that stops replication should never occur. Indiscriminate use of this option results in slaves becoming hopelessly out of sync with the master, and you have no idea why.

  For error codes, you should use the numbers provided by the error message in your slave error log and in the output of `SHOW SLAVE STATUS`. The server error codes are listed in Chapter 24, *Error Handling in MySQL*.

  You can (but should not) also use the very non-recommended value of `all` which ignores all error messages and keeps barging along regardless of what happens. Needless to say, if you use it, we make no promises regarding your data integrity. Please do not complain if your data on the slave is not anywhere close to what it is on the master in this case. You have been warned.

  Examples:

```
--slave-skip-errors=1062,1053
--slave-skip-errors=all
```

The `--replicate-*` rules are evaluated as follows to determine whether a statement is executed by the slave or ignored:

1. Are there some `--replicate-do-db` or `--replicate-ignore-db` rules?

   - Yes: Test them as for `--binlog-do-db` and `--binlog-ignore-db` (see Section 5.9.4,

"The Binary Log"). What is the result of the test?

- Ignore the statement: Ignore it and exit.
- Execute the statement: Don't execute it immediately, defer the decision, go to the next step.

- No: Go to the next step.

2. Are there some `--replicate-*-table` rules?
   - No: Execute the query and exit.
   - Yes: Go to the next step. Only tables that are to be updated are compared to the rules (`INSERT INTO sales SELECT * FROM prices`: only `sales` are compared to the rules). If several tables are to be updated (multiple-table statement), the first matching table (matching ``do'' or ``ignore'') wins. That is, the first table is compared to the rules. Then, if no decision could be mad, the second table is compared to the rules, and so forth.
3. Are there some `--replicate-do-table` rules?
   - Yes: Does the table match any of them?
     - Yes: Execute the query and exit.
     - No: Go to the next step.
   - No: Go to the next step.
4. Are there some `--replicate-ignore-table` rules?
   - Yes: Does the table match any of them?
     - Yes: Ignore the query and exit.
     - No: Go to the next step.
   - No: Go to the next step.
5. Are there some `--replicate-wild-do-table` rules?
   - Yes: Does the table match any of them?
     - Yes: Execute the query and exit.
     - No: Go to the next step.
   - No: Go to the next step.
6. Are there some `--replicate-wild-ignore-table` rules?
   - Yes: Does the table match any of them?
     - Yes: Ignore the query and exit.
     - No: Go to the next step.
   - No: Go to the next step.
7. No `--replicate-*-table` rule was matched. Is there another table to test against these rules?

- Yes: Loop.
- No: We have tested all tables to be updated and could not match any rule. Are there `--replicate-do-table` or `--replicate-wild-do-table` rules?
  - Yes: Ignore the query and exit.
  - No: Execute the query and exit.

6.9. Replication FAQ

Q: How do I configure a slave if the master is running and I do not want to stop it?

A: There are several options. If you have taken a backup of the master at some point and recorded the binary log name and offset (from the output of `SHOW MASTER STATUS`) corresponding to the snapshot, use the following procedure:

1. Make sure that the slave is assigned a unique server ID.
2. Execute the following statement on the slave, filling in appropriate values for each option:

```
mysql> CHANGE MASTER TO
    ->     MASTER_HOST='master_host_name',
    ->     MASTER_USER='master_user_name',
    ->     MASTER_PASSWORD='master_pass',
    ->     MASTER_LOG_FILE='recorded_log_file_name',
    ->     MASTER_LOG_POS=recorded_log_position;
```

3. Execute `START SLAVE` on the slave.

If you do not have a backup of the master server, here is a quick procedure for creating one. All steps should be performed on the master host.

1. Issue this statement:

```
mysql> FLUSH TABLES WITH READ LOCK;
```

2. With the lock still in place, execute this command (or a variation of it):

```
shell> tar zcf /tmp/backup.tar.gz /var/lib/mysql
```

3. Issue this statement and make sure to record the output, which you need later:

```
mysql> SHOW MASTER STATUS;
```

4. Release the lock:

```
mysql> UNLOCK TABLES;
```

An alternative is to make an SQL dump of the master instead of a binary copy as in the preceding procedure. To do this, you can use **mysqldump --master-data** on your master and later load the SQL dump into your slave. However, this is slower than making a binary copy.

No matter which of the two methods you use, afterward follow the instructions for the case when you have a snapshot and have recorded the log name and offset. You can use the same snapshot to set up several slaves. Once you have the snapshot of the master, you can wait to set up a slave as long as the binary logs of the master are left intact. The two practical limitations on the length of time you can wait are the amount of disk space available to retain binary logs on the master and the length of time it takes the slave to catch up.

You can also use `LOAD DATA FROM MASTER`. This is a convenient statement that transfers a snapshot to the slave and adjusts the log name and offset all at once. In the future, `LOAD DATA FROM MASTER` will be the recommended way to set up a slave. Be warned, however, that it works only for `MyISAM` tables and it may hold a read lock for a long time. It is not yet implemented as efficiently as we would like. If you have large tables, the preferred method at this time is still to make a binary snapshot on the master server after executing `FLUSH TABLES WITH READ LOCK`.

Q: Does the slave need to be connected to the master all the time?

A: No, it does not. The slave can go down or stay disconnected for hours or even days, then reconnect and catch up on the updates. For example, you can set up a master/slave relationship over a dial-up link where the link is up only sporadically and for short periods of time. The implication of this is that, at any given time, the slave is not guaranteed to be in sync with the master unless you take some special measures. In the future, we will have the option to block the master until at least one slave is in sync.

Q: How do I know how late a slave is compared to the master? In other words, how do I know the date of the last query replicated by the slave?

A: If the slave is 4.1.1 or newer, read the `Seconds_Behind_Master` column in `SHOW SLAVE STATUS`. For older versions, the following applies. This is possible only if `SHOW SLAVE STATUS` on the slave shows that the SQL thread is running (or for MySQL 3.23, that the slave thread is running), and that the thread has executed at least one event from the master. See Section 6.3, “Replication Implementation Details”.

When the slave SQL thread executes an event read from the master, it modifies its own time to the event timestamp (this is why `TIMESTAMP` is well replicated). In the `Time` column in the output of `SHOW PROCESSLIST`, the number of seconds displayed for the slave SQL thread is the number of seconds between the timestamp of the last replicated event and the real time of the slave machine. You can use this to determine the date of the last replicated event. Note that if your slave has been disconnected from the master for one hour, and then reconnects, you may immediately see `Time` values like 3600 for the slave SQL thread in `SHOW PROCESSLIST`. This would be because the slave is executing statements that are one hour old.

Q: How do I force the master to block updates until the slave catches up?

A: Use the following procedure:

1. On the master, execute these statements:

   ```
   mysql> FLUSH TABLES WITH READ LOCK;
   mysql> SHOW MASTER STATUS;
   ```

   Record the log name and the offset from the output of the `SHOW` statement. These are the replication coordinates.

2. On the slave, issue the following statement, where the arguments to the `MASTER_POS_WAIT()`

function are the replication coordinate values obtained in the previous step:

```
mysql> SELECT MASTER_POS_WAIT('log_name', log_offset);
```

The `SELECT` statement blocks until the slave reaches the specified log file and offset. At that point, the slave is in sync with the master and the statement returns.

3. On the master, issue the following statement to allow the master to begin processing updates again:

```
mysql> UNLOCK TABLES;
```

Q: What issues should I be aware of when setting up two-way replication?

A: MySQL replication currently does not support any locking protocol between master and slave to guarantee the atomicity of a distributed (cross-server) update. In other words, it is possible for client A to make an update to co-master 1, and in the meantime, before it propagates to co-master 2, client B could make an update to co-master 2 that makes the update of client A work differently than it did on co-master 1. Thus, when the update of client A makes it to co-master 2, it produces tables that are different than what you have on co-master 1, even after all the updates from co-master 2 have also propagated. This means that you should not co-chain two servers in a two-way replication relationship unless you are sure that your updates can safely happen in any order, or unless you take care of mis-ordered updates somehow in the client code.

You must also realize that two-way replication actually does not improve performance very much (if at all), as far as updates are concerned. Both servers need to do the same number of updates each, as you would have one server do. The only difference is that there is a little less lock contention, because the updates originating on another server are serialized in one slave thread. Even this benefit might be offset by network delays.

Q: How can I use replication to improve performance of my system?

A: You should set up one server as the master and direct all writes to it. Then configure as many slaves as you have the budget and rackspace for, and distribute the reads among the master and the slaves. You can also start the slaves with the `--skip-innodb`, `--skip-bdb`, `--low-priority-updates`, and `--delay-key-write=ALL` options to get speed improvements on the slave end. In this case, the slave uses non-transactional `MyISAM` tables instead of `InnoDB` and `BDB` tables to get more speed.

Q: What should I do to prepare client code in my own applications to use performance-enhancing replication?

A: If the part of your code that is responsible for database access has been properly abstracted/modularized, converting it to run with a replicated setup should be very smooth and easy. Just change the implementation of your database access to send all writes to the master, and to send reads to either the master or a slave. If your code does not have this level of abstraction, setting up a replicated system gives you the opportunity and motivation to it clean up. You should start by creating a wrapper library or module with the following functions:

- `safe_writer_connect()`
- `safe_reader_connect()`
- `safe_reader_statement()`
- `safe_writer_statement()`

`safe_` in each function name means that the function takes care of handling all the error conditions. You can use different names for the functions. The important thing is to have a unified interface for connecting for reads, connecting for writes, doing a read, and doing a write.

You should then convert your client code to use the wrapper library. This may be a painful and scary process at first, but it pays off in the long run. All applications that use the approach just described are able to take advantage of a master/slave configuration, even one involving multiple slaves. The code is a lot easier to maintain, and adding troubleshooting options is trivial. You just need to modify one or two functions; for example, to log how long each statement took, or which statement among your many thousands gave you an error.

If you have written a lot of code, you may want to automate the conversion task by using the **replace** utility that comes with standard MySQL distributions, or just write your own conversion script. Ideally, your code uses consistent programming style conventions. If not, then you are probably better off rewriting it anyway, or at least going through and manually regularizing it to use a consistent style.

Q: When and how much can MySQL replication improve the performance of my system?

A: MySQL replication is most beneficial for a system with frequent reads and infrequent writes. In theory, by using a single-master/multiple-slave setup, you can scale the system by adding more slaves until you either run out of network bandwidth, or your update load grows to the point that the master cannot handle it.

In order to determine how many slaves you can get before the added benefits begin to level out, and how much you can improve performance of your site, you need to know your query patterns, and to determine empirically by benchmarking the relationship between the throughput for reads (reads per second, or `max_reads`) and for writes (`max_writes`) on a typical master and a typical slave. The example here shows a rather simplified calculation of what you can get with replication for a hypothetical system.

Let's say that system load consists of 10% writes and 90% reads, and we have determined by benchmarking that `max_reads` is 1200 # 2 `*` `max_writes`. In other words, the system can do 1,200 reads per second with no writes, the average write is twice as slow as the average read, and the relationship is linear. Let us suppose that the master and each slave have the same capacity, and that we have one master and N slaves. Then we have for each server (master or slave):

```
reads = 1200 - 2 * writes
```

`reads = 9 * writes / (N + 1)` (reads are split, but writes go to all servers)

```
9 * writes / (N + 1) + 2 * writes = 1200
```

```
writes = 1200 / (2 + 9/(N+1))
```

The last equation indicates that the maximum number of writes for N slaves, given a maximum possible read rate of 1,200 per minute and a ratio of nine reads per write.

This analysis yields the following conclusions:

- If N = 0 (which means we have no replication), our system can handle about 1200/11 = 109 writes per second.
- If N = 1, we get up to 184 writes per second.
- If N = 8, we get up to 400 writes per second.
- If N = 17, we get up to 480 writes per second.
- Eventually, as N approaches infinity (and our budget negative infinity), we can get very close to 600

writes per second, increasing system throughput about 5.5 times. However, with only eight servers, we increased it almost four times.

Note that these computations assume infinite network bandwidth and neglect several other factors that could turn out to be significant on your system. In many cases, you may not be able to perform a computation similar to the one just shown that accurately predicts what will happen on your system if you add N replication slaves. However, answering the following questions should help you decide whether and how much replication will improve the performance of your system:

- What is the read/write ratio on your system?
- How much more write load can one server handle if you reduce the reads?
- For how many slaves do you have bandwidth available on your network?

Q: How can I use replication to provide redundancy/high availability?

A: With the currently available features, you would have to set up a master and a slave (or several slaves), and write a script that monitors the master to see whether it is up. Then instruct your applications and the slaves to change master in case of failure. Some suggestions:

- To tell a slave to change its master, use the `CHANGE MASTER TO` statement.
- A good way to keep your applications informed as to the location of the master is by having a dynamic DNS entry for the master. With `bind` you can use `nsupdate` to dynamically update your DNS.
- You should run your slaves with the `--log-bin` option and without `--log-slave-updates`. This way the slave is ready to become a master as soon as you issue `STOP SLAVE`; `RESET MASTER`, and `CHANGE MASTER TO` on the other slaves. For example, assume that you have the following setup:

```
        WC
         \
          v
 WC----> M
        / | \
       /  |  \
      v   v   v
     S1   S2  S3
```

  M means the master, S the slaves, WC the clients that issue database writes and reads; clients that issue only database reads are not represented, because they need not switch. S1, S2, and S3 are slaves running with `--log-bin` and without `--log-slave-updates`. Because updates received by a slave from the master are not logged in the binary log unless `--log-slave-updates` is specified, the binary log on each slave is empty. If for some reason M becomes unavailable, you can pick one slave to become the new master. For example, if you pick S1, all WC should be redirected to S1, and S2 and S3 should replicate from S1.

  Make sure that all slaves have processed any statements in their relay log. On each slave, issue `STOP SLAVE IO_THREAD`, then check the output of `SHOW PROCESSLIST` until you see `Has read all relay log`. When this is true for all slaves, they can be reconfigured to the new setup. On the slave S1 being promoted to become the master, issue `STOP SLAVE` and `RESET MASTER`.

On the other slaves S2 and S3, use `STOP SLAVE` and `CHANGE MASTER TO MASTER_HOST='S1'` (where `'S1'` represents the real hostname of S1). To `CHANGE MASTER`, add all information about how to connect to S1 from S2 or S3 (user, password, port). In `CHANGE MASTER`, there is no need to specify the name of S1's binary log or binary log position to read from: We know it is the first binary log and position 4, which are the defaults for `CHANGE MASTER`. Finally, use `START SLAVE` on S2 and S3.

Then instruct all WC to direct their statements to S1. From that point on, all updates statements sent by WC to S1 are written to the binary log of S1, which contains every update statement sent to S1 since M died.

The result is this configuration:

```
       WC
      /
      |
 WC   |  M(unavailable)
  \   |
   \  |
    v v
     S1<--S2  S3
      ^       |
      +-------+
```

When M is up again, you just have to issue on it the same `CHANGE MASTER` as the one issued on S2 and S3, so that M becomes a slave of S1 and picks all the WC writes it has missed while it was down. To make M a master again (because it is the most powerful machine, for example), use the preceding procedure as if S1 was unavailable and M was to be the new master. During the procedure, don't forget to run `RESET MASTER` on M before making S1, S2, and S3 slaves of M. Otherwise, they may pick up old WC writes from before the point at which M became unavailable.

We are currently working on integrating an automatic master election system into MySQL, but until it is ready, you have to create your own monitoring tools.

6.10. Troubleshooting Replication

If you have followed the instructions, and your replication setup is not working, first check the following:

- **Check the error log for messages**. Many users have lost time by not doing this early enough.
- Is the master logging to the binary log? Check with `SHOW MASTER STATUS`. If it is, `Position` is non-zero. If not, verify that you are running the master with the `log-bin` and `server-id` options.
- Is the slave running? Use `SHOW SLAVE STATUS` to check whether the `Slave_IO_Running` and `Slave_SQL_Running` values are both `Yes`. If not, verify the options that were used when starting the slave server.
- If the slave is running, did it establish a connection to the master? Use `SHOW PROCESSLIST`, find the I/O and SQL threads and check their `State` column to see how they display. See Section 6.3, “Replication Implementation Details”. If the I/O thread state says `Connecting to master`, verify the privileges for the replication user on the master, master hostname, your DNS setup, whether the master is actually running, and whether it is reachable from the slave.

- If the slave was running before but has stopped, the reason usually is that some statement that succeeded on the master failed on the slave. This should never happen if you have taken a proper snapshot of the master, and never modify the data on the slave outside of the slave thread. If it does, it is a bug or you have encountered one of the known replication limitations described in Section 6.7, “Replication Features and Known Problems”. If it is a bug, see Section 6.11, “Reporting Replication Bugs” for instructions on how to report it.

- If a statement that succeeded on the master refuses to run on the slave, and it is not feasible to do a full database resynchronization (that is, to delete the slave's database and copy a new snapshot from the master), try the following:

  1. Determine whether the slave's table is different from the master's. Try to understand how this happened. Then make the slave's table identical to the master's and run `START SLAVE`.

  2. If the preceding step does not work or does not apply, try to understand whether it would be safe to make the update manually (if needed) and then ignore the next statement from the master.

  3. If you decide that you can skip the next statement from the master, issue the following statements:

     ```
     mysql> SET GLOBAL SQL_SLAVE_SKIP_COUNTER = n;
     mysql> START SLAVE;
     ```

     The value of `n` should be 1 if the next statement from the master does not use `AUTO_INCREMENT` or `LAST_INSERT_ID()`. Otherwise, the value should be 2. The reason for using a value of 2 for statements that use `AUTO_INCREMENT` or `LAST_INSERT_ID()` is that they take two events in the binary log of the master.

  4. If you are sure that the slave started out perfectly synchronized with the master, and no one has updated the tables involved outside of slave thread, then presumably the discrepancy is the result of a bug. If you are running the most recent version, please report the problem. If you are running an older version of MySQL, try upgrading.

6.11. Reporting Replication Bugs

When you have determined that there is no user error involved, and replication still either does not work at all or is unstable, it is time to send us a bug report. We need to get as much information as possible from you to be able to track down the bug. Please do spend some time and effort preparing a good bug report.

If you have a repeatable test case that demonstrates the bug, please enter it into our bugs database at http://bugs.mysql.com/. If you have a phantom problem (one that you cannot duplicate ``at will''), use the following procedure:

1. Verify that no user error is involved. For example, if you update the slave outside of the slave thread, the data goes out of sync, and you can have unique key violations on updates. In this case, the slave thread stops and waits for you to clean up the tables manually to bring them in sync. This is not a replication problem. It is a problem of outside interference that causes replication to fail.

2. Run the slave with the `--log-slave-updates` and `--log-bin` options. They cause the slave to log the updates that it receives from the master into its own binary logs.

3. Save all evidence before resetting the replication state. If we have no information or only sketchy information, it becomes difficult or impossible for us to track down the problem. The evidence you

should collect is:

- All binary logs from the master
- All binary logs from the slave
- The output of `SHOW MASTER STATUS` from the master at the time you have discovered the problem
- The output of `SHOW SLAVE STATUS` from the master at the time you have discovered the problem
- Error logs from the master and the slave

4. Use **mysqlbinlog** to examine the binary logs. The following should be helpful to find the trouble query, for example:

```
shell> mysqlbinlog -j pos_from_slave_status \
           /path/to/log_from_slave_status | head
```

Once you have collected the evidence for the phantom problem, try hard to isolate it into a separate test case first. Then enter the problem into our bugs database at http://bugs.mysql.com/ with as much information as possible.

Chapter 7. MySQL Optimization

Optimization is a complex task because ultimately it requires understanding of the entire system to be optimized. Although it may be possible to perform some local optimizations with little knowledge of your system or application, the more optimal you want your system to become, the more you have to know about it.

This chapter tries to explain and give some examples of different ways to optimize MySQL. Remember, however, that there are always additional ways to make the system even faster, although they may require increasing effort to achieve.

7.1. Optimization Overview

The most important factor in making a system fast is its basic design. You also need to know what kinds of things your system is doing, and what your bottlenecks are.

The most common system bottlenecks are:

- Disk seeks. It takes time for the disk to find a piece of data. With modern disks, the mean time for this is usually lower than 10ms, so we can in theory do about 100 seeks a second. This time improves slowly with new disks and is very hard to optimize for a single table. The way to optimize seek time is to distribute the data onto more than one disk.
- Disk reading and writing. When the disk is at the correct position, we need to read the data. With modern disks, one disk delivers at least 10-20MB/s throughput. This is easier to optimize than seeks because you can read in parallel from multiple disks.
- CPU cycles. When we have the data in main memory, we need to process it to get our result. Having small tables compared to the amount of memory is the most common limiting factor. But with small tables, speed is usually not the problem.
- Memory bandwidth. When the CPU needs more data than can fit in the CPU cache, main memory bandwidth becomes a bottleneck. This is an uncommon bottleneck for most systems, but one to be aware of.

7.1.1. MySQL Design Limitations and Tradeoffs

When using the `MyISAM` storage engine, MySQL uses extremely fast table locking that allows multiple readers or a single writer. The biggest problem with this storage engine occurs when you have a steady stream of mixed updates and slow selects on a single table. If this is a problem for certain tables, you can use another storage engine for them. See Chapter 14, *MySQL Storage Engines and Table Types*.

MySQL can work with both transactional and non-transactional tables. To be able to work smoothly with non-transactional tables (which can't roll back if something goes wrong), MySQL has the following rules (when not running in strict mode or if you use the `IGNORE` specifier to `INSERT` or `UPDATE`).

- All columns have default values.
- If you insert an ``incorrect'' value in a column, such as a too-large numeric value into a numeric column, MySQL sets the column to the ``best possible value'' instead of giving an error. For numerical values, this is 0, the smallest possible value or the largest possible value. For strings, this is either the empty string or the longest possible string that can be stored in the column.

- All calculated expressions return a value that can be used instead of signaling an error condition. For example, 1/0 returns `NULL`. (This behavior can be changed by using the `ERROR_FOR_DIVISION_BY_ZERO` SQL mode).

If you are using non-transactional tables, you should not use MySQL to check column content. In general, the safest (and often fastest) way is to let the application ensure that it passes only legal values to the database.

For more information about this, see Section 1.5.6, “How MySQL Deals with Constraints” and Section 13.1.4, “`INSERT` Syntax” or Section 5.2.2, “The Server SQL Mode”.

7.1.2. Designing Applications for Portability

Because all SQL servers implement different parts of standard SQL, it takes work to write portable SQL applications. It is very easy to achieve portability for very simple selects and inserts, but becomes more difficult the more capabilities you require. If you want an application that is fast with many database systems, it becomes even harder!

To make a complex application portable, you need to determine which SQL servers it must work with, then determine what features those servers support.

All database systems have some weak points. That is, they have different design compromises that lead to different behavior.

You can use the MySQL `crash-me` program to find functions, types, and limits that you can use with a selection of database servers. `crash-me` does not check for every possible feature, but it is still reasonably comprehensive, performing about 450 tests.

An example of the type of information `crash-me` can provide is that you shouldn't have column names longer than 18 characters if you want to be able to use Informix or DB2.

The `crash-me` program and the MySQL benchmarks are all very database independent. By taking a look at how they are written, you can get a feeling for what you have to do to make your own applications database independent. The programs can be found in the `sql-bench` directory of MySQL source distributions. They are written in Perl and use the DBI database interface. Use of DBI in itself solves part of the portability problem because it provides database-independent access methods.

For `crash-me` results, visit http://dev.mysql.com/tech-resources/crash-me.php. See http://dev.mysql.com/tech-resources/benchmarks/ for the results from the benchmarks.

If you strive for database independence, you need to get a good feeling for each SQL server's bottlenecks. For example, MySQL is very fast in retrieving and updating records for `MyISAM` tables, but has a problem in mixing slow readers and writers on the same table. Oracle, on the other hand, has a big problem when you try to access rows that you have recently updated (until they are flushed to disk). Transactional databases in general are not very good at generating summary tables from log tables, because in this case row locking is almost useless.

To make your application *really* database independent, you need to define an easily extendable interface through which you manipulate your data. As C++ is available on most systems, it makes sense to use a C++ class-based interface to the databases.

If you use some feature that is specific to a given database system (such as the `REPLACE` statement, which is specific to MySQL), you should implement the same feature for other SQL servers by coding an alternative method. Although the alternative may be slower, it allows the other servers to perform the same tasks.

With MySQL, you can use the `/*! */` syntax to add MySQL-specific keywords to a query. The code

inside `/**/` is treated as a comment (and ignored) by most other SQL servers.

If high performance is more important than exactness, as in some Web applications, it is possible to create an application layer that caches all results to give you even higher performance. By letting old results ``expire'' after a while, you can keep the cache reasonably fresh. This provides a method to handle high load spikes, in which case you can dynamically increase the cache and set the expiration timeout higher until things get back to normal.

In this case, the table creation information should contain information of the initial size of the cache and how often the table should normally be refreshed.

An alternative to implementing an application cache is to use the MySQL query cache. By enabling the query cache, the server handles the details of determining whether a query result can be reused. This simplifies your application. See Section 5.11, “The MySQL Query Cache”.

7.1.3. What We Have Used MySQL For

This section describes an early application for MySQL.

During MySQL initial development, the features of MySQL were made to fit our largest customer, which handled data warehousing for a couple of the largest retailers in Sweden.

From all stores, we got weekly summaries of all bonus card transactions, and were expected to provide useful information for the store owners to help them find how their advertising campaigns were affecting their own customers.

The volume of data was quite huge (about seven million summary transactions per month), and we had data for 4-10 years that we needed to present to the users. We got weekly requests from our customers, who wanted to get ``instant'' access to new reports from this data.

We solved this problem by storing all information per month in compressed ``transaction'' tables. We had a set of simple macros that generated summary tables grouped by different criteria (product group, customer id, store, and so on) from the tables in which the transactions were stored. The reports were Web pages that were dynamically generated by a small Perl script. This script parsed a Web page, executed the SQL statements in it, and inserted the results. We would have used PHP or mod_perl instead, but they were not available at the time.

For graphical data, we wrote a simple tool in C that could process SQL query results and produce GIF images based on those results. This tool also was dynamically executed from the Perl script that parses the Web pages.

In most cases, a new report could be created simply by copying an existing script and modifying the SQL query in it. In some cases, we needed to add more columns to an existing summary table or generate a new one. This also was quite simple because we kept all transaction-storage tables on disk. (This amounted to about 50GB of transaction tables and 200GB of other customer data.)

We also let our customers access the summary tables directly with ODBC so that the advanced users could experiment with the data themselves.

This system worked well and we had no problems handling the data with quite modest Sun Ultra SPARCstation hardware (2x200MHz). Eventually the system was migrated to Linux.

7.1.4. The MySQL Benchmark Suite

This section should contain a technical description of the MySQL benchmark suite (and `crash-me`), but that description has not yet been written. Currently, you can get a good idea of the benchmarks by looking at the code and results in the `sql-bench` directory in any MySQL source distribution.

This benchmark suite is meant to tell any user what operations a given SQL implementation performs well or poorly.

Note that this benchmark is single-threaded, so it measures the minimum time for the operations performed. We plan to add multi-threaded tests to the benchmark suite in the future.

To use the benchmark suite, the following requirements must be satisfied:

- The benchmark suite is provided with MySQL source distributions. You can either download a released distribution from http://dev.mysql.com/downloads/, or use the current development source tree (see Section 2.8.3, “Installing from the Development Source Tree”).
- The benchmark scripts are written in Perl and use the Perl DBI module to access database servers, so DBI must be installed. You also need the server-specific DBD drivers for each of the servers you want to test. For example, to test MySQL, PostgreSQL, and DB2, you must have the `DBD::mysql`, `DBD::Pg`, and `DBD::DB2` modules installed. See Section 2.13, “Perl Installation Notes”.

After you obtain a MySQL source distribution, you can find the benchmark suite located in its `sql-bench` directory. To run the benchmark tests, build MySQL, then change location into the `sql-bench` directory and execute the `run-all-tests` script:

```
shell> cd sql-bench
shell> perl run-all-tests --server=server_name
```

`server_name` is one of the supported servers. To get a list of all options and supported servers, invoke this command:

```
shell> perl run-all-tests --help
```

The `crash-me` script also is located in the `sql-bench` directory. `crash-me` tries to determine what features a database supports and what its capabilities and limitations are by actually running queries. For example, it determines:

- What column types are supported
- How many indexes are supported
- What functions are supported
- How big a query can be
- How big a `VARCHAR` column can be

You can find the results from `crash-me` for many different database servers at http://dev.mysql.com/tech-resources/crash-me.php. For more information about benchmark results, visit http://dev.mysql.com/tech-resources/benchmarks/.

7.1.5. Using Your Own Benchmarks

You should definitely benchmark your application and database to find out where the bottlenecks are. By fixing a bottleneck (or by replacing it with a ``dummy module''), you can then easily identify the next bottleneck. Even if the overall performance for your application currently is acceptable, you should at least make a plan for each bottleneck, and decide how to solve it if someday you really need the extra

performance.

For an example of portable benchmark programs, look at the MySQL benchmark suite. See Section 7.1.4, “The MySQL Benchmark Suite”. You can take any program from this suite and modify it for your needs. By doing this, you can try different solutions to your problem and test which really is fastest for you.

Another free benchmark suite is the Open Source Database Benchmark, available at http://osdb.sourceforge.net/.

It is very common for a problem to occur only when the system is very heavily loaded. We have had many customers who contact us when they have a (tested) system in production and have encountered load problems. In most cases, performance problems turn out to be due to issues of basic database design (for example, table scans are *not good* at high load) or problems with the operating system or libraries. Most of the time, these problems would be a *lot* easier to fix if the systems were not in production.

To avoid problems like this, you should put some effort into benchmarking your whole application under the worst possible load! You can use Super Smack for this. It is available at http://jeremy.zawodny.com/mysql/super-smack/. As the name suggests, it can bring a system to its knees if you ask it, so make sure to use it only on your development systems.

7.2. Optimizing `SELECT` Statements and Other Queries

First, one factor affects all statements: The more complex your permission setup is, the more overhead you have.

Using simpler permissions when you issue `GRANT` statements enables MySQL to reduce permission-checking overhead when clients execute statements. For example, if you don't grant any table-level or column-level privileges, the server need not ever check the contents of the `tables_priv` and `columns_priv` tables. Similarly, if you place no resource limits on any accounts, the server does not have to perform resource counting. If you have a very high query volume, it may be worth the time to use a simplified grant structure to reduce permission-checking overhead.

If your problem is with some specific MySQL expression or function, you can use the `BENCHMARK()` function from the **mysql** client program to perform a timing test. Its syntax is `BENCHMARK(`*`loop_count,expression`*`)`. For example:

```
mysql> SELECT BENCHMARK(1000000,1+1);
+------------------------+
| BENCHMARK(1000000,1+1) |
+------------------------+
|                      0 |
+------------------------+
1 row in set (0.32 sec)
```

This result was obtained on a Pentium II 400MHz system. It shows that MySQL can execute 1,000,000 simple addition expressions in 0.32 seconds on that system.

All MySQL functions should be very optimized, but there may be some exceptions. `BENCHMARK()` is a great tool to find out if this is a problem with your query.

7.2.1. `EXPLAIN` Syntax (Get Information About a `SELECT`)

```
EXPLAIN tbl_name
```

Or:

```
EXPLAIN SELECT select_options
```

The EXPLAIN statement can be used either as a synonym for DESCRIBE or as a way to obtain information about how MySQL executes a SELECT statement:

- The EXPLAIN *tbl_name* syntax is synonymous with DESCRIBE *tbl_name* or SHOW COLUMNS FROM *tbl_name*.
- When you precede a SELECT statement with the keyword EXPLAIN, MySQL explains how it would process the SELECT, providing information about how tables are joined and in which order.

This section provides information about the second use of EXPLAIN.

With the help of EXPLAIN, you can see when you must add indexes to tables to get a faster SELECT that uses indexes to find records.

If you have a problem with incorrect index usage, you should run ANALYZE TABLE to update table statistics such as cardinality of keys, which can affect the choices the optimizer makes. See Section 13.5.2.1, "ANALYZE TABLE Syntax".

You can also see whether the optimizer joins the tables in an optimal order. To force the optimizer to use a join order corresponding to the order in which the tables are named in the SELECT statement, begin the statement with SELECT STRAIGHT_JOIN rather than just SELECT.

EXPLAIN returns a row of information for each table used in the SELECT statement. The tables are listed in the output in the order that MySQL would read them while processing the query. MySQL resolves all joins using a single-sweep multi-join method. This means that MySQL reads a row from the first table, then finds a matching row in the second table, then in the third table, and so on. When all tables are processed, it outputs the selected columns and backtracks through the table list until a table is found for which there are more matching rows. The next row is read from this table and the process continues with the next table.

In MySQL version 4.1, the EXPLAIN output format was changed to work better with constructs such as UNION statements, subqueries, and derived tables. Most notable is the addition of two new columns: id and select_type. You do not see these columns when using servers older than MySQL 4.1.

Each output row from EXPLAIN provides information about one table, and each row consists of the following columns:

- id

  The SELECT identifier. This is the sequential number of the SELECT within the query.

- select_type

  The type of SELECT, which can be any of the following:

  - SIMPLE

    Simple SELECT (not using UNION or subqueries)

- `PRIMARY`

  Outermost `SELECT`

- `UNION`

  Second or later `SELECT` statement in a `UNION`

- `DEPENDENT UNION`

  Second or later `SELECT` statement in a `UNION`, dependent on outer query

- `UNION RESULT`

  Result of a `UNION`.

- `SUBQUERY`

  First `SELECT` in subquery

- `DEPENDENT SUBQUERY`

  First `SELECT` in subquery, dependent on outer query

- `DERIVED`

  Derived table `SELECT` (subquery in `FROM` clause)

- `table`

  The table to which the row of output refers.

- `type`

  The join type. The different join types are listed here, ordered from the best type to the worst:

  - `system`

    The table has only one row (= system table). This is a special case of the `const` join type.

  - `const`

    The table has at most one matching row, which is read at the start of the query. Because there is only one row, values from the column in this row can be regarded as constants by the rest of the optimizer. `const` tables are very fast because they are read only once!

    `const` is used when you compare all parts of a `PRIMARY KEY` or `UNIQUE` index with constant values. In the following queries, *`tbl_name`* can be used as a `const` table:

    ```
    SELECT * FROM tbl_name WHERE primary_key=1;

    SELECT * FROM tbl_name
    WHERE primary_key_part1=1 AND primary_key_part2=2;
    ```

  - `eq_ref`

One row is read from this table for each combination of rows from the previous tables. Other than the `const` types, this is the best possible join type. It is used when all parts of an index are used by the join and the index is a `PRIMARY KEY` or `UNIQUE` index.

`eq_ref` can be used for indexed columns that are compared using the `=` operator. The comparison value can be a constant or an expression that uses columns from tables that are read before this table.

In the following examples, MySQL can use an `eq_ref` join to process *`ref_table`*:

```
SELECT * FROM ref_table,other_table
WHERE ref_table.key_column=other_table.column;

SELECT * FROM ref_table,other_table
WHERE ref_table.key_column_part1=other_table.column
AND ref_table.key_column_part2=1;
```

- `ref`

  All rows with matching index values are read from this table for each combination of rows from the previous tables. `ref` is used if the join uses only a leftmost prefix of the key or if the key is not a `PRIMARY KEY` or `UNIQUE` index (in other words, if the join cannot select a single row based on the key value). If the key that is used matches only a few rows, this is a good join type.

  `ref` can be used for indexed columns that are compared using the `=` operator.

  In the following examples, MySQL can use a `ref` join to process *`ref_table`*:

```
SELECT * FROM ref_table WHERE key_column=expr;

SELECT * FROM ref_table,other_table
WHERE ref_table.key_column=other_table.column;

SELECT * FROM ref_table,other_table
WHERE ref_table.key_column_part1=other_table.column
AND ref_table.key_column_part2=1;
```

- `ref_or_null`

  This join type is like `ref`, but with the addition that MySQL does an extra search for rows that contain `NULL` values. This join type optimization is new for MySQL 4.1.1 and is mostly used when resolving subqueries.

  In the following examples, MySQL can use a `ref_or_null` join to process *`ref_table`*:

```
SELECT * FROM ref_table
WHERE key_column=expr OR key_column IS NULL;
```

  See Section 7.2.7, “How MySQL Optimizes `IS NULL`”.

- `index_merge`

  This join type indicates that the Index Merge optimization is used. In this case, the `key` column

contains a list of indexes used, and `key_len` contains a list of the longest key parts for the indexes used. For more information, see Section 7.2.6, “Index Merge Optimization”.

- `unique_subquery`

  This type replaces `ref` for some `IN` subqueries of the following form:

  value `IN (SELECT` *primary_key* `FROM` *single_table* `WHERE` *some_expr*`)`

  `unique_subquery` is just an index lookup function that replaces the subquery completely for better efficiency.

- `index_subquery`

  This join type is similar to `unique_subquery`. It replaces `IN` subqueries, but it works for non-unique indexes in subqueries of the following form:

  value `IN (SELECT` *key_column* `FROM` *single_table* `WHERE` *some_expr*`)`

- `range`

  Only rows that are in a given range are retrieved, using an index to select the rows. The `key` column indicates which index is used. The `key_len` contains the longest key part that was used. The `ref` column is `NULL` for this type.

  `range` can be used for when a key column is compared to a constant using any of the `=`, `<>`, `>`, `>=`, `<`, `<=`, `IS NULL`, `<=>`, `BETWEEN`, or `IN` operators:

  ```
  SELECT * FROM tbl_name
  WHERE key_column = 10;

  SELECT * FROM tbl_name
  WHERE key_column BETWEEN 10 and 20;

  SELECT * FROM tbl_name
  WHERE key_column IN (10,20,30);

  SELECT * FROM tbl_name
  WHERE key_part1= 10 AND key_part2 IN (10,20,30);
  ```

- `index`

  This join type is the same as `ALL`, except that only the index tree is scanned. This usually is faster than `ALL`, because the index file usually is smaller than the data file.

  MySQL can use this join type when the query uses only columns that are part of a single index.

- `ALL`

  A full table scan is done for each combination of rows from the previous tables. This is normally not good if the table is the first table not marked `const`, and usually *very* bad in all other cases. Normally, you can avoid `ALL` by adding indexes that allow row retrieval from the table based on constant values or column values from earlier tables.

- `possible_keys`

The possible_keys column indicates which indexes MySQL could use to find the rows in this table. Note that this column is totally independent of the order of the tables as displayed in the output from EXPLAIN. That means that some of the keys in possible_keys might not be usable in practice with the generated table order.

If this column is NULL, there are no relevant indexes. In this case, you may be able to improve the performance of your query by examining the WHERE clause to see whether it refers to some column or columns that would be suitable for indexing. If so, create an appropriate index and check the query with EXPLAIN again. See Section 13.2.2, "ALTER TABLE Syntax".

To see what indexes a table has, use SHOW INDEX FROM *tbl_name*.

- key

  The key column indicates the key (index) that MySQL actually decided to use. The key is NULL if no index was chosen. To force MySQL to use or ignore an index listed in the possible_keys column, use FORCE INDEX, USE INDEX, or IGNORE INDEX in your query. See Section 13.1.7, "SELECT Syntax".

  For MyISAM and BDB tables, running ANALYZE TABLE helps the optimizer choose better indexes. For MyISAM tables, **myisamchk --analyze** does the same. See Section 13.5.2.1, "ANALYZE TABLE Syntax" and Section 5.7.3, "Table Maintenance and Crash Recovery".

- key_len

  The key_len column indicates the length of the key that MySQL decided to use. The length is NULL if the key column says NULL. Note that the value of key_len allows you to determine how many parts of a multiple-part key MySQL actually uses.

- ref

  The ref column shows which columns or constants are used with the key to select rows from the table.

- rows

  The rows column indicates the number of rows MySQL believes it must examine to execute the query.

- Extra

  This column contains additional information about how MySQL resolves the query. Here is an explanation of the different text strings that can appear in this column:

  - Distinct

    MySQL stops searching for more rows for the current row combination after it has found the first matching row.

  - Not exists

    MySQL was able to do a LEFT JOIN optimization on the query and does not examine more rows in this table for the previous row combination after it finds one row that matches the LEFT JOIN criteria.

    Here is an example of the type of query that can be optimized this way:

```
SELECT * FROM t1 LEFT JOIN t2 ON t1.id=t2.id
WHERE t2.id IS NULL;
```

Assume that `t2.id` is defined as `NOT NULL`. In this case, MySQL scans `t1` and looks up the rows in `t2` using the values of `t1.id`. If MySQL finds a matching row in `t2`, it knows that `t2.id` can never be `NULL`, and does not scan through the rest of the rows in `t2` that have the same `id` value. In other words, for each row in `t1`, MySQL needs to do only a single lookup in `t2`, regardless of how many rows actually match in `t2`.

- `range checked for each record (index map: #)`

  MySQL found no good index to use, but found that some of indexes might be used once column values from preceding tables are known. For each row combination in the preceding tables, MySQL checks whether it is possible to use a `range` or `index_merge` access method to retrieve rows. The applicability criteria are as described in Section 7.2.5, “Range Optimization” and Section 7.2.6, “Index Merge Optimization”, with the exception that all column values for the preceding table are known and considered to be constants.

  This is not very fast, but is faster than performing a join with no index at all.

- `Using filesort`

  MySQL needs to do an extra pass to find out how to retrieve the rows in sorted order. The sort is done by going through all rows according to the join type and storing the sort key and pointer to the row for all rows that match the `WHERE` clause. The keys then are sorted and the rows are retrieved in sorted order. See Section 7.2.10, “How MySQL Optimizes `ORDER BY`”.

- `Using index`

  The column information is retrieved from the table using only information in the index tree without having to do an additional seek to read the actual row. This strategy can be used when the query uses only columns that are part of a single index.

- `Using temporary`

  To resolve the query, MySQL needs to create a temporary table to hold the result. This typically happens if the query contains `GROUP BY` and `ORDER BY` clauses that list columns differently.

- `Using where`

  A `WHERE` clause is used to restrict which rows to match against the next table or send to the client. Unless you specifically intend to fetch or examine all rows from the table, you may have something wrong in your query if the `Extra` value is not `Using where` and the table join type is `ALL` or `index`.

  If you want to make your queries as fast as possible, you should look out for `Extra` values of `Using filesort` and `Using temporary`.

- `Using sort_union(...)`, `Using union(...)`, `Using intersect(...)`

  These indicate how index scans are merged for the `index_merge` join type. See Section 7.2.6, “Index Merge Optimization” for more information.

- `Using index for group-by`

  Similar to the `Using index` way of accessing a table, `Using index for group-by` in-

dicates that MySQL found an index that can be used to retrieve all columns of a GROUP BY or DISTINCT query without any extra disk access to the actual table. Additionally, the index is used in the most efficient way so that for each group, only a few index entries are read. For details, see Section 7.2.11, "How MySQL Optimizes GROUP BY".

You can get a good indication of how good a join is by taking the product of the values in the rows column of the EXPLAIN output. This should tell you roughly how many rows MySQL must examine to execute the query. If you restrict queries with the max_join_size system variable, this product also is used to determine which multiple-table SELECT statements to execute. See Section 7.5.2, "Tuning Server Parameters".

The following example shows how a multiple-table join can be optimized progressively based on the information provided by EXPLAIN.

Suppose that you have the SELECT statement shown here and you plan to examine it using EXPLAIN:

```
EXPLAIN SELECT tt.TicketNumber, tt.TimeIn,
            tt.ProjectReference, tt.EstimatedShipDate,
            tt.ActualShipDate, tt.ClientID,
            tt.ServiceCodes, tt.RepetitiveID,
            tt.CurrentProcess, tt.CurrentDPPerson,
            tt.RecordVolume, tt.DPPrinted, et.COUNTRY,
            et_1.COUNTRY, do.CUSTNAME
        FROM tt, et, et AS et_1, do
        WHERE tt.SubmitTime IS NULL
            AND tt.ActualPC = et.EMPLOYID
            AND tt.AssignedPC = et_1.EMPLOYID
            AND tt.ClientID = do.CUSTNMBR;
```

For this example, make the following assumptions:

- The columns being compared have been declared as follows:

Table	Column	Column Type
tt	ActualPC	CHAR(10)
tt	AssignedPC	CHAR(10)
tt	ClientID	CHAR(10)
et	EMPLOYID	CHAR(15)
do	CUSTNMBR	CHAR(15)

- The tables have the following indexes:

Table	Index
tt	ActualPC
tt	AssignedPC
tt	ClientID
et	EMPLOYID (primary key)
do	CUSTNMBR (primary key)

- The `tt.ActualPC` values are not evenly distributed.

Initially, before any optimizations have been performed, the `EXPLAIN` statement produces the following information:

```
table type possible_keys key  key_len ref  rows  Extra
et    ALL  PRIMARY       NULL NULL    NULL 74
do    ALL  PRIMARY       NULL NULL    NULL 2135
et_1  ALL  PRIMARY       NULL NULL    NULL 74
tt    ALL  AssignedPC,   NULL NULL    NULL 3872
           ClientID,
           ActualPC
      range checked for each record (key map: 35)
```

Because `type` is `ALL` for each table, this output indicates that MySQL is generating a Cartesian product of all the tables; that is, every combination of rows. This takes quite a long time, because the product of the number of rows in each table must be examined. For the case at hand, this product is `74 * 2135 * 74 * 3872 = 45,268,558,720` rows. If the tables were bigger, you can only imagine how long it would take.

One problem here is that MySQL can use indexes on columns more efficiently if they are declared the same. (For `ISAM` tables, indexes may not be used at all unless the columns are declared the same.) In this context, `VARCHAR` and `CHAR` are the same unless they are declared as different lengths. Because `tt.ActualPC` is declared as `CHAR(10)` and `et.EMPLOYID` is declared as `CHAR(15)`, there is a length mismatch.

To fix this disparity between column lengths, use `ALTER TABLE` to lengthen `ActualPC` from 10 characters to 15 characters:

```
mysql> ALTER TABLE tt MODIFY ActualPC VARCHAR(15);
```

`tt.ActualPC` and `et.EMPLOYID` are both `VARCHAR(15)`. Executing the `EXPLAIN` statement again produces this result:

```
table type   possible_keys key     key_len ref         rows    Extra
tt    ALL    AssignedPC,   NULL    NULL    NULL        3872    Using
             ClientID,                                         where
             ActualPC
do    ALL    PRIMARY       NULL    NULL    NULL        2135
      range checked for each record (key map: 1)
et_1  ALL    PRIMARY       NULL    NULL    NULL        74
      range checked for each record (key map: 1)
et    eq_ref PRIMARY       PRIMARY 15      tt.ActualPC 1
```

This is not perfect, but is much better: The product of the `rows` values is less by a factor of 74. This version is executed in a couple of seconds.

A second alteration can be made to eliminate the column length mismatches for the `tt.AssignedPC = et_1.EMPLOYID` and `tt.ClientID = do.CUSTNMBR` comparisons:

```
mysql> ALTER TABLE tt MODIFY AssignedPC VARCHAR(15),
    ->                MODIFY ClientID   VARCHAR(15);
```

`EXPLAIN` produces the output shown here:

```
table type   possible_keys key      key_len ref           rows Extra
```

```
et    ALL    PRIMARY       NULL     NULL    NULL          74
tt    ref    AssignedPC,   ActualPC 15      et.EMPLOYID   52   Using
             ClientID,                                         where
             ActualPC
et_1  eq_ref PRIMARY       PRIMARY  15      tt.AssignedPC 1
do    eq_ref PRIMARY       PRIMARY  15      tt.ClientID   1
```

This is almost as good as it can get.

The remaining problem is that, by default, MySQL assumes that values in the `tt.ActualPC` column are evenly distributed, and that is not the case for the `tt` table. Fortunately, it is easy to tell MySQL to analyze the key distribution:

```
mysql> ANALYZE TABLE tt;
```

The join is perfect, and `EXPLAIN` produces this result:

```
table type   possible_keys key     key_len ref           rows Extra
tt    ALL    AssignedPC    NULL    NULL    NULL          3872 Using
             ClientID,                                        where
             ActualPC
et    eq_ref PRIMARY       PRIMARY 15      tt.ActualPC   1
et_1  eq_ref PRIMARY       PRIMARY 15      tt.AssignedPC 1
do    eq_ref PRIMARY       PRIMARY 15      tt.ClientID   1
```

Note that the `rows` column in the output from `EXPLAIN` is an educated guess from the MySQL join optimizer. You should check whether the numbers are even close to the truth. If not, you may get better performance by using `STRAIGHT_JOIN` in your `SELECT` statement and trying to list the tables in a different order in the `FROM` clause.

7.2.2. Estimating Query Performance

In most cases, you can estimate the performance by counting disk seeks. For small tables, you can usually find a row in one disk seek (because the index is probably cached). For bigger tables, you can estimate that, using B-tree indexes, you need this many seeks to find a row: `log(`*`row_count`*`) / log(`*`index_block_length`*` / 3 * 2 / (`*`index_length`*` + `*`data_pointer_length`*`)) + 1`.

In MySQL, an index block is usually 1024 bytes and the data pointer is usually 4 bytes. For a 500,000-row table with an index length of 3 bytes (medium integer), the formula indicates `log(500,000)/log(1024/3*2/(3+4)) + 1` = 4 seeks.

This index would require storage of about 500,000 * 7 * 3/2 = 5.2MB (assuming a typical index buffer fill ratio of 2/3), so you probably have much of the index in memory and you probably need only one or two calls to read data to find the row.

For writes, however, you need four seek requests (as above) to find where to place the new index and normally two seeks to update the index and write the row.

Note that the preceding discussion doesn't mean that your application performance slowly degenerates by log N. As long as everything is cached by the OS or the MySQL server, things become only marginally slower as the table gets bigger. After the data gets too big to be cached, things start to go much slower until your applications are only bound by disk-seeks (which increase by log *N*). To avoid this, increase the key cache size as the data grows. For `MyISAM` tables, the key cache size is controlled by the `key_buffer_size` system variable. See Section 7.5.2, “Tuning Server Parameters”.

7.2.3. Speed of SELECT Queries

In general, when you want to make a slow SELECT ... WHERE query faster, the first thing to check is whether you can add an index. All references between different tables should usually be done with indexes. You can use the EXPLAIN statement to determine which indexes are used for a SELECT. See Section 7.4.5, "How MySQL Uses Indexes" and Section 7.2.1, "EXPLAIN Syntax (Get Information About a SELECT)".

Some general tips for speeding up queries on MyISAM tables:

- To help MySQL optimize queries better, use ANALYZE TABLE or run **myisamchk --analyze** on a table after it has been loaded with data. This updates a value for each index part that indicates the average number of rows that have the same value. (For unique indexes, this is always 1.) MySQL uses this to decide which index to choose when you join two tables based on a non-constant expression. You can check the result from the table analysis by using SHOW INDEX FROM *tbl_name* and examining the Cardinality value. **myisamchk --description --verbose** shows index distribution information.
- To sort an index and data according to an index, use **myisamchk --sort-index --sort-records=1** (if you want to sort on index 1). This is a good way to make queries faster if you have a unique index from which you want to read all records in order according to the index. Note that the first time you sort a large table this way, it may take a long time.

7.2.4. How MySQL Optimizes WHERE Clauses

This section discusses optimizations that can be made for processing WHERE clauses. The examples use SELECT statements, but the same optimizations apply for WHERE clauses in DELETE and UPDATE statements.

Note that work on the MySQL optimizer is ongoing, so this section is incomplete. MySQL does many optimizations, not all of which are documented here.

Some of the optimizations performed by MySQL are listed here:

- Removal of unnecessary parentheses:

```
   ((a AND b) AND c OR (((a AND b) AND (c AND d))))
-> (a AND b AND c) OR (a AND b AND c AND d)
```

- Constant folding:

```
   (a<b AND b=c) AND a=5
-> b>5 AND b=c AND a=5
```

- Constant condition removal (needed because of constant folding):

```
   (B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)
-> B=5 OR B=6
```

- Constant expressions used by indexes are evaluated only once.
- COUNT(*) on a single table without a WHERE is retrieved directly from the table information for MyISAM and HEAP tables. This is also done for any NOT NULL expression when used with only

one table.

- Early detection of invalid constant expressions. MySQL quickly detects that some `SELECT` statements are impossible and returns no rows.
- `HAVING` is merged with `WHERE` if you don't use `GROUP BY` or group functions (`COUNT()`, `MIN()`, and so on).
- For each table in a join, a simpler `WHERE` is constructed to get a fast `WHERE` evaluation for the table and also to skip records as soon as possible.
- All constant tables are read first before any other tables in the query. A constant table is any of the following:
  - An empty table or a table with one row.
  - A table that is used with a `WHERE` clause on a `PRIMARY KEY` or a `UNIQUE` index, where all index parts are compared to constant expressions and are defined as `NOT NULL`.

  All of the following tables are used as constant tables:

  ```
  SELECT * FROM t WHERE primary_key=1;
  SELECT * FROM t1,t2
      WHERE t1.primary_key=1 AND t2.primary_key=t1.id;
  ```

- The best join combination for joining the tables is found by trying all possibilities. If all columns in `ORDER BY` and `GROUP BY` clauses come from the same table, that table is preferred first when joining.
- If there is an `ORDER BY` clause and a different `GROUP BY` clause, or if the `ORDER BY` or `GROUP BY` contains columns from tables other than the first table in the join queue, a temporary table is created.
- If you use `SQL_SMALL_RESULT`, MySQL uses an in-memory temporary table.
- Each table index is queried, and the best index is used unless the optimizer believes that it is more efficient to use a table scan. At one time, a scan was used based on whether the best index spanned more than 30% of the table. The optimizer is more complex and bases its estimate on additional factors such as table size, number of rows, and I/O block size, so a fixed percentage no longer determines the choice between using an index or a scan.
- In some cases, MySQL can read rows from the index without even consulting the data file. If all columns used from the index are numeric, only the index tree is used to resolve the query.
- Before each record is output, those that do not match the `HAVING` clause are skipped.

Some examples of queries that are very fast:

```
SELECT COUNT(*) FROM tbl_name;

SELECT MIN(key_part1),MAX(key_part1) FROM tbl_name;

SELECT MAX(key_part2) FROM tbl_name
    WHERE key_part1=constant;

SELECT ... FROM tbl_name
    ORDER BY key_part1,key_part2,... LIMIT 10;
```

```
SELECT ... FROM tbl_name
    ORDER BY key_part1 DESC, key_part2 DESC, ... LIMIT 10;
```

The following queries are resolved using only the index tree, assuming that the indexed columns are numeric:

```
SELECT key_part1,key_part2 FROM tbl_name WHERE key_part1=val;

SELECT COUNT(*) FROM tbl_name
    WHERE key_part1=val1 AND key_part2=val2;

SELECT key_part2 FROM tbl_name GROUP BY key_part1;
```

The following queries use indexing to retrieve the rows in sorted order without a separate sorting pass:

```
SELECT ... FROM tbl_name
    ORDER BY key_part1,key_part2,... ;

SELECT ... FROM tbl_name
    ORDER BY key_part1 DESC, key_part2 DESC, ... ;
```

7.2.5. Range Optimization

The `range` access method uses a single index to retrieve a subset of table records that are contained within one or several index value intervals. It can be used for a single-part or multiple-part index. A detailed description of how intervals are extracted from the `WHERE` clause is given in the following sections.

7.2.5.1. Range Access Method for Single-Part Indexes

For a single-part index, index value intervals can be conveniently represented by corresponding conditions in the `WHERE` clause, so we'll talk about ``range conditions'' instead of intervals.

The definition of a range condition for a single-part index is as follows:

- For both `BTREE` and `HASH` indexes, comparison of a key part with a constant value is a range condition when using the `=`, `<=>`, `IN`, `IS NULL`, or `IS NOT NULL` operators.
- For `BTREE` indexes, comparison of a key part with a constant value is a range condition when using the `>`, `<`, `>=`, `<=`, `BETWEEN`, `!=`, or `<>` operators, or `LIKE 'pattern'` (where `'pattern'` doesn't start with a wildcard).
- For all types of indexes, multiple range conditions combined with `OR` or `AND` form a range condition.

``Constant value'' in the preceding descriptions means one of the following:

- A constant from the query string
- A column of a `const` or `system` table from the same join
- The result of an uncorrelated subquery
- Any expression composed entirely from subexpressions of the preceding types

Here are some examples of queries with range conditions in the WHERE clause:

```
SELECT * FROM t1 WHERE key_col > 1 AND key_col < 10;

SELECT * FROM t1 WHERE key_col = 1 OR key_col IN (15,18,20);

SELECT * FROM t1 WHERE key_col LIKE 'ab%' OR key_col BETWEEN
'bar' AND 'foo';
```

Note that some non-constant values may be converted to constants during the constant propagation phase.

MySQL tries to extract range conditions from the WHERE clause for each of the possible indexes. During the extraction process, conditions that can't be used for constructing the range condition are dropped, conditions that produce overlapping ranges are combined, and conditions that produce empty ranges are removed.

For example, consider the following statement, where key1 is an indexed column and nonkey is not indexed:

```
SELECT * FROM t1 WHERE
   (key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
   (key1 < 'bar' AND nonkey = 4) OR
   (key1 < 'uux' AND key1 > 'z');
```

The extraction process for key key1 is as follows:

1. Start with original WHERE clause:

   ```
   (key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
   (key1 < 'bar' AND nonkey = 4) OR
   (key1 < 'uux' AND key1 > 'z')
   ```

2. Remove nonkey = 4 and key1 LIKE '%b' because they cannot be used for a range scan. The right way to remove them is to replace them with TRUE, so that we don't miss any matching records when doing the range scan. Having replaced them with TRUE, we get:

   ```
   (key1 < 'abc' AND (key1 LIKE 'abcde%' OR TRUE)) OR
   (key1 < 'bar' AND TRUE) OR
   (key1 < 'uux' AND key1 > 'z')
   ```

3. Collapse conditions that are always true or false:

   - (key1 LIKE 'abcde%' OR TRUE) is always true
   - (key1 < 'uux' AND key1 > 'z') is always false

   Replacing these conditions with constants, we get:

   ```
   (key1 < 'abc' AND TRUE) OR (key1 < 'bar' AND TRUE) OR (FALSE)
   ```

   Removing unnecessary TRUE and FALSE constants, we obtain

   ```
   (key1 < 'abc') OR (key1 < 'bar')
   ```

4. Combining overlapping intervals into one yields the final condition to be used for the range scan:

   ```
   (key1 < 'bar')
   ```

In general (and as demonstrated in the example), the condition used for a range scan is less restrictive than the `WHERE` clause. MySQL performs an additional check to filter out rows that satisfy the range condition but not the full `WHERE` clause.

The range condition extraction algorithm can handle nested `AND`/`OR` constructs of arbitrary depth, and its output doesn't depend on the order in which conditions appear in `WHERE` clause.

7.2.5.2. Range Access Method for Multiple-Part Indexes

Range conditions on a multiple-part index are an extension of range conditions for a single-part index. A range condition on a multiple-part index restricts index records to lie within one or several key tuple intervals. Key tuple intervals are defined over a set of key tuples, using ordering from the index.

For example, consider a multiple-part index defined as `key1(key_part1, key_part2, key_part3)`, and the following set of key tuples listed in key order:

```
key_part1  key_part2  key_part3
  NULL       1          'abc'
  NULL       1          'xyz'
  NULL       2          'foo'
   1         1          'abc'
   1         1          'xyz'
   1         2          'abc'
   2         1          'aaa'
```

The condition `key_part1 = 1` defines this interval:

```
(1, -inf, -inf) <= (key_part1, key_part2, key_part3) < (1, +inf, +inf)
```

The interval covers the 4th, 5th, and 6th tuples in the preceding data set and can be used by the range access method.

By contrast, the condition `key_part3 = 'abc'` does not define a single interval and cannot be used by the range access method.

The following descriptions indicate how range conditions work for multiple-part indexes in greater detail.

- For `HASH` indexes, each interval containing identical values can be used. This means that the interval can be produced only for conditions in the following form:

  ```
      key_part1 cmp const1
  AND key_part2 cmp const2
  AND ...
  AND key_partN cmp constN;
  ```

  Here, `const1`, `const2`, ... are constants, `cmp` is one of the `=`, `<=>`, or `IS NULL` comparison operators, and the conditions cover all index parts. (That is, there are `N` conditions, one for each part of an `N`-part index.)

See Section 7.2.5.1, “Range Access Method for Single-Part Indexes” for the definition of what is considered to be a constant.

For example, the following is a range condition for a three-part HASH index:

```
key_part1 = 1 AND key_part2 IS NULL AND key_part3 = 'foo'
```

- For a BTREE index, an interval might be usable for conditions combined with AND, where each condition compares a key part with a constant value using =, <=>, IS NULL, >, <, >=, <=, !=, <>, BETWEEN, or LIKE 'pattern' (where 'pattern' doesn't start with a wildcard). An interval can be used as long as it is possible to determine a single key tuple containing all records that match the condition (or two intervals if <> or != is used). For example, for this condition:

  ```
  key_part1 = 'foo' AND key_part2 >= 10 AND key_part3 > 10
  ```

  The single interval is:

  ```
  ('foo', 10, 10)
     < (key_part1, key_part2, key_part3)
        < ('foo', +inf, +inf)
  ```

  It is possible that the created interval contains more records than the initial condition. For example, the preceding interval includes the value ('foo', 11, 0), which does not satisfy the original condition.

- If conditions that cover sets of records contained within intervals are combined with OR, they form a condition that covers a set of records contained within the union of their intervals. If the conditions are combined with AND, they form a condition that covers a set of records contained within the intersection of their intervals. For example, for this condition on a two-part index:

  ```
  (key_part1 = 1 AND key_part2 < 2)
  OR (key_part1 > 5)
  ```

  The intervals is:

  ```
  (1, -inf) < (key_part1, key_part2) < (1, 2)
  (5, -inf) < (key_part1, key_part2)
  ```

  In this example, the interval on the first line uses one key part for the left bound and two key parts for the right bound. The interval on the second line uses only one key part. The key_len column in the EXPLAIN output indicates the maximum length of the key prefix used.

  In some cases, key_len may indicate that a key part was used, but that might be not what you would expect. Suppose that key_part1 and key_part2 can be NULL. Then the key_len column displays two key part lengths for the following condition:

  ```
  key_part1 >= 1 AND key_part2 < 2
  ```

  But in fact, the condition is converted to this:

  ```
  key_part1 >= 1 AND key_part2 IS NOT NULL
  ```

Section 7.2.5.1, “Range Access Method for Single-Part Indexes” describes how optimizations are performed to combine or eliminate intervals for range conditions on single-part index. Analogous steps are performed for range conditions on multiple-part keys.

7.2.6. Index Merge Optimization

The Index Merge (`index_merge`) method is used to retrieve rows with several `ref`, `ref_or_null`, or `range` scans and merge the results into one. This method is employed when the table condition is a disjunction of conditions for which `ref`, `ref_or_null`, or `range` could be used with different keys.

This ``join'' type optimization is new in MySQL 5.0.0, and represents a significant change in behavior with regard to indexes, because the *old* rule was that the server is only ever able to use at most one index for each referenced table.

In `EXPLAIN` output, this method appears as `index_merge` in the `type` column. In this case, the `key` column contains a list of indexes used, and `key_len` contains a list of the longest key parts for those indexes.

Examples:

```
SELECT * FROM tbl_name WHERE key_part1 = 10 OR key_part2 = 20;

SELECT * FROM tbl_name
    WHERE (key_part1 = 10 OR key_part2 = 20) AND non_key_part=30;

SELECT * FROM t1, t2
    WHERE (t1.key1 IN (1,2) OR t1.key2 LIKE 'value%')
    AND t2.key1=t1.some_col;

SELECT * FROM t1, t2
    WHERE t1.key1=1
    AND (t2.key1=t1.some_col OR t2.key2=t1.some_col2);
```

The Index Merge method has several access algorithms (seen in the `Extra` field of `EXPLAIN` output):

- intersection
- union
- sort-union

The following sections describe these methods in greater detail.

Note: The Index Merge optimization algorithm has the following known deficiencies:

- If a range scan is possible on some key, an Index Merge is not considered. For example, consider this query:

  ```
  SELECT * FROM t1 WHERE (goodkey1 < 10 OR goodkey2 < 20) AND badkey < 30;
  ```

  For this query, two plans are possible:

  1. An Index Merge scan using the `(goodkey1 < 10 OR goodkey2 < 20)` condition.
  2. A range scan using the `badkey < 30` condition.

However, the optimizer only considers the second plan. If that is not what you want, you can make the optimizer consider `index_merge` by using `IGNORE INDEX` or `FORCE INDEX`. The following queries are executed using Index Merge:

```
SELECT * FROM t1 FORCE INDEX(goodkey1,goodkey2)
WHERE (goodkey1 < 10 OR goodkey2 < 20) AND badkey < 30;

SELECT * FROM t1 IGNORE INDEX(badkey)
WHERE (goodkey1 < 10 OR goodkey2 < 20) AND badkey < 30;
```

- If your query has a complex `WHERE` clause with deep `AND`/`OR` nesting and MySQL doesn't choose the optimal plan, try distributing terms using the following identity laws:

  ```
  (x AND y) OR z = (x OR z) AND (y OR z)
  (x OR y) AND z = (x AND z) OR (y AND z)
  ```

The choice between different possible variants of the `index_merge` access method and other access methods is based on cost estimates of various available options.

7.2.6.1. Index Merge Intersection Access Algorithm

This access algorithm can be employed when a `WHERE` clause was converted to several range conditions on different keys combined with `AND`, and each condition is one of the following:

- In this form, where the index has exactly *N* parts (that is, all index parts are covered):

  ```
  key_part1=const1 AND key_part2=const2 ... AND key_partN=constN
  ```

- Any range condition over a primary key of an `InnoDB` or `BDB` table.

Here are some examples:

```
SELECT * FROM innodb_table WHERE primary_key < 10 AND key_col1=20;

SELECT * FROM tbl_name
WHERE (key1_part1=1 AND key1_part2=2) AND key2=2;
```

The Index Merge intersection algorithm performs simultaneous scans on all used indexes and produces the intersection of row sequences that it receives from the merged index scans.

If all columns used in the query are covered by the used indexes, full table records are not retrieved and (`EXPLAIN` output contains `Using index` in `Extra` field in this case). Here is an example of such query:

```
SELECT COUNT(*) FROM t1 WHERE key1=1 AND key2=1;
```

If the used indexes don't cover all columns used in the query, full records are retrieved only when the range conditions for all used keys are satisfied.

If one of the merged conditions is a condition over a primary key of an `InnoDB` or `BDB` table, it is not used for record retrieval, but is used to filter out records retrieved using other conditions.

7.2.6.2. Index Merge Union Access Algorithm

The applicability criteria for this algorithm are similar to those of the Index Merge method intersection algorithm. The algorithm can be employed when the table `WHERE` clause was converted to several range conditions on different keys combined with `OR`, and each condition is one of the following:

- In this form, where the index has exactly *`N`* parts (that is, all index parts are covered):

  ```
  key_part1=const1 AND key_part2=const2 ... AND key_partN=constN
  ```

- Any range condition over a primary key of an `InnoDB` or `BDB` table.
- A condition for which the Index Merge method intersection algorithm is applicable.

Here are some examples:

```
SELECT * FROM t1 WHERE key1=1 OR key2=2 OR key3=3;

SELECT * FROM innodb_table WHERE (key1=1 AND key2=2) OR
  (key3='foo' AND key4='bar') AND key5=5;
```

7.2.6.3. Index Merge Sort-Union Access Algorithm

This access algorithm is employed when the `WHERE` clause was converted to several range conditions combined by `OR`, but for which the Index Merge method union algorithm is not applicable.

Here are some examples:

```
SELECT * FROM tbl_name WHERE key_col1 < 10 OR key_col2 < 20;

SELECT * FROM tbl_name
     WHERE (key_col1 > 10 OR key_col2 = 20) AND nonkey_col=30;
```

The difference between the sort-union algorithm and the union algorithm is that the sort-union algorithm must first fetch row IDs for all records and sort them before returning any records.

7.2.7. How MySQL Optimizes `IS NULL`

MySQL can do the same optimization on *`col_name`* `IS NULL` that it can do with *`col_name`* `=` *`constant_value`*. For example, MySQL can use indexes and ranges to search for `NULL` with `IS NULL`.

```
SELECT * FROM tbl_name WHERE key_col IS NULL;

SELECT * FROM tbl_name WHERE key_col <=> NULL;

SELECT * FROM tbl_name
    WHERE key_col=const1 OR key_col=const2 OR key_col IS NULL;
```

If a `WHERE` clause includes a *`col_name`* `IS NULL` condition for a column that is declared as `NOT NULL`, that expression is optimized away. This optimization does not occur in cases when the column might produce `NULL` anyway; for example, if it comes from a table on the right side of a `LEFT JOIN`.

MySQL 4.1.1 and up can additionally optimize the combination *col_name* = *expr* AND *col_name* IS NULL, a form that is common in resolved subqueries. EXPLAIN shows ref_or_null when this optimization is used.

This optimization can handle one IS NULL for any key part.

Some examples of queries that are optimized, assuming that there is an index on columns a and b of table t2:

```
SELECT * FROM t1 WHERE t1.a=expr OR t1.a IS NULL;

SELECT * FROM t1, t2 WHERE t1.a=t2.a OR t2.a IS NULL;

SELECT * FROM t1, t2
    WHERE (t1.a=t2.a OR t2.a IS NULL) AND t2.b=t1.b;

SELECT * FROM t1, t2
    WHERE t1.a=t2.a AND (t2.b=t1.b OR t2.b IS NULL);

SELECT * FROM t1, t2
    WHERE (t1.a=t2.a AND t2.a IS NULL AND ...)
    OR (t1.a=t2.a AND t2.a IS NULL AND ...);
```

ref_or_null works by first doing a read on the reference key, and then a separate search for rows with a NULL key value.

Note that the optimization can handle only one IS NULL level. In the following query, MySQL uses key lookups only on the expression (t1.a=t2.a AND t2.a IS NULL) and is not able to use the key part on b:

```
SELECT * FROM t1, t2
     WHERE (t1.a=t2.a AND t2.a IS NULL)
     OR (t1.b=t2.b AND t2.b IS NULL);
```

7.2.8. How MySQL Optimizes DISTINCT

DISTINCT combined with ORDER BY needs a temporary table in many cases.

Note that because DISTINCT may use GROUP BY, you should be aware of how MySQL works with columns in ORDER BY or HAVING clauses that are not part of the selected columns. See Section 12.9.3, “GROUP BY with Hidden Fields”.

In most cases, a DISTINCT clause can be considered as a special case of GROUP BY. For example, the following two queries are equivalent:

```
SELECT DISTINCT c1, c2, c3 FROM t1 WHERE c1 > const;

SELECT c1, c2, c3 FROM t1 WHERE c1 > const GROUP BY c1, c2, c3;
```

Due to this equivalence, the optimizations applicable to GROUP BY queries can be also applied to queries with a DISTINCT clause. Thus, for more details on the optimization possibilities for DISTINCT queries, see Section 7.2.11, “How MySQL Optimizes GROUP BY”.

When combining LIMIT *row_count* with DISTINCT, MySQL stops as soon as it finds *row_count* unique rows.

If you don't use columns from all tables named in a query, MySQL stops scanning the not-used tables as

soon as it finds the first match. In the following case, assuming that `t1` is used before `t2` (which you can check with `EXPLAIN`), MySQL stops reading from `t2` (for any particular row in `t1`) when the first row in `t2` is found:

```
SELECT DISTINCT t1.a FROM t1, t2 where t1.a=t2.a;
```

7.2.9. How MySQL Optimizes LEFT JOIN and RIGHT JOIN

`A LEFT JOIN B join_condition` is implemented in MySQL as follows:

- Table `B` is set to depend on table `A` and all tables on which `A` depends.
- Table `A` is set to depend on all tables (except `B`) that are used in the `LEFT JOIN` condition.
- The `LEFT JOIN` condition is used to decide how to retrieve rows from table B. (In other words, any condition in the `WHERE` clause is not used.)
- All standard join optimizations are done, with the exception that a table is always read after all tables on which it depends. If there is a circular dependence, MySQL issues an error.
- All standard `WHERE` optimizations are done.
- If there is a row in `A` that matches the `WHERE` clause, but there is no row in `B` that matches the `ON` condition, an extra `B` row is generated with all columns set to `NULL`.
- If you use `LEFT JOIN` to find rows that don't exist in some table and you have the following test: *`col_name`* `IS NULL` in the `WHERE` part, where *`col_name`* is a column that is declared as `NOT NULL`, MySQL stops searching for more rows (for a particular key combination) after it has found one row that matches the `LEFT JOIN` condition.

`RIGHT JOIN` is implemented analogously to `LEFT JOIN`, with the roles of the tables reversed.

The join optimizer calculates the order in which tables should be joined. The table read order forced by `LEFT JOIN` and `STRAIGHT_JOIN` helps the join optimizer do its work much more quickly, because there are fewer table permutations to check. Note that this means that if you do a query of the following type, MySQL does a full scan on `b` because the `LEFT JOIN` forces it to be read before `d`:

```
SELECT *
    FROM a,b LEFT JOIN c ON (c.key=a.key) LEFT JOIN d ON (d.key=a.key)
    WHERE b.key=d.key;
```

The fix in this case is to rewrite the query as follows:

```
SELECT *
    FROM b,a LEFT JOIN c ON (c.key=a.key) LEFT JOIN d ON (d.key=a.key)
    WHERE b.key=d.key;
```

Starting from 4.0.14, MySQL does the following `LEFT JOIN` optimization: If the `WHERE` condition is always false for the generated `NULL` row, the `LEFT JOIN` is changed to a normal join.

For example, the `WHERE` clause would be false in the following query if `t2.column1` would be `NULL`:

```
SELECT * FROM t1 LEFT JOIN t2 ON (column1) WHERE t2.column2=5;
```

Therefore, it's safe to convert the query to a normal join:

```
SELECT * FROM t1, t2 WHERE t2.column2=5 AND t1.column1=t2.column1;
```

This can be made faster because MySQL can use table `t2` before table `t1` if this would result in a better query plan. To force a specific table order, use `STRAIGHT_JOIN`.

7.2.10. How MySQL Optimizes `ORDER BY`

In some cases, MySQL can use an index to satisfy an `ORDER BY` clause without doing any extra sorting.

The index can also be used even if the `ORDER BY` doesn't match the index exactly, as long as all the unused index parts and all the extra are `ORDER BY` columns are constants in the `WHERE` clause. The following queries use the index to resolve the `ORDER BY` part:

```
SELECT * FROM t1 ORDER BY key_part1,key_part2,... ;
SELECT * FROM t1 WHERE key_part1=constant ORDER BY key_part2;
SELECT * FROM t1 ORDER BY key_part1 DESC, key_part2 DESC;
SELECT * FROM t1
    WHERE key_part1=1 ORDER BY key_part1 DESC, key_part2 DESC;
```

In some cases, MySQL *cannot* use indexes to resolve the `ORDER BY`, although it still uses indexes to find the rows that match the `WHERE` clause. These cases include the following:

- You use `ORDER BY` on different keys:

  ```
  SELECT * FROM t1 ORDER BY key1, key2;
  ```

- You use `ORDER BY` on non-consecutive key parts:

  ```
  SELECT * FROM t1 WHERE key2=constant ORDER BY key_part2;
  ```

- You mix `ASC` and `DESC`:

  ```
  SELECT * FROM t1 ORDER BY key_part1 DESC, key_part2 ASC;
  ```

- The key used to fetch the rows is not the same as the one used in the `ORDER BY`:

  ```
  SELECT * FROM t1 WHERE key2=constant ORDER BY key1;
  ```

- You are joining many tables, and the columns in the `ORDER BY` are not all from the first non-constant table that is used to retrieve rows. (This is the first table in the `EXPLAIN` output that doesn't have a `const` join type.)
- You have different `ORDER BY` and `GROUP BY` expressions.
- The type of table index used doesn't store rows in order. For example, this is true for a `HASH` index in a `HEAP` table.

With `EXPLAIN SELECT ... ORDER BY`, you can check whether MySQL can use indexes to resolve the query. It cannot if you see `Using filesort` in the `Extra` column. See Section 7.2.1, "`EXPLAIN` Syntax (Get Information About a `SELECT`)".

In those cases where MySQL must sort the result, it uses the following `filesort` algorithm before MySQL 4.1:

1. Read all rows according to key or by table scanning. Rows that don't match the `WHERE` clause are skipped.
2. For each row, store a pair of values in a buffer (the sort key and the row pointer). The size of the buffer is the value of the `sort_buffer_size` system variable.
3. When the buffer gets full, run a qsort (quicksort) on it and store the result in a temporary file. Save a pointer to the sorted block. (If all pairs fit into the sort buffer, no temporary file is created.)
4. Repeat the preceding steps until all rows have been read.
5. Do a multi-merge of up to `MERGEBUFF` (7) regions to one block in another temporary file. Repeat until all blocks from the first file are in the second file.
6. Repeat the following until there are fewer than `MERGEBUFF2` (15) blocks left.
7. On the last multi-merge, only the pointer to the row (the last part of the sort key) is written to a result file.
8. Read the rows in sorted order by using the row pointers in the result file. To optimize this, we read in a big block of row pointers, sort them, and use them to read the rows in sorted order into a row buffer. The size of the buffer is the value of the `read_rnd_buffer_size` system variable. The code for this step is in the `sql/records.cc` source file.

One problem with this approach is that it reads rows twice: One time when evaluating the `WHERE` clause, and again after sorting the pair values. And even if the rows were accessed successively the first time (for example, if a table scan is done), the second time they are accessed randomly. (The sort keys are ordered, but the row positions are not.)

In MySQL 4.1 and up, a `filesort` optimization is used that records not only the sort key value and row position, but also the columns required for the query. This avoids reading the rows twice. The modified `filesort` algorithm works like this:

1. Read the rows that match the `WHERE` clause, as before.
2. For each row, record a tuple of values consisting of the sort key value and row position, and also the columns required for the query.
3. Sort the tuples by sort key value
4. Retrieve the rows in sorted order, but read the required columns directly from the sorted tuples rather than by accessing the table a second time.

Using the modified `filesort` algorithm, the tuples are longer than the pairs used in the original method, and fewer of them fit in the sort buffer (the size of which is given by `sort_buffer_size`). As a result, it is possible for the extra I/O to make the modified approach slower, not faster. To avoid a slowdown, the optimization is used only if the total size of the extra columns in the sort tuple does not exceed the value of the `max_length_for_sort_data` system variable. (A symptom of setting the

value of this variable too high is that you should see high disk activity and low CPU activity.)

If you want to increase `ORDER BY` speed, first see whether you can get MySQL to use indexes rather than an extra sorting phase. If this is not possible, you can try the following strategies:

- Increase the size of the `sort_buffer_size` variable.
- Increase the size of the `read_rnd_buffer_size` variable.
- Change `tmpdir` to point to a dedicated filesystem with lots of empty space. If you use MySQL 4.1 or later, this option accepts several paths that are used in round-robin fashion. Paths should be separated by colon characters (`':'`) on Unix and semicolon characters (`';'`) on Windows, NetWare, and OS/2. You can use this feature to spread the load across several directories. *Note:* The paths should be for directories in filesystems that are located on different *physical* disks, not different partitions of the same disk.

By default, MySQL sorts all `GROUP BY` *`col1, col2, ...`* queries as if you specified `ORDER BY` *`col1, col2, ...`* in the query as well. If you include an `ORDER BY` clause explicitly that contains the same column list, MySQL optimizes it away without any speed penalty, although the sorting still occurs. If a query includes `GROUP BY` but you want to avoid the overhead of sorting the result, you can suppress sorting by specifying `ORDER BY NULL`. For example:

```
INSERT INTO foo
SELECT a, COUNT(*) FROM bar GROUP BY a ORDER BY NULL;
```

7.2.11. How MySQL Optimizes GROUP BY

The most general way to satisfy a `GROUP BY` clause is to scan the whole table and create a new temporary table where all rows from each group are consecutive, and then use this temporary table to discover groups and apply aggregate functions (if any). In some cases, MySQL is able to do much better than that and to avoid creation of temporary tables by using index access.

The most important preconditions for using indexes for `GROUP BY` are that all `GROUP BY` columns reference attributes from the same index, and the index stores its keys in order (for example, this is a B-Tree index, and not a HASH index). Whether usage of temporary tables can be replaced by index access also depends on which parts of an index are used in a query, the conditions specified for these parts, and the selected aggregate functions.

There are two ways to execute a `GROUP BY` query via index access, as detailed in the following sections. In the first method, the grouping operation is applied together with all range predicates (if any). The second method first performs a range scan, and then groups the resulting tuples.

7.2.11.1. Loose index scan

The most efficient way is when the index is used to directly retrieve the group fields. With this access method, MySQL uses the property of some index types (for example, B-Trees) that the keys are ordered. This property allows use of lookup groups in an index without having to consider all keys in the index that satisfy all `WHERE` conditions. Since this access method considers only a fraction of the keys in an index, it is called ``loose index scan.'' When there is no `WHERE` clause, a loose index scan reads as many keys as the number of groups, which may be a much smaller number than all keys. If the `WHERE` clause contains range predicates (described in Section 7.2.1, “`EXPLAIN` Syntax (Get Information About a `SELECT`)”, under the `range` join type), a loose index scan looks up the first key of each group that satisfies the range conditions, and again reads the least possible number of keys. This is possible under the following conditions:

- The query is over a single table.
- The `GROUP BY` includes the first consecutive parts of the index (if instead of `GROUP BY`, the query has a `DISTINCT` clause, then all distinct attributes refer to the beginning of the index).
- The only aggregate functions used (if any) are `MIN()` and `MAX()`, and all of them refer to the same column.
- Any other index parts than the ones from `GROUP BY` referenced in the query must be constants (that is, they must be referenced in equalities with constants), except for the argument of `MIN()` or `MAX()` functions.

The `EXPLAIN` output for such queries shows `Using index for group-by` in the `Extra` column.

The following queries provide several examples that fall into this category, assuming there is an index `idx(c1, c2, c3)` on table `t1(c1,c2,c3,c4)`:

```
SELECT c1, c2 FROM t1 GROUP BY c1, c2;
SELECT DISTINCT c1, c2 FROM t1;
SELECT c1, MIN(c2) FROM t1 GROUP BY c1;
SELECT c1, c2 FROM t1 WHERE c1 < const GROUP BY c1, c2;
SELECT MAX(c3), MIN(c3), c1, c2 FROM t1 WHERE c2 > const GROUP BY c1, c2;
SELECT c2 FROM t1 WHERE c1 < const GROUP BY c1, c2;
SELECT c1, c2 FROM t1 WHERE c3 = const GROUP BY c1, c2;
```

The following queries cannot be executed with this quick select method, for the reasons given:

- There are other aggregate function than `MIN()` or `MAX()`:

  ```
  SELECT c1, SUM(c2) FROM t1 GROUP BY c1;
  ```

- The fields in `GROUP BY` do not refer to the beginning of the index:

  ```
  SELECT c1,c2 FROM t1 GROUP BY c2, c3;
  ```

- The query refers to a key part that is after the `GROUP BY` parts, and for which there is no equality with a constant:

  ```
  SELECT c1,c3 FROM t1 GROUP BY c1, c2;
  ```

7.2.11.2. Tight index scan

A tight index scan may be either a full index scan or a range index scan, depending on the query conditions.

When the conditions for a loose index scan are not met, it is still possible to avoid creation of temporary tables for `GROUP BY` queries. If there are range conditions in the `WHERE` clause, this method reads only the keys that satisfy these conditions. Otherwise, it performs an index scan. Since this method reads all keys in each range defined by the `WHERE` clause, or scans the whole index if there are no range conditions, we term it a ``tight index scan.'' Notice that with a tight index scan, the grouping operation is performed after all keys that satisfy the range conditions have been found.

For this method to work, it is sufficient that for all columns in a query referring to key parts before or in between the `GROUP BY` key parts, there is a constant equality condition. The constants from the equality conditions fill in the ``gaps'' in the search keys so that it is possible to form complete prefixes of the index. Then these index prefixes can be used for index lookups. If we require sorting of the `GROUP BY` result, and it is possible to form search keys that are prefixes of the index, MySQL also avoids sorting because searching with prefixes in an ordered index retrieves all keys in order.

The following queries do not work with the first method above, but still work with the second index access method (assuming we have the aforementioned index `idx` on table `t1`):

- There is a ``gap'' in `GROUP BY`, but it is covered by the condition (c2 = 'a').

  ```
  SELECT c1, c2, c3 FROM t1 WHERE c2 = 'a' GROUP BY c1, c3;
  ```

- `GROUP BY` does not begin from the first key part, but there is a condition that provides a constant for that key part:

  ```
  SELECT c1, c2, c3 FROM t1 WHERE c1 = 'a' GROUP BY c2, c3;
  ```

7.2.12. How MySQL Optimizes `LIMIT`

In some cases, MySQL handles a query differently when you are using `LIMIT` *`row_count`* and not using `HAVING`:

- If you are selecting only a few rows with `LIMIT`, MySQL uses indexes in some cases when normally it would prefer to do a full table scan.
- If you use `LIMIT` *`row_count`* with `ORDER BY`, MySQL ends the sorting as soon as it has found the first *`row_count`* lines rather than sorting the whole table.
- When combining `LIMIT` *`row_count`* with `DISTINCT`, MySQL stops as soon as it finds *`row_count`* unique rows.
- In some cases, a `GROUP BY` can be resolved by reading the key in order (or doing a sort on the key) and then calculating summaries until the key value changes. In this case, `LIMIT` *`row_count`* does not calculate any unnecessary `GROUP BY` values.
- As soon as MySQL has sent the required number of rows to the client, it aborts the query unless you are using `SQL_CALC_FOUND_ROWS`.
- `LIMIT 0` always quickly returns an empty set. This is useful to check the query or to get the column types of the result columns.
- When the server uses temporary tables to resolve the query, the `LIMIT` *`row_count`* is used to calculate how much space is required.

7.2.13. How to Avoid Table Scans

The output from `EXPLAIN` shows `ALL` in the `type` column when MySQL uses a table scan to resolve a query. This usually happens under the following conditions:

- The table is so small that it's faster to do a table scan than a key lookup. This is a common case for tables with fewer than 10 rows and a short row length.
- There are no usable restrictions in the `ON` or `WHERE` clause for indexed columns.
- You are comparing indexed columns with constant values and MySQL has calculated (based on the index tree) that the constants cover too large a part of the table and that a table scan would be faster. See Section 7.2.4, “How MySQL Optimizes `WHERE` Clauses”.
- You are using a key with low cardinality (many rows match the key value) through another column. In this case, MySQL assumes that by using the key it probably does a lot of key lookups and that a table scan would be faster.

For small tables, a table scan often is appropriate. For large tables, try the following techniques to avoid having the optimizer incorrectly choose a table scan:

- Use `ANALYZE TABLE` *`tbl_name`* to update the key distributions for the scanned table. See Section 13.5.2.1, “`ANALYZE TABLE` Syntax”.
- Use `FORCE INDEX` for the scanned table to tell MySQL that table scans are very expensive compared to using the given index. See Section 13.1.7, “`SELECT` Syntax”.

  ```
  SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
  WHERE t1.col_name=t2.col_name;
  ```

- Start **mysqld** with the `--max-seeks-for-key=1000` option or use `SET max_seeks_for_key=1000` to tell the optimizer to assume that no key scan causes more than 1,000 key seeks. See Section 5.2.3, “Server System Variables”.

7.2.14. Speed of `INSERT` Statements

The time to insert a record is determined by the following factors, where the numbers indicate approximate proportions:

- Connecting: (3)
- Sending query to server: (2)
- Parsing query: (2)
- Inserting record: (1 x size of record)
- Inserting indexes: (1 x number of indexes)
- Closing: (1)

This does not take into consideration the initial overhead to open tables, which is done once for each concurrently running query.

The size of the table slows down the insertion of indexes by log N, assuming B-tree indexes.

You can use the following methods to speed up inserts:

- If you are inserting many rows from the same client at the same time, use `INSERT` statements with multiple `VALUES` lists to insert several rows at a time. This is much faster (many times faster in some cases) than using separate single-row `INSERT` statements. If you are adding data to a non-empty table, you may tune the `bulk_insert_buffer_size` variable to make it even faster. See Section 5.2.3, "Server System Variables".

- If you are inserting a lot of rows from different clients, you can get higher speed by using the `INSERT DELAYED` statement. See Section 13.1.4, "`INSERT` Syntax".

- With `MyISAM` tables you can insert rows at the same time that `SELECT` statements are running if there are no deleted rows in the tables.

- When loading a table from a text file, use `LOAD DATA INFILE`. This is usually 20 times faster than using a lot of `INSERT` statements. See Section 13.1.5, "`LOAD DATA INFILE` Syntax".

- With some extra work, it is possible to make `LOAD DATA INFILE` run even faster when the table has many indexes. Use the following procedure:

  1. Optionally create the table with `CREATE TABLE`.

  2. Execute a `FLUSH TABLES` statement or a **mysqladmin flush-tables** command.

  3. Use **myisamchk --keys-used=0 -rq** ***/path/to/db/tbl_name.*** This removes all use of indexes for the table.

  4. Insert data into the table with `LOAD DATA INFILE`. This does not update any indexes and therefore is very fast.

  5. If you are going to only read the table in the future, use **myisampack** to make it smaller. See Section 14.1.3.3, "Compressed Table Characteristics".

  6. Re-create the indexes with **myisamchk -r -q** ***/path/to/db/tbl_name***. This creates the index tree in memory before writing it to disk, which is much faster because it avoids lots of disk seeks. The resulting index tree is also perfectly balanced.

  7. Execute a `FLUSH TABLES` statement or a **mysqladmin flush-tables** command.

  Note that `LOAD DATA INFILE` also performs the preceding optimization if you insert into an empty `MyISAM` table; the main difference is that you can let **myisamchk** allocate much more temporary memory for the index creation than you might want the server to allocate for index re-creation when it executes the `LOAD DATA INFILE` statement.

  As of MySQL 4.0, you can also use `ALTER TABLE` *`tbl_name`* `DISABLE KEYS` instead of **myisamchk --keys-used=0 -rq** ***/path/to/db/tbl_name*** and `ALTER TABLE` *`tbl_name`* `ENABLE KEYS` instead of **myisamchk -r -q** ***/path/to/db/tbl_name***. This way you can also skip the `FLUSH TABLES` steps.

- You can speed up `INSERT` operations that are done with multiple statements by locking your tables:

  ```
  LOCK TABLES a WRITE;
  INSERT INTO a VALUES (1,23),(2,34),(4,33);
  INSERT INTO a VALUES (8,26),(6,29);
  UNLOCK TABLES;
  ```

  A performance benefit occurs because the index buffer is flushed to disk only once, after all `INSERT` statements have completed. Normally there would be as many index buffer flushes as there are different `INSERT` statements. Explicit locking statements are not needed if you can insert all rows with a single statement.

For transactional tables, you should use BEGIN/COMMIT instead of LOCK TABLES to get a speedup.

Locking also lowers the total time of multiple-connection tests, although the maximum wait time for individual connections might go up because they wait for locks. For example:

```
Connection 1 does 1000 inserts
Connections 2, 3, and 4 do 1 insert
Connection 5 does 1000 inserts
```

If you don't use locking, connections 2, 3, and 4 finish before 1 and 5. If you use locking, connections 2, 3, and 4 probably do not finish before 1 or 5, but the total time should be about 40% faster.

INSERT, UPDATE, and DELETE operations are very fast in MySQL, but you can obtain better overall performance by adding locks around everything that does more than about five inserts or updates in a row. If you do very many inserts in a row, you could do a LOCK TABLES followed by an UNLOCK TABLES once in a while (about each 1,000 rows) to allow other threads access to the table. This would still result in a nice performance gain.

INSERT is still much slower for loading data than LOAD DATA INFILE, even when using the strategies just outlined.

- To get some more speed for MyISAM tables, for both LOAD DATA INFILE and INSERT, enlarge the key cache by increasing the key_buffer_size system variable. See Section 7.5.2, “Tuning Server Parameters”.

7.2.15. Speed of UPDATE Statements

Update statements are optimized as a SELECT query with the additional overhead of a write. The speed of the write depends on the amount of data being updated and the number of indexes that are updated. Indexes that are not changed does not get updated.

Also, another way to get fast updates is to delay updates and then do many updates in a row later. Doing many updates in a row is much quicker than doing one at a time if you lock the table.

Note that for a MyISAM table that uses dynamic record format, updating a record to a longer total length may split the record. If you do this often, it is very important to use OPTIMIZE TABLE occasionally. See Section 13.5.2.5, “OPTIMIZE TABLE Syntax”.

7.2.16. Speed of DELETE Statements

The time to delete individual records is exactly proportional to the number of indexes. To delete records more quickly, you can increase the size of the key cache. See Section 7.5.2, “Tuning Server Parameters”.

If you want to delete all rows in the table, use TRUNCATE TABLE tbl_name rather than DELETE FROM *tbl_name*. See Section 13.1.9, “TRUNCATE Syntax”.

7.2.17. Other Optimization Tips

This section lists a number of miscellaneous tips for improving query processing speed:

- Use persistent connections to the database to avoid connection overhead. If you can't use persistent

connections and you are initiating many new connections to the database, you may want to change the value of the `thread_cache_size` variable. See Section 7.5.2, “Tuning Server Parameters”.

- Always check whether all your queries really use the indexes you have created in the tables. In MySQL, you can do this with the `EXPLAIN` statement. See Section 7.2.1, “`EXPLAIN` Syntax (Get Information About a `SELECT`)”.

- Try to avoid complex `SELECT` queries on `MyISAM` tables that are updated frequently, to avoid problems with table locking that occur due to contention between readers and writers.

- With `MyISAM` tables that have no deleted rows, you can insert rows at the end at the same time that another query is reading from the table. If this is important for you, you should consider using the table in ways that avoid deleting rows. Another possibility is to run `OPTIMIZE TABLE` after you have deleted a lot of rows.

- Use `ALTER TABLE ... ORDER BY` *`expr1, expr2, ...`* if you mostly retrieve rows in *`expr1, expr2, ...`* order. By using this option after extensive changes to the table, you may be able to get higher performance.

- In some cases, it may make sense to introduce a column that is ``hashed'' based on information from other columns. If this column is short and reasonably unique, it may be much faster than a big index on many columns. In MySQL, it's very easy to use this extra column:

  ```
  SELECT * FROM tbl_name
      WHERE hash_col=MD5(CONCAT(col1,col2))
      AND col1='constant' AND col2='constant';
  ```

- For `MyISAM` tables that change a lot, you should try to avoid all variable-length columns (`VARCHAR`, `BLOB`, and `TEXT`). The table uses dynamic record format if it includes even a single variable-length column. See Chapter 14, *MySQL Storage Engines and Table Types*.

- It's normally not useful to split a table into different tables just because the rows get ``big.'' To access a row, the biggest performance hit is the disk seek to find the first byte of the row. After finding the data, most modern disks can read the whole row fast enough for most applications. The only cases where it really matters to split up a table is if it's a `MyISAM` table with dynamic record format (see above) that you can change to a fixed record size, or if you very often need to scan the table but do not need most of the columns. See Chapter 14, *MySQL Storage Engines and Table Types*.

- If you very often need to calculate results such as counts based on information from a lot of rows, it's probably much better to introduce a new table and update the counter in real time. An update of the following form is very fast:

  ```
  UPDATE tbl_name SET count_col=count_col+1 WHERE key_col=constant;
  ```

  This is really important when you use MySQL storage engines such as `MyISAM` and `ISAM` that have only table-level locking (multiple readers / single writers). This also gives better performance with most databases, because the row locking manager in this case has less to do.

- If you need to collect statistics from large log tables, use summary tables instead of scanning the entire log table. Maintaining the summaries should be much faster than trying to calculate statistics ``live.'' It's much faster to regenerate new summary tables from the logs when things change (depending on business decisions) than to have to change the running application!

- If possible, you should classify reports as ``live'' or ``statistical,'' where data needed for statistical reports is created only from summary tables that are generated periodically from the live data.

- Take advantage of the fact that columns have default values. Insert values explicitly only when the

value to be inserted differs from the default. This reduces the parsing that MySQL needs to do and improves the insert speed.

- In some cases, it's convenient to pack and store data into a `BLOB` column. In this case, you must add some extra code in your application to pack and unpack information in the `BLOB` values, but this may save a lot of accesses at some stage. This is practical when you have data that doesn't conform to a rows-and-columns table structure.

- Normally, you should try to keep all data non-redundant (what is called "third normal form" in database theory). However, do not be afraid to duplicate information or create summary tables if necessary to gain more speed.

- Stored procedures or UDFs (user-defined functions) may be a good way to get more performance for some tasks. However, if you use a database system that does not support these capabilities, you should always have another way to perform the same tasks, even if the alternative method is slower.

- You can always gain something by caching queries or answers in your application and then performing many inserts or updates together. If your database supports table locks (like MySQL and Oracle), this should help to ensure that the index cache is only flushed once after all updates.

- Use `INSERT DELAYED` when you do not need to know when your data is written. This speeds things up because many records can be written with a single disk write.

- Use `INSERT LOW_PRIORITY` when you want to give `SELECT` statements higher priority than your inserts.

- Use `SELECT HIGH_PRIORITY` to get retrievals that jump the queue. That is, the `SELECT` is done even if there is another client waiting to do a write.

- Use multiple-row `INSERT` statements to store many rows with one SQL statement (many SQL servers support this).

- Use `LOAD DATA INFILE` to load large amounts of data. This is faster than using `INSERT` statements.

- Use `AUTO_INCREMENT` columns to generate unique values.

- Use `OPTIMIZE TABLE` once in a while to avoid fragmentation with `MyISAM` tables when using a dynamic table format. See Section 14.1.3, “`MyISAM` Table Storage Formats”.

- Use `HEAP` tables when possible to get more speed. See Chapter 14, *MySQL Storage Engines and Table Types*.

- When using a normal Web server setup, images should be stored as files. That is, store only a file reference in the database. The main reason for this is that a normal Web server is much better at caching files than database contents, so it's much easier to get a fast system if you are using files.

- Use in-memory tables for non-critical data that is accessed often, such as information about the last displayed banner for users who don't have cookies enabled in their Web browser.

- Columns with identical information in different tables should be declared to have identical data types. Before MySQL 3.23, you get slow joins otherwise.

  Try to keep column names simple. For example, in a table named `customer`, use a column name of `name` instead of `customer_name`. To make your names portable to other SQL servers, you should keep them shorter than 18 characters.

- If you need really high speed, you should take a look at the low-level interfaces for data storage that the different SQL servers support! For example, by accessing the MySQL `MyISAM` storage engine

directly, you could get a speed increase of two to five times compared to using the SQL interface. To be able to do this, the data must be on the same server as the application, and usually it should only be accessed by one process (because external file locking is really slow). One could eliminate these problems by introducing low-level `MyISAM` commands in the MySQL server (this could be one easy way to get more performance if needed). By carefully designing the database interface, it should be quite easy to support this types of optimization.

- If you are using numerical data, it's faster in many cases to access information from a database (using a live connection) than to access a text file. Information in the database is likely to be stored in a more compact format than in the text file, so accessing it involves fewer disk accesses. You also save code in your application because you don't have to parse your text files to find line and column boundaries.

- Replication can provide a performance benefit for some operations. You can distribute client retrievals among replication servers to split up the load. To avoid slowing down the master while making backups, you can make backups using a slave server. See Chapter 6, *Replication in MySQL*.

- Declaring a `MyISAM` table with the `DELAY_KEY_WRITE=1` table option makes index updates faster because they are not flushed to disk until the table is closed. The downside is that if something kills the server while such a table is open, you should ensure that they are okay by running the server with the `--myisam-recover` option, or by running **myisamchk** before restarting the server. (However, even in this case, you should not lose anything by using `DELAY_KEY_WRITE`, because the key information can always be generated from the data rows.)

7.3. Locking Issues

7.3.1. Locking Methods

Currently, MySQL supports table-level locking for `ISAM`, `MyISAM`, and `MEMORY` (`HEAP`) tables, page-level locking for `BDB` tables, and row-level locking for `InnoDB` tables.

In many cases, you can make an educated guess about which locking type is best for an application, but generally it's very hard to say that a given lock type is better than another. Everything depends on the application and different parts of an application may require different lock types.

To decide whether you want to use a storage engine with row-level locking, you should look at what your application does and what mix of select and update statements it uses. For example, most Web applications do lots of selects, very few deletes, updates based mainly on key values, and inserts into some specific tables. The base MySQL `MyISAM` setup is very well tuned for this.

Table locking in MySQL is deadlock-free for storage engines that use table-level locking. Deadlock avoidance is managed by always requesting all needed locks at once at the beginning of a query and always locking the tables in the same order.

The table-locking method MySQL uses for `WRITE` locks works as follows:

- If there are no locks on the table, put a write lock on it.

- Otherwise, put the lock request in the write lock queue.

The table-locking method MySQL uses for `READ` locks works as follows:

- If there are no write locks on the table, put a read lock on it.

- Otherwise, put the lock request in the read lock queue.

When a lock is released, the lock is made available to the threads in the write lock queue, then to the threads in the read lock queue.

This means that if you have many updates for a table, `SELECT` statements wait until there are no more updates.

Starting in MySQL 3.23.33, you can analyze the table lock contention on your system by checking the `Table_locks_waited` and `Table_locks_immediate` status variables:

```
mysql> SHOW STATUS LIKE 'Table%';
+-----------------------+---------+
| Variable_name         | Value   |
+-----------------------+---------+
| Table_locks_immediate | 1151552 |
| Table_locks_waited    | 15324   |
+-----------------------+---------+
```

As of MySQL 3.23.7 (3.23.25 for Windows), you can freely mix concurrent `INSERT` and `SELECT` statements for a `MyISAM` table without locks if the `INSERT` statements are non-conflicting. That is, you can insert rows into a `MyISAM` table at the same time other clients are reading from it. No conflict occurs if the data file contains no free blocks in the middle, because in that case, records always are inserted at the end of the data file. (Holes can result from rows having been deleted from or updated in the middle of the table.) If there are holes, concurrent inserts are re-enabled automatically when all holes have been filled with new data.

If you want to do many `INSERT` and `SELECT` operations on a table when concurrent inserts are not possible, you can insert rows in a temporary table and update the real table with the records from the temporary table once in a while. This can be done with the following code:

```
mysql> LOCK TABLES real_table WRITE, insert_table WRITE;
mysql> INSERT INTO real_table SELECT * FROM insert_table;
mysql> TRUNCATE TABLE insert_table;
mysql> UNLOCK TABLES;
```

`InnoDB` uses row locks and `BDB` uses page locks. For the `InnoDB` and `BDB` storage engines, deadlock is possible. This is because `InnoDB` automatically acquires row locks and `BDB` acquires page locks during the processing of SQL statements, not at the start of the transaction.

Advantages of row-level locking:

- Fewer lock conflicts when accessing different rows in many threads.
- Fewer changes for rollbacks.
- Makes it possible to lock a single row a long time.

Disadvantages of row-level locking:

- Takes more memory than page-level or table-level locks.
- Is slower than page-level or table-level locks when used on a large part of the table because you must acquire many more locks.

- Is definitely much worse than other locks if you often do `GROUP BY` operations on a large part of the data or if you often must scan the entire table.
- With higher-level locks, you can also more easily support locks of different types to tune the application, because the lock overhead is less than for row-level locks.

Table locks are superior to page-level or row-level locks in the following cases:

- Most statements for the table are reads.
- Read and updates on strict keys, where you update or delete a row that can be fetched with a single key read:

  ```
  UPDATE tbl_name SET column=value WHERE unique_key_col=key_value;
  DELETE FROM tbl_name WHERE unique_key_col=key_value;
  ```

- `SELECT` combined with concurrent `INSERT` statements, and very few `UPDATE` and `DELETE` statements.
- Many scans or `GROUP BY` operations on the entire table without any writers.

Options other than row-level or page-level locking:

Versioning (such as we use in MySQL for concurrent inserts) where you can have one writer at the same time as many readers. This means that the database/table supports different views for the data depending on when you started to access it. Other names for this are time travel, copy on write, or copy on demand.

Copy on demand is in many cases much better than page-level or row-level locking. However, the worst case does use much more memory than when using normal locks.

Instead of using row-level locks, you can use application-level locks, such as `GET_LOCK()` and `RELEASE_LOCK()` in MySQL. These are advisory locks, so they work only in well-behaved applications.

7.3.2. Table Locking Issues

To achieve a very high lock speed, MySQL uses table locking (instead of page, row, or column locking) for all storage engines except `InnoDB` and `BDB`.

For `InnoDB` and `BDB` tables, MySQL only uses table locking if you explicitly lock the table with `LOCK TABLES`. For these table types, we recommend you to not use `LOCK TABLES` at all, because `InnoDB` uses automatic row-level locking and `BDB` uses page-level locking to ensure transaction isolation.

For large tables, table locking is much better than row locking for most applications, but there are some pitfalls.

Table locking enables many threads to read from a table at the same time, but if a thread wants to write to a table, it must first get exclusive access. During the update, all other threads that want to access this particular table must wait until the update is done.

Table updates normally are considered to be more important than table retrievals, so they are given higher priority. This should ensure that updates to a table are not ``starved'' even if there is heavy `SELECT` activity for the table.

Table locking causes problems in cases such as when a thread is waiting because the disk is full and free space needs to become available before the thread can proceed. In this case, all threads that want to ac-

cess the problem table are also put in a waiting state until more disk space is made available.

Table locking is also disadvantageous under the following scenario:

- A client issues a SELECT that takes a long time to run.
- Another client then issues an UPDATE on the same table. This client waits until the SELECT is finished.
- Another client issues another SELECT statement on the same table. Because UPDATE has higher priority than SELECT, this SELECT waits for the UPDATE to finish. It also waits for the first SELECT to finish!

The following list describes some ways to avoid or reduce contention caused by table locking:

- Try to get the SELECT statements to run faster. You might have to create some summary tables to do this.
- Start **mysqld** with --low-priority-updates. This gives all statements that update (modify) a table lower priority than SELECT statements. In this case, the second SELECT statement in the preceding scenario would execute before the INSERT statement, and would not need to wait for the first SELECT to finish.
- You can specify that all updates issued in a specific connection should be done with low priority by using the SET LOW_PRIORITY_UPDATES=1 statement. See Section 13.5.3, "SET Syntax".
- You can give a specific INSERT, UPDATE, or DELETE statement lower priority with the LOW_PRIORITY attribute.
- You can give a specific SELECT statement higher priority with the HIGH_PRIORITY attribute. See Section 13.1.7, "SELECT Syntax".
- Starting from MySQL 3.23.7, you can start **mysqld** with a low value for the max_write_lock_count system variable to force MySQL to temporarily elevate the priority of all SELECT statements that are waiting for a table after a specific number of inserts to the table occur. This allows READ locks after a certain number of WRITE locks.
- If you have problems with INSERT combined with SELECT, switch to using MyISAM tables, which support concurrent SELECT and INSERT statements.
- If you mix inserts and deletes on the same table, INSERT DELAYED may be of great help. See Section 13.1.4.2, "INSERT DELAYED Syntax".
- If you have problems with mixed SELECT and DELETE statements, the LIMIT option to DELETE may help. See Section 13.1.1, "DELETE Syntax".
- Using SQL_BUFFER_RESULT with SELECT statements can help to make the duration of table locks shorter. See Section 13.1.7, "SELECT Syntax".
- You could change the locking code in mysys/thr_lock.c to use a single queue. In this case, write locks and read locks would have the same priority, which might help some applications.

Here are some tips about table locking in MySQL:

- Concurrent users are not a problem if you don't mix updates with selects that need to examine many

rows in the same table.

- You can use `LOCK TABLES` to speed up things (many updates within a single lock is much faster than updates without locks). Splitting table contents into separate tables may also help.
- If you encounter speed problems with table locks in MySQL, you may be able to improve performance by converting some of your tables to `InnoDB` or `BDB` tables. See Chapter 15, *The `InnoDB` Storage Engine*. See Section 14.4, “The `BDB` (`BerkeleyDB`) Storage Engine”.

7.4. Optimizing Database Structure

7.4.1. Design Choices

MySQL keeps row data and index data in separate files. Many (almost all) other databases mix row and index data in the same file. We believe that the MySQL choice is better for a very wide range of modern systems.

Another way to store the row data is to keep the information for each column in a separate area (examples are SDBM and Focus). This causes a performance hit for every query that accesses more than one column. Because this degenerates so quickly when more than one column is accessed, we believe that this model is not good for general-purpose databases.

The more common case is that the index and data are stored together (as in Oracle/Sybase, et al). In this case, you find the row information at the leaf page of the index. The good thing with this layout is that it, in many cases, depending on how well the index is cached, saves a disk read. The bad things with this layout are:

- Table scanning is much slower because you have to read through the indexes to get at the data.
- You can't use only the index table to retrieve data for a query.
- You use more space because you must duplicate indexes from the nodes (you can't store the row in the nodes).
- Deletes degenerate the table over time (because indexes in nodes are usually not updated on delete).
- It's harder to cache only the index data.

7.4.2. Make Your Data as Small as Possible

One of the most basic optimizations is to design your tables to take as little space on the disk as possible. This can give huge improvements because disk reads are faster, and smaller tables normally require less main memory while their contents are being actively processed during query execution. Indexing also is a lesser resource burden if done on smaller columns.

MySQL supports a lot of different table types and row formats. For each table, you can decide which storage/index method to use. Choosing the right table format for your application may give you a big performance gain. See Chapter 14, *MySQL Storage Engines and Table Types*.

You can get better performance on a table and minimize storage space using the techniques listed here:

- Use the most efficient (smallest) data types possible. MySQL has many specialized types that save disk space and memory.

- Use the smaller integer types if possible to get smaller tables. For example, `MEDIUMINT` is often better than `INT`.
- Declare columns to be `NOT NULL` if possible. It makes everything faster and you save one bit per column. If you really need `NULL` in your application, you should definitely use it. Just avoid having it on all columns by default.
- For `MyISAM` tables, if you don't have any variable-length columns (`VARCHAR`, `TEXT`, or `BLOB` columns), a fixed-size record format is used. This is faster but unfortunately may waste some space. See Section 14.1.3, "`MyISAM` Table Storage Formats". You can hint that you want to have fixed length rows even if you have `VARCHAR` columns with the `CREATE` option `ROW_FORMAT=fixed`.
- Starting with MySQL/InnoDB 5.0.3, `InnoDB` tables use a more compact storage format. In earlier versions of MySQL, InnoDB records contain some redundant information, such as the number of columns and the length of each column, even for fixed-size columns. By default, tables are created in the compact format (`ROW_FORMAT=COMPACT`). If you wish to downgrade to older versions of MySQL/InnoDB, you can request the old format with `ROW_FORMAT=REDUNDANT`.

  The compact InnoDB format also changes the way how `CHAR` columns containing UTF-8 data are stored. In the `ROW_FORMAT=REDUNDANT` format, a UTF-8 `CHAR(`*`n`*`)` occupies 3**n* bytes, given that the maximum length of a UTF-8 encoded character is 3 bytes. Since many languages can be written mostly with single-byte UTF-8 characters, a fixed storage length often wastes space. The `ROW_FORMAT=COMPACT` format allocates a variable amount of *n*..3**n* bytes for these columns by stripping trailing spaces if necessary. The minimum storage length is kept as *n* bytes in order to facilitate in-place updates in typical cases.
- The primary index of a table should be as short as possible. This makes identification of each row easy and efficient.
- Create only the indexes that you really need. Indexes are good for retrieval but bad when you need to store things fast. If you mostly access a table by searching on a combination of columns, make an index on them. The first index part should be the most used column. If you are *always* using many columns, you should use the column with more duplicates first to get better compression of the index.
- If it's very likely that a column has a unique prefix on the first number of characters, it's better to index only this prefix. MySQL supports an index on the leftmost part of a character column. Shorter indexes are faster not only because they take less disk space, but also because they give you more hits in the index cache and thus fewer disk seeks. See Section 7.5.2, "Tuning Server Parameters".
- In some circumstances, it can be beneficial to split into two a table that is scanned very often. This is especially true if it is a dynamic format table and it is possible to use a smaller static format table that can be used to find the relevant rows when scanning the table.

7.4.3. Column Indexes

All MySQL column types can be indexed. Use of indexes on the relevant columns is the best way to improve the performance of `SELECT` operations.

The maximum number of indexes per table and the maximum index length is defined per storage engine. See Chapter 14, *MySQL Storage Engines and Table Types*. All storage engines support at least 16 indexes per table and a total index length of at least 256 bytes. Most storage engines have higher limits.

With `col_name(length)` syntax in an index specification, you can create an index that uses only the first `length` characters of a `CHAR` or `VARCHAR` column. Indexing only a prefix of column values like this can make the index file much smaller.

The MyISAM and (as of MySQL 4.0.14) InnoDB storage engines also support indexing on BLOB and TEXT columns. When indexing a BLOB or TEXT column, you *must* specify a prefix length for the index. For example:

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

Prefixes can be up to 255 bytes long (or 1000 bytes for MyISAM and InnoDB tables as of MySQL 4.1.2). Note that prefix limits are measured in bytes, whereas the prefix length in CREATE TABLE statements is interpreted as number of characters. Take this into account when specifying a prefix length for a column that uses a multi-byte character set.

As of MySQL 3.23.23, you can also create FULLTEXT indexes. They are used for full-text searches. Only the MyISAM table type supports FULLTEXT indexes and only for CHAR, VARCHAR, and TEXT columns. Indexing always happens over the entire column and partial (prefix) indexing is not supported. See Section 12.6, “Full-Text Search Functions” for details.

As of MySQL 4.1.0, you can create indexes on spatial column types. Currently, spatial types are supported only by the MyISAM storage engine. Spatial indexes use R-trees.

The MEMORY (HEAP) storage engine uses hash indexes by default. It also supports B-tree indexes as of MySQL 4.1.0.

7.4.4. Multiple-Column Indexes

MySQL can create indexes on multiple columns. An index may consist of up to 15 columns. For certain column types, you can index a prefix of the column (see Section 7.4.3, “Column Indexes”).

A multiple-column index can be considered a sorted array containing values that are created by concatenating the values of the indexed columns.

MySQL uses multiple-column indexes in such a way that queries are fast when you specify a known quantity for the first column of the index in a WHERE clause, even if you don't specify values for the other columns.

Suppose that a table has the following specification:

```
CREATE TABLE test (
    id INT NOT NULL,
    last_name CHAR(30) NOT NULL,
    first_name CHAR(30) NOT NULL,
    PRIMARY KEY (id),
    INDEX name (last_name,first_name));
```

The name index is an index over last_name and first_name. The index can be used for queries that specify values in a known range for last_name, or for both last_name and first_name. Therefore, the name index is used in the following queries:

```
SELECT * FROM test WHERE last_name='Widenius';

SELECT * FROM test
    WHERE last_name='Widenius' AND first_name='Michael';

SELECT * FROM test
    WHERE last_name='Widenius'
    AND (first_name='Michael' OR first_name='Monty');

SELECT * FROM test
    WHERE last_name='Widenius'
```

```
    AND first_name >='M' AND first_name < 'N';
```

However, the `name` index is *not* used in the following queries:

```
SELECT * FROM test WHERE first_name='Michael';

SELECT * FROM test
    WHERE last_name='Widenius' OR first_name='Michael';
```

The manner in which MySQL uses indexes to improve query performance is discussed further in the next section.

7.4.5. How MySQL Uses Indexes

Indexes are used to find rows with specific column values fast. Without an index, MySQL has to start with the first record and then read through the whole table to find the relevant rows. The larger the table, the more this costs. If the table has an index for the columns in question, MySQL can quickly determine the position to seek to in the middle of the data file without having to look at all the data. If a table has 1,000 rows, this is at least 100 times faster than reading sequentially. Note that if you need to access almost all 1,000 rows, it is faster to read sequentially, because that minimizes disk seeks.

Most MySQL indexes (`PRIMARY KEY`, `UNIQUE`, `INDEX`, and `FULLTEXT`) are stored in B-trees. Exceptions are that indexes on spatial column types use R-trees, and `MEMORY` (`HEAP`) tables support hash indexes.

Strings are automatically prefix- and end-space compressed. See Section 13.2.5, "`CREATE INDEX` Syntax".

In general, indexes are used as described in the following discussion. Characteristics specific to hash indexes (as used in `MEMORY` tables) are described at the end of this section.

Indexes are used for these operations:

- To quickly find the rows that match a `WHERE` clause.
- To eliminate rows from consideration. If there is a choice between multiple indexes, MySQL normally uses the index that finds the smallest number of rows.
- To retrieve rows from other tables when performing joins.
- To find the `MIN()` or `MAX()` value for a specific indexed column *`key_col`*. This is optimized by a preprocessor that checks whether you are using `WHERE` *`key_part_# = constant`* on all key parts that occur before *`key_col`* in the index. In this case, MySQL does a single key lookup for each `MIN()` or `MAX()` expression and replace it with a constant. If all expressions are replaced with constants, the query returns at once. For example:

  ```
  SELECT MIN(key_part2),MAX(key_part2)
  FROM tbl_name WHERE key_part1=10;
  ```

- To sort or group a table if the sorting or grouping is done on a leftmost prefix of a usable key (for example, `ORDER BY` *`key_part1, key_part2`*). If all key parts are followed by `DESC`, the key is read in reverse order. See Section 7.2.10, "How MySQL Optimizes `ORDER BY`".
- In some cases, a query can be optimized to retrieve values without consulting the data rows. If a query uses only columns from a table that are numeric and that form a leftmost prefix for some key,

the selected values may be retrieved from the index tree for greater speed:

```
SELECT key_part3 FROM tbl_name WHERE key_part1=1
```

Suppose that you issue the following SELECT statement:

```
mysql> SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;
```

If a multiple-column index exists on col1 and col2, the appropriate rows can be fetched directly. If separate single-column indexes exist on col1 and col2, the optimizer tries to find the most restrictive index by deciding which index finds fewer rows and using that index to fetch the rows.

If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to find rows. For example, if you have a three-column index on (col1, col2, col3), you have indexed search capabilities on (col1), (col1, col2), and (col1, col2, col3).

MySQL can't use a partial index if the columns don't form a leftmost prefix of the index. Suppose that you have the SELECT statements shown here:

```
SELECT * FROM tbl_name WHERE col1=val1;
SELECT * FROM tbl_name WHERE col2=val2;
SELECT * FROM tbl_name WHERE col2=val2 AND col3=val3;
```

If an index exists on (col1, col2, col3), only the first of the preceding queries uses the index. The second and third queries do involve indexed columns, but (col2) and (col2, col3) are not leftmost prefixes of (col1, col2, col3).

A B-tree index can be used for column comparisons in expressions that use the =, >, >=, <, <=, or BETWEEN operators. The index also can be used for LIKE comparisons if the argument to LIKE is a constant string that doesn't start with a wildcard character. For example, the following SELECT statements use indexes:

```
SELECT * FROM tbl_name WHERE key_col LIKE 'Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE 'Pat%_ck%';
```

In the first statement, only rows with 'Patrick' <= key_col < 'Patricl' are considered. In the second statement, only rows with 'Pat' <= key_col < 'Pau' are considered.

The following SELECT statements do not use indexes:

```
SELECT * FROM tbl_name WHERE key_col LIKE '%Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE other_col;
```

In the first statement, the LIKE value begins with a wildcard character. In the second statement, the LIKE value is not a constant.

MySQL 4.0 and up performs an additional LIKE optimization. If you use ... LIKE '%string%' and string is longer than three characters, MySQL uses the Turbo Boyer-Moore algorithm to initialize the pattern for the string and then use this pattern to perform the search quicker.

Searching using col_name IS NULL uses indexes if col_name is indexed.

Any index that doesn't span all AND levels in the WHERE clause is not used to optimize the query. In other words, to be able to use an index, a prefix of the index must be used in every AND group.

The following WHERE clauses use indexes:

```
... WHERE index_part1=1 AND index_part2=2 AND other_column=3
    /* index = 1 OR index = 2 */
... WHERE index=1 OR A=10 AND index=2
    /* optimized like "index_part1='hello'" */
... WHERE index_part1='hello' AND index_part3=5
    /* Can use index on index1 but not on index2 or index3 */
... WHERE index1=1 AND index2=2 OR index1=3 AND index3=3;
```

These WHERE clauses do *not* use indexes:

```
    /* index_part1 is not used */
... WHERE index_part2=1 AND index_part3=2
    /* Index is not used in both AND parts */
... WHERE index=1 OR A=10
    /* No index spans all rows  */
... WHERE index_part1=1 OR index_part2=10
```

Sometimes MySQL does not use an index, even if one is available. One way this occurs is when the optimizer estimates that using the index would require MySQL to access a large percentage of the rows in the table. (In this case, a table scan is probably much faster, because it requires fewer seeks.) However, if such a query uses LIMIT to only retrieve part of the rows, MySQL uses an index anyway, because it can much more quickly find the few rows to return in the result.

Hash indexes have somewhat different characteristics than those just discussed:

- They are used only for equality comparisons that use the = or <=> operators (but are *very* fast). They are not used for comparison operators such as < that find a range of values.
- The optimizer cannot use a hash index to speed up ORDER BY operations. (This type of index cannot be used to search for the next entry in order.)
- MySQL cannot determine approximately how many rows there are between two values (this is used by the range optimizer to decide which index to use). This may affect some queries if you change a MyISAM table to a hash-indexed MEMORY table.
- Only whole keys can be used to search for a row. (With a B-tree index, any leftmost prefix of the key can be used to find rows.)

7.4.6. The MyISAM Key Cache

To minimize disk I/O, the MyISAM storage engine employs a strategy that is used by many database management systems. It exploits a cache mechanism to keep the most frequently accessed table blocks in memory:

- For index blocks, a special structure called the key cache (key buffer) is maintained. The structure contains a number of block buffers where the most-used index blocks are placed.
- For data blocks, MySQL uses no special cache. Instead it relies on the native operating system filesystem cache.

This section first describes the basic operation of the MyISAM key cache. Then it discusses changes made in MySQL 4.1 that improve key cache performance and that enable you to better control cache op-

eration:

- Access to the key cache no longer is serialized among threads. Multiple threads can access the cache concurrently.
- You can set up multiple key caches and assign table indexes to specific caches.

The key cache mechanism also is used for `ISAM` tables. However, the significance of this fact is on the wane. `ISAM` table use has been decreasing since MySQL 3.23 when `MyISAM` was introduced. MySQL 4.1 carries this trend further; the `ISAM` storage engine is disabled by default.

You can control the size of the key cache by means of the `key_buffer_size` system variable. If this variable is set equal to zero, no key cache is used. The key cache also is not used if the `key_buffer_size` value is too small to allocate the minimal number of block buffers (8).

When the key cache is not operational, index files are accessed using only the native filesystem buffering provided by the operating system. (In other words, table index blocks are accessed using the same strategy as that employed for table data blocks.)

An index block is a contiguous unit of access to the `MyISAM` index files. Usually the size of an index block is equal to the size of nodes of the index B-tree. (Indexes are represented on disk using a B-tree data structure. Nodes at the bottom of the tree are leaf nodes. Nodes above the leaf nodes are non-leaf nodes.)

All block buffers in a key cache structure are the same size. This size can be equal to, greater than, or less than the size of a table index block. Usually one these two values is a multiple of the other.

When data from any table index block must be accessed, the server first checks whether it is available in some block buffer of the key cache. If it is, the server accesses data in the key cache rather than on disk. That is, it reads from the cache or writes into it rather than reading from or writing to disk. Otherwise, the server chooses a cache block buffer containing a different table index block (or blocks) and replaces the data there by a copy of required table index block. As soon as the new index block is in the cache, the index data can be accessed.

If it happens that a block selected for replacement has been modified, the block is considered ``dirty." In this case, before being replaced, its contents are flushed to the table index from which it came.

Usually the server follows an LRU (Least Recently Used) strategy: When choosing a block for replacement, it selects the least recently used index block. To be able to make such a choice easy, the key cache module maintains a special queue (LRU chain) of all used blocks. When a block is accessed, it is placed at the end of the queue. When blocks need to be replaced, blocks at the beginning of the queue are the least recently used and become the first candidates for eviction.

7.4.6.1. Shared Key Cache Access

Prior to MySQL 4.1, access to the key cache is serialized: No two threads can access key cache buffers simultaneously. The server processes a request for an index block only after it has finished processing the previous request. As a result, a request for an index block not present in any key cache buffer blocks access by other threads while a buffer is being updated to contain the requested index block.

Starting from version 4.1.0, the server supports shared access to the key cache:

- A buffer that is not being updated can be accessed by multiple threads.
- A buffer that is being updated causes threads that need to use it to wait until the update is complete.
- Multiple threads can initiate requests that result in cache block replacements, as long as they do not

interfere with each other (that is, as long as they need different index blocks, and thus cause different cache blocks to be replaced).

Shared access to the key cache allows the server to improve throughput significantly.

7.4.6.2. Multiple Key Caches

Shared access to the key cache improves performance but does not eliminate contention among threads entirely. They still compete for control structures that manage access to the key cache buffers. To reduce key cache access contention further, MySQL 4.1.1 offers the feature of multiple key caches. This allows you to assign different table indexes to different key caches.

When there can be multiple key caches, the server must know which cache to use when processing queries for a given `MyISAM` table. By default, all `MyISAM` table indexes are cached in the default key cache. To assign table indexes to a specific key cache, use the `CACHE INDEX` statement.

For example, the following statement assigns indexes from the tables `t1`, `t2`, and `t3` to the key cache named `hot_cache`:

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+---------+--------------------+----------+----------+
| Table   | Op                 | Msg_type | Msg_text |
+---------+--------------------+----------+----------+
| test.t1 | assign_to_keycache | status   | OK       |
| test.t2 | assign_to_keycache | status   | OK       |
| test.t3 | assign_to_keycache | status   | OK       |
+---------+--------------------+----------+----------+
```

Note: If the server has been built with the `ISAM` storage engine enabled, `ISAM` tables use the key cache mechanism. However, `ISAM` indexes use only the default key cache and cannot be reassigned to a different cache.

The key cache referred to in a `CACHE INDEX` statement can be created by setting its size with a `SET GLOBAL` parameter setting statement or by using server startup options. For example:

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

To destroy a key cache, set its size to zero:

```
mysql> SET GLOBAL keycache1.key_buffer_size=0;
```

Key cache variables are structured system variables that have a name and components. For `keycache1.key_buffer_size`, `keycache1` is the cache variable name and `key_buffer_size` is the cache component. See Section 9.4.1, “Structured System Variables” for a description of the syntax used for referring to structured key cache system variables.

By default, table indexes are assigned to the main (default) key cache created at the server startup. When a key cache is destroyed, all indexes assigned to it are reassigned to the default key cache.

For a busy server, we recommend a strategy that uses three key caches:

- A hot key cache that takes up 20% of the space allocated for all key caches. This is used for tables that are heavily used for searches but that are not updated.
- A cold key cache that takes up 20% of the space allocated for all key caches. This is used for medi-

um-sized intensively modified tables, such as temporary tables.

- A warm key cache that takes up 60% of the key cache space. This is the default key cache, to be used by default for all other tables.

One reason the use of three key caches is beneficial is that access to one key cache structure does not block access to the others. Queries that access tables assigned to one cache do not compete with queries that access tables assigned to another cache. Performance gains occur for other reasons as well:

- The hot cache is used only for retrieval queries, so its contents are never modified. Consequently, whenever an index block needs to be pulled in from disk, the contents of the cache block chosen for replacement need not be flushed first.
- For an index assigned to the hot cache, if there are no queries requiring an index scan, there is a high probability that the index blocks corresponding to non-leaf nodes of the index B-tree remain in the cache.
- An update operation most frequently executed for temporary tables is performed much faster when the updated node is in the cache and need not be read in from disk first. If the size of the indexes of the temporary tables are comparable with the size of cold key cache, the probability is very high that the updated node is in the cache.

`CACHE INDEX` sets up an association between a table and a key cache, but the association is lost each time the server restarts. If you want the association to take effect each time the server starts, one way to accomplish this is to use an option file: Include variable settings that configure your key caches, and an `init-file` option that names a file containing `CACHE INDEX` statements to be executed. For example:

```
key_buffer_size = 4G
hot_cache.key_buffer_size = 2G
cold_cache.key_buffer_size = 2G
init_file=/path/to/data-directory/mysqld_init.sql
```

The statements in `mysqld_init.sql` are executed each time the server starts. It should contain one SQL statement per line. The following example assigns several tables each to `hot_cache` and `cold_cache`:

```
CACHE INDEX a.t1, a.t2, b.t3 IN hot_cache
CACHE INDEX a.t4, b.t5, b.t6 IN cold_cache
```

7.4.6.3. Midpoint Insertion Strategy

By default, the key cache management system of MySQL 4.1 uses the LRU strategy for choosing key cache blocks to be evicted, but it also supports a more sophisticated method called the "midpoint insertion strategy."

When using the midpoint insertion strategy, the LRU chain is divided into two parts: a hot sub-chain and a warm sub-chain. The division point between two parts is not fixed, but the key cache management system takes care that the warm part is not ``too short,'' always containing at least `key_cache_division_limit` percent of the key cache blocks. `key_cache_division_limit` is a component of structured key cache variables, so its value is a parameter that can be set per cache.

When an index block is read from a table into the key cache, it is placed at the end of the warm sub-

chain. After a certain number of hits (accesses of the block), it is promoted to the hot sub-chain. At present, the number of hits required to promote a block (3) is the same for all index blocks. In the future, we will allow the hit count to depend on the B-tree level of the node corresponding to an index block: Fewer hits are required for promotion of an index block if it contains a non-leaf node from the upper levels of the index B-tree than if it contains a leaf node.

A block promoted into the hot sub-chain is placed at the end of the chain. The block then circulates within this sub-chain. If the block stays at the beginning of the sub-chain for a long enough time, it is demoted to the warm chain. This time is determined by the value of the `key_cache_age_threshold` component of the key cache.

The threshold value prescribes that, for a key cache containing *N* blocks, the block at the beginning of the hot sub-chain not accessed within the last *N*`*key_cache_age_threshold/100` hits is to be moved to the beginning of the warm sub-chain. It then becomes the first candidate for eviction, because blocks for replacement always are taken from the beginning of the warm sub-chain.

The midpoint insertion strategy allows you to keep more-valued blocks always in the cache. If you prefer to use the plain LRU strategy, leave the `key_cache_division_limit` value set to its default of 100.

The midpoint insertion strategy helps to improve performance when execution of a query that requires an index scan effectively pushes out of the cache all the index blocks corresponding to valuable high-level B-tree nodes. To avoid this, you must use a midpoint insertion strategy with the `key_cache_division_limit` set to much less than 100. Then valuable frequently hit nodes are preserved in the hot sub-chain during an index scan operation as well.

7.4.6.4. Index Preloading

If there are enough blocks in a key cache to hold blocks of an entire index, or at least the blocks corresponding to its non-leaf nodes, then it makes sense to preload the key cache with index blocks before starting to use it. Preloading allows you to put the table index blocks into a key cache buffer in the most efficient way: by reading the index blocks from disk sequentially.

Without preloading, the blocks are still placed into the key cache as needed by queries. Although the blocks will stay in the cache, because there are enough buffers for all of them, they are fetched from disk in a random order, not sequentially.

To preload an index into a cache, use the `LOAD INDEX INTO CACHE` statement. For example, the following statement preloads nodes (index blocks) of indexes of the tables `t1` and `t2`:

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+---------+--------------+----------+----------+
| Table   | Op           | Msg_type | Msg_text |
+---------+--------------+----------+----------+
| test.t1 | preload_keys | status   | OK       |
| test.t2 | preload_keys | status   | OK       |
+---------+--------------+----------+----------+
```

The `IGNORE LEAVES` modifier causes only blocks for the non-leaf nodes of the index to be preloaded. Thus, the statement shown preloads all index blocks from `t1`, but only blocks for the non-leaf nodes from `t2`.

If an index has been assigned to a key cache using a `CACHE INDEX` statement, preloading places index blocks into that cache. Otherwise, the index is loaded into the default key cache.

7.4.6.5. Key Cache Block Size

MySQL 4.1 introduces a new `key_cache_block_size` variable on a per-key cache basis. This variable specifies the size of the block buffers for a key cache. It is intended to allow tuning of the performance of I/O operations for index files.

The best performance for I/O operations is achieved when the size of read buffers is equal to the size of the native operating system I/O buffers. But setting the size of key nodes equal to the size of the I/O buffer does not always ensure the best overall performance. When reading the big leaf nodes, the server pulls in a lot of unnecessary data, effectively preventing reading other leaf nodes.

Currently, you cannot control the size of the index blocks in a table. This size is set by the server when the `.MYI` index file is created, depending on the size of the keys in the indexes present in the table definition. In most cases, it is set equal to the I/O buffer size. In the future, this will be changed and then the `key_cache_block_size` variable is fully employed.

7.4.6.6. Restructuring a Key Cache

A key cache can be restructured at any time by updating its parameter values. For example:

```
mysql> SET GLOBAL cold_cache.key_buffer_size=4*1024*1024;
```

If you assign to either the `key_buffer_size` or `key_cache_block_size` key cache component a value that differs from the component's current value, the server destroys the cache's old structure and creates a new one based on the new values. If the cache contains any dirty blocks, the server saves them to disk before destroying and re-creating the cache. Restructuring does not occur if you set other key cache parameters.

When restructuring a key cache, the server first flushes the contents of any dirty buffers to disk. After that, the cache contents become unavailable. However, restructuring does not block queries that need to use indexes assigned to the cache. Instead, the server directly accesses the table indexes using native filesystem caching. Filesystem caching is not as efficient as using a key cache, so although queries execute, a slowdown can be anticipated. Once the cache has been restructured, it becomes available again for caching indexes assigned to it, and the use of filesystem caching for the indexes ceases.

7.4.7. How MySQL Counts Open Tables

When you execute a **mysqladmin status** command, you'll see something like this:

```
Uptime: 426 Running threads: 1 Questions: 11082
Reloads: 1 Open tables: 12
```

The `Open tables` value of 12 can be somewhat puzzling if you have only six tables.

MySQL is multi-threaded, so there may be many clients issuing queries for a given table simultaneously. To minimize the problem with multiple client threads having different states on the same table, the table is opened independently by each concurrent thread. This takes some memory but normally increases performance. With `MyISAM` tables, one extra file descriptor is required for the data file for each client that has the table open. (By contrast, the index file descriptor is shared between all threads.) The `ISAM` storage engine shares this behavior.

You can read more about this topic in the next section. See Section 7.4.8, “How MySQL Opens and Closes Tables”.

7.4.8. How MySQL Opens and Closes Tables

The `table_cache`, `max_connections`, and `max_tmp_tables` system variables affect the max-

imum number of files the server keeps open. If you increase one or more of these values, you may run up against a limit imposed by your operating system on the per-process number of open file descriptors. Many operating systems allow you to increase the open-files limit, although the method varies widely from system to system. Consult your operating system documentation to determine whether it is possible to increase the limit and how to do so.

`table_cache` is related to `max_connections`. For example, for 200 concurrent running connections, you should have a table cache size of at least `200 * N`, where `N` is the maximum number of tables in a join. You also need to reserve some extra file descriptors for temporary tables and files.

Make sure that your operating system can handle the number of open file descriptors implied by the `table_cache` setting. If `table_cache` is set too high, MySQL may run out of file descriptors and refuse connections, fail to perform queries, and be very unreliable. You also have to take into account that the `MyISAM` storage engine needs two file descriptors for each unique open table. You can increase the number of file descriptors available for MySQL with the `--open-files-limit` startup option to **mysqld_safe**. See Section A.2.17, “File Not Found”.

The cache of open tables is kept at a level of `table_cache` entries. The default value is 64; this can be changed with the `--table_cache` option to **mysqld**. Note that MySQL may temporarily open even more tables to be able to execute queries.

An unused table is closed and removed from the table cache under the following circumstances:

- When the cache is full and a thread tries to open a table that is not in the cache.
- When the cache contains more than `table_cache` entries and a thread is no longer using a table.
- When a table flushing operation occurs. This happens when someone issues a `FLUSH TABLES` statement or executes a **mysqladmin flush-tables** or **mysqladmin refresh** command.

When the table cache fills up, the server uses the following procedure to locate a cache entry to use:

- Tables that are not currently in use are released, in least recently used order.
- If a new table needs to be opened, but the cache is full and no tables can be released, the cache is temporarily extended as necessary.

When the cache is in a temporarily extended state and a table goes from a used to unused state, the table is closed and released from the cache.

A table is opened for each concurrent access. This means the table needs to be opened twice if two threads access the same table or if a thread accesses the table twice in the same query (for example, by joining the table to itself). Each concurrent open requires an entry in the table cache. The first open of any table takes two file descriptors: one for the data file and one for the index file. Each additional use of the table takes only one file descriptor, for the data file. The index file descriptor is shared among all threads.

If you are opening a table with the `HANDLER tbl_name OPEN` statement, a dedicated table object is allocated for the thread. This table object is not shared by other threads and is not closed until the thread calls `HANDLER tbl_name CLOSE` or the thread terminates. When this happens, the table is put back in the table cache (if the cache isn't full). See Section 13.1.3, “`HANDLER` Syntax”.

You can determine whether your table cache is too small by checking the **mysqld** status variable `Opened_tables`:

```
mysql> SHOW STATUS LIKE 'Opened_tables';
```

```
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Opened_tables | 2741  |
+---------------+-------+
```

If the value is quite big, even when you haven't issued a lot of `FLUSH TABLES` statements, you should increase your table cache size. See Section 5.2.3, “Server System Variables” and Section 5.2.4, “Server Status Variables”.

7.4.9. Drawbacks to Creating Many Tables in the Same Database

If you have many `MyISAM` or `ISAM` tables in a database directory, open, close, and create operations are slow. If you execute `SELECT` statements on many different tables, there is a little overhead when the table cache is full, because for every table that has to be opened, another must be closed. You can reduce this overhead by making the table cache larger.

7.5. Optimizing the MySQL Server

7.5.1. System Factors and Startup Parameter Tuning

We start with system-level factors, because some of these decisions must be made very early to achieve large performance gains. In other cases, a quick look at this section may suffice. However, it is always nice to have a sense of how much can be gained by changing things at this level.

The default operating system to use is very important! To get the best use of multiple-CPU machines, you should use Solaris (because its threads implementation works really well) or Linux (because the 2.4 kernel has really good SMP support). Note that older Linux kernels have a 2GB filesize limit by default. If you have such a kernel and a desperate need for files larger than 2GB, you should get the Large File Support (LFS) patch for the ext2 filesystem. Other filesystems such as ReiserFS and XFS do not have this 2GB limitation.

Before using MySQL in production, we advise you to test it on your intended platform.

Other tips:

- If you have enough RAM, you could remove all swap devices. Some operating systems use a swap device in some contexts even if you have free memory.
- Use the `--skip-external-locking` MySQL option to avoid external locking. This option is on by default as of MySQL 4.0. Before that, it is on by default when compiling with MIT-pthreads, because `flock()` isn't fully supported by MIT-pthreads on all platforms. It's also on by default for Linux because Linux file locking is not yet safe.

  Note that the `--skip-external-locking` option does not affect MySQL's functionality as long as you run only one server. Just remember to take down the server (or lock and flush the relevant tables) before you run **myisamchk**. On some systems this option is mandatory, because the external locking does not work in any case.

  The only case when you can't use `--skip-external-locking` is if you run multiple MySQL *servers* (not clients) on the same data, or if you run **myisamchk** to check (not repair) a table without telling the server to flush and lock the tables first.

You can still use `LOCK TABLES` and `UNLOCK TABLES` even if you are using `--skip-external-locking`.

7.5.2. Tuning Server Parameters

You can determine the default buffer sizes used by the **mysqld** server with this command (prior to MySQL 4.1, omit `--verbose`):

```
shell> mysqld --verbose --help
```

This command produces a list of all **mysqld** options and configurable system variables. The output includes the default variable values and looks something like this:

```
back_log                  current value: 5
bdb_cache_size            current value: 1048540
binlog_cache_size         current value: 32768
connect_timeout           current value: 5
delayed_insert_limit      current value: 100
delayed_insert_timeout    current value: 300
delayed_queue_size        current value: 1000
flush_time                current value: 0
interactive_timeout       current value: 28800
join_buffer_size          current value: 131072
key_buffer_size           current value: 1048540
long_query_time           current value: 10
lower_case_table_names    current value: 0
max_allowed_packet        current value: 1048576
max_binlog_cache_size     current value: 4294967295
max_connect_errors        current value: 10
max_connections           current value: 100
max_delayed_threads       current value: 20
max_heap_table_size       current value: 16777216
max_join_size             current value: 4294967295
max_sort_length           current value: 1024
max_tmp_tables            current value: 32
max_write_lock_count      current value: 4294967295
myisam_sort_buffer_size   current value: 8388608
net_buffer_length         current value: 16384
net_read_timeout          current value: 30
net_retry_count           current value: 10
net_write_timeout         current value: 60
read_buffer_size          current value: 131072
read_rnd_buffer_size      current value: 262144
slow_launch_time          current value: 2
sort_buffer               current value: 2097116
table_cache               current value: 64
thread_concurrency        current value: 10
thread_stack              current value: 131072
tmp_table_size            current value: 1048576
wait_timeout              current value: 28800
```

If there is a **mysqld** server currently running, you can see what values it actually is using for the system variables by connecting to it and issuing this statement:

```
mysql> SHOW VARIABLES;
```

You can also see some statistical and status indicators for a running server by issuing this statement:

```
mysql> SHOW STATUS;
```

System variable and status information also can be obtained using **mysqladmin**:

```
shell> mysqladmin variables
shell> mysqladmin extended-status
```

You can find a full description for all system and status variables in Section 5.2.3, “Server System Variables” and Section 5.2.4, “Server Status Variables”.

MySQL uses algorithms that are very scalable, so you can usually run with very little memory. However, normally you get better performance by giving MySQL more memory.

When tuning a MySQL server, the two most important variables to configure are `key_buffer_size` and `table_cache`. You should first feel confident that you have these set appropriately before trying to change any other variables.

The following examples indicate some typical variable values for different runtime configurations. The examples use the **mysqld_safe** script and use `--var_name=value` syntax to set the variable `var_name` to the value `value`. This syntax is available as of MySQL 4.0. For older versions of MySQL, take the following differences into account:

- Use **safe_mysqld** rather than **mysqld_safe**.
- Set variables using `--set-variable=var_name=value` or `-O var_name=value` syntax.
- For variable names that end in `_size`, you may need to specify them without `_size`. For example, the old name for `sort_buffer_size` is `sort_buffer`. The old name for `read_buffer_size` is `record_buffer`. To see which variables your version of the server recognizes, use **mysqld --help**.

If you have at least 256MB of memory and many tables and want maximum performance with a moderate number of clients, you should use something like this:

```
shell> mysqld_safe --key_buffer_size=64M --table_cache=256 \
           --sort_buffer_size=4M --read_buffer_size=1M &
```

If you have only 128MB of memory and only a few tables, but you still do a lot of sorting, you can use something like this:

```
shell> mysqld_safe --key_buffer_size=16M --sort_buffer_size=1M
```

If there are very many simultaneous connections, swapping problems may occur unless **mysqld** has been configured to use very little memory for each connection. **mysqld** performs better if you have enough memory for all connections.

With little memory and lots of connections, use something like this:

```
shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=100K \
           --read_buffer_size=100K &
```

Or even this:

```
shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=16K \
```

```
--table_cache=32 --read_buffer_size=8K \
--net_buffer_length=1K &
```

If you are doing `GROUP BY` or `ORDER BY` operations on tables that are much larger than your available memory, you should increase the value of `read_rnd_buffer_size` to speed up the reading of rows after sorting operations.

When you have installed MySQL, the `support-files` directory contains some different `my.cnf` sample files: `my-huge.cnf`, `my-large.cnf`, `my-medium.cnf`, and `my-small.cnf`. You can use these as a basis for optimizing your system.

Note that if you specify an option on the command line for **mysqld** or **mysqld_safe**, it remains in effect only for that invocation of the server. To use the option every time the server runs, put it in an option file.

To see the effects of a parameter change, do something like this (prior to MySQL 4.1, omit `--verbose`):

```
shell> mysqld --key_buffer_size=32M --verbose --help
```

The variable values are listed near the end of the output. Make sure that the `--verbose` and `--help` options are last. Otherwise, the effect of any options listed after them on the command line are not reflected in the output.

For information on tuning the `InnoDB` storage engine, see Section 15.12, “`InnoDB` Performance Tuning Tips”.

7.5.3. Controlling Query Optimizer Performance

The task of the query optimizer is to find an optimal plan for executing an SQL query. Because the difference in performance between ``good'' and ``bad'' plans can be orders of magnitude (that is, seconds versus hours or even days), most query optimizers, including that of MySQL, perform more or less exhaustive search for an optimal plan among all possible query evaluation plans. For join queries, the number of possible plans investigated by the MySQL optimizer grows exponentially with the number of tables referenced in a query. For small numbers of tables (typically less than 7-10) this is not a problem. However, when bigger queries are submitted, the time spent in query optimization may easily become the major bottleneck in the server performance.

MySQL 5.0.1 introduces a new more flexible method for query optimization that allows the user to control how exhaustive the optimizer is in its search for an optimal query evaluation plan. The general idea is that the fewer plans that are investigated by the optimizer, the less time it spends in compiling a query. On the other hand, because the optimizer skips some plans, it may miss finding an optimal plan.

The behavior of the optimizer with respect to the number of plans it evaluates can be controlled via two system variables:

- The `optimizer_prune_level` variable tells the optimizer to skip certain plans based on estimates of the number of rows accessed for each table. Our experience shows that this kind of ``educated guess'' rarely misses optimal plans, while it may dramatically reduce query compilation times. That is why this option is on (`optimizer_prune_level`=1) by default. However, if you believe that the optimizer missed better query plans, then this option can be switched off (`optimizer_prune_level`=0) with the risk that query compilation may take much longer. Notice that even with the use of this heuristic, the optimizer still explores a roughly exponential number of plans.

- The `optimizer_search_depth` variable tells how far in the ``future'' of each incomplete plan the optimizer should look in order to evaluate whether it should be expanded further. Smaller values of `optimizer_search_depth` may result in orders of magnitude smaller query compilation times. For example, queries with 12-13 or more tables may easily require hours and even days to compile if `optimizer_search_depth` is close to the number of tables in the query. At the same time, if compiled with `optimizer_search_depth` equal to 3 or 4, the compiler may compile in less than a minute for the same query. If you are unsure of what a reasonable value is for `optimizer_search_depth`, this variable can be set to 0 to tell the optimizer to determine the value automatically.

7.5.4. How Compiling and Linking Affects the Speed of MySQL

Most of the following tests were performed on Linux with the MySQL benchmarks, but they should give some indication for other operating systems and workloads.

You get the fastest executables when you link with `-static`.

On Linux, you get the fastest code when compiling with `pgcc` and `-O3`. You need about 200MB memory to compile `sql_yacc.cc` with these options, because **gcc/pgcc** needs a lot of memory to make all functions inline. You should also set `CXX=gcc` when configuring MySQL to avoid inclusion of the `libstdc++` library, which is not needed. Note that with some versions of `pgcc`, the resulting code runs only on true Pentium processors, even if you use the compiler option indicating that you want the resulting code to work on all x586-type processors (such as AMD).

By just using a better compiler and better compiler options, you can get a 10-30% speed increase in your application. This is particularly important if you compile the MySQL server yourself.

We have tested both the Cygnus CodeFusion and Fujitsu compilers, but when we tested them, neither was sufficiently bug-free to allow MySQL to be compiled with optimizations enabled.

The standard MySQL binary distributions are compiled with support for all character sets. When you compile MySQL yourself, you should include support only for the character sets that you are going to use. This is controlled by the `--with-charset` option to **configure**.

Here is a list of some measurements that we have made:

- If you use `pgcc` and compile everything with `-O6`, the **mysqld** server is 1% faster than with **gcc** 2.95.2.
- If you link dynamically (without `-static`), the result is 13% slower on Linux. Note that you still can use a dynamically linked MySQL library for your client applications. It is the server that is most critical for performance.
- If you strip your **mysqld** binary with `strip mysqld`, the resulting binary can be up to 4% faster.
- For a connection from a client to a server running on the same host, if you connect using TCP/IP rather than a Unix socket file, performance is 7.5% slower. (On Unix, if you connect to the hostname `localhost`, MySQL uses a socket file by default.)
- For TCP/IP connections from a client to a server, connecting to a remote server on another host is 8-11% slower than connecting to the local server on the same host, even for connections over 100Mb/s Ethernet.
- When running our benchmark tests using secure connections (all data encrypted with internal SSL

support) performance was 55% slower than for unencrypted connections.

- If you compile with `--with-debug=full`, most queries are 20% slower. Some queries may take substantially longer; for example, the MySQL benchmarks ran 35% slower. If you use `--with-debug` (without `=full`), the slowdown is only 15%. For a version of **mysqld** that has been compiled with `--with-debug=full`, you can disable memory checking at runtime by starting it with the `--skip-safemalloc` option. The execution speed should then be close to that obtained when configuring with `--with-debug`.
- On a Sun UltraSPARC-IIe, a server compiled with Forte 5.0 is 4% faster than one compiled with **gcc** 3.2.
- On a Sun UltraSPARC-IIe, a server compiled with Forte 5.0 is 4% faster in 32-bit mode than in 64-bit mode.
- Compiling with **gcc** 2.95.2 for UltraSPARC with the `-mcpu=v8 -Wa,-xarch=v8plusa` options gives 4% more performance.
- On Solaris 2.5.1, MIT-pthreads is 8-12% slower than Solaris native threads on a single processor. With more load or CPUs, the difference should be larger.
- Compiling on Linux-x86 using **gcc** without frame pointers (`-fomit-frame-pointer` or `-fomit-frame-pointer -ffixed-ebp`) makes **mysqld** 1-4% faster.

Binary MySQL distributions for Linux that are provided by MySQL AB used to be compiled with `pgcc`. We had to go back to regular **gcc** due to a bug in `pgcc` that would generate code that does not run on AMD. We will continue using **gcc** until that bug is resolved. In the meantime, if you have a non-AMD machine, you can get a faster binary by compiling with `pgcc`. The standard MySQL Linux binary is linked statically to make it faster and more portable.

7.5.5. How MySQL Uses Memory

The following list indicates some of the ways that the **mysqld** server uses memory. Where applicable, the name of the system variable relevant to the memory use is given:

- The key buffer (variable `key_buffer_size`) is shared by all threads; other buffers used by the server are allocated as needed. See Section 7.5.2, “Tuning Server Parameters”.
- Each connection uses some thread-specific space:
  - A stack (default 64KB, variable `thread_stack`)
  - A connection buffer (variable `net_buffer_length`)
  - A result buffer (variable `net_buffer_length`)

  The connection buffer and result buffer are dynamically enlarged up to `max_allowed_packet` when needed. While a query is running, a copy of the current query string is also allocated.
- All threads share the same base memory.
- Only compressed `ISAM` and `MyISAM` tables are memory mapped. This is because the 32-bit memory space of 4GB is not large enough for most big tables. When systems with a 64-bit address space become more common, we may add general support for memory mapping.
- Each request that performs a sequential scan of a table allocates a read buffer (variable

read_buffer_size).

- When reading rows in ``random'' order (for example, after a sort), a random-read buffer may be allocated to avoid disk seeks. (variable read_rnd_buffer_size).

- All joins are done in one pass, and most joins can be done without even using a temporary table. Most temporary tables are memory-based (HEAP) tables. Temporary tables with a large record length (calculated as the sum of all column lengths) or that contain BLOB columns are stored on disk.

  One problem before MySQL 3.23.2 is that if an internal in-memory heap table exceeds the size of tmp_table_size, the error The table *tbl_name* is full occurs. From 3.23.2 on, this is handled automatically by changing the in-memory heap table to a disk-based MyISAM table as necessary. To work around this problem for older servers, you can increase the temporary table size by setting the tmp_table_size option to **mysqld**, or by setting the SQL option SQL_BIG_TABLES in the client program. See Section 13.5.3, “SET Syntax”.

  In MySQL 3.20, the maximum size of the temporary table is record_buffer*16; if you are using this version, you have to increase the value of record_buffer. You can also start **mysqld** with the --big-tables option to always store temporary tables on disk. However, this affects the speed of many complicated queries.

- Most requests that perform a sort allocate a sort buffer and zero to two temporary files depending on the result set size. See Section A.4.4, “Where MySQL Stores Temporary Files”.

- Almost all parsing and calculating is done in a local memory store. No memory overhead is needed for small items, so the normal slow memory allocation and freeing is avoided. Memory is allocated only for unexpectedly large strings; this is done with malloc() and free().

- For each MyISAM and ISAM table that is opened, the index file is opened once and the data file is opened once for each concurrently running thread. For each concurrent thread, a table structure, column structures for each column, and a buffer of size 3 * *N* are allocated (where *N* is the maximum row length, not counting BLOB columns). A BLOB column requires five to eight bytes plus the length of the BLOB data. The MyISAM and ISAM storage engines maintain one extra row buffer for internal use.

- For each table having BLOB columns, a buffer is enlarged dynamically to read in larger BLOB values. If you scan a table, a buffer as large as the largest BLOB value is allocated.

- Handler structures for all in-use tables are saved in a cache and managed as a FIFO. By default, the cache has 64 entries. If a table has been used by two running threads at the same time, the cache contains two entries for the table. See Section 7.4.8, “How MySQL Opens and Closes Tables”.

- A FLUSH TABLES statement or **mysqladmin flush-tables** command closes all tables that are not in use at once and marks all in-use tables to be closed when the currently executing thread finishes. This effectively frees most in-use memory. FLUSH TABLES does not return until all tables have been closed.

ps and other system status programs may report that **mysqld** uses a lot of memory. This may be caused by thread stacks on different memory addresses. For example, the Solaris version of **ps** counts the unused memory between stacks as used memory. You can verify this by checking available swap with swap -s. We have tested **mysqld** with several memory-leakage detectors (both commercial and open source), so there should be no memory leaks.

7.5.6. How MySQL Uses DNS

When a new client connects to **mysqld**, **mysqld** spawns a new thread to handle the request. This thread

first checks whether the hostname is in the hostname cache. If not, the thread attempts to resolve the hostname:

- If the operating system supports the thread-safe `gethostbyaddr_r()` and `gethostbyname_r()` calls, the thread uses them to perform hostname resolution.
- If the operating system doesn't support the thread-safe calls, the thread locks a mutex and calls `gethostbyaddr()` and `gethostbyname()` instead. In this case, no other thread can resolve hostnames that are not in the hostname cache until the first thread unlocks the mutex.

You can disable DNS hostname lookups by starting **mysqld** with the `--skip-name-resolve` option. However, in this case, you can use only IP numbers in the MySQL grant tables.

If you have a very slow DNS and many hosts, you can get more performance by either disabling DNS lookups with `--skip-name-resolve` or by increasing the `HOST_CACHE_SIZE` define (default value: 128) and recompiling **mysqld**.

You can disable the hostname cache by starting the server with the `--skip-host-cache` option. To clear the hostname cache, issue a `FLUSH HOSTS` statement or execute the **mysqladmin flush-hosts** command.

If you want to disallow TCP/IP connections entirely, start **mysqld** with the `--skip-networking` option.

7.6. Disk Issues

- Disk seeks are a big performance bottleneck. This problem becomes more apparent when the amount of data starts to grow so large that effective caching becomes impossible. For large databases where you access data more or less randomly, you can be sure that you need at least one disk seek to read and a couple of disk seeks to write things. To minimize this problem, use disks with low seek times.
- Increase the number of available disk spindles (and thereby reduce the seek overhead) by either symlinking files to different disks or striping the disks:
  - Using symbolic links

    This means that, for `MyISAM` tables, you symlink the index file and/or data file from their usual location in the data directory to another disk (that may also be striped). This makes both the seek and read times better, assuming that the disk is not used for other purposes as well. See Section 7.6.1, “Using Symbolic Links”.

  - Striping

    Striping means that you have many disks and put the first block on the first disk, the second block on the second disk, and the Nth block on the (N mod number_of_disks) disk, and so on. This means if your normal data size is less than the stripe size (or perfectly aligned), you get much better performance. Striping is very dependent on the operating system and the stripe size, so benchmark your application with different stripe sizes. See Section 7.1.5, “Using Your Own Benchmarks”.

    The speed difference for striping is *very* dependent on the parameters. Depending on how you set the striping parameters and number of disks, you may get differences measured in orders of magnitude. You have to choose to optimize for random or sequential access.
- For reliability you may want to use RAID 0+1 (striping plus mirroring), but in this case, you need

2*N drives to hold N drives of data. This is probably the best option if you have the money for it. However, you may also have to invest in some volume-management software to handle it efficiently.

- A good option is to vary the RAID level according to how critical a type of data is. For example, store semi-important data that can be regenerated on a RAID 0 disk, but store really important data such as host information and logs on a RAID 0+1 or RAID N disk. RAID N can be a problem if you have many writes, due to the time required to update the parity bits.

- On Linux, you can get much more performance by using `hdparm` to configure your disk's interface. (Up to 100% under load is not uncommon.) The following `hdparm` options should be quite good for MySQL, and probably for many other applications:

  ```
  hdparm -m 16 -d 1
  ```

  Note that performance and reliability when using this command depends on your hardware, so we strongly suggest that you test your system thoroughly after using `hdparm`. Please consult the `hdparm` manual page for more information. If `hdparm` is not used wisely, filesystem corruption may result, so back up everything before experimenting!

- You can also set the parameters for the filesystem that the database uses:

  If you don't need to know when files were last accessed (which is not really useful on a database server), you can mount your filesystems with the `-o noatime` option. That skips updates to the last access time in inodes on the filesystem, which avoids some disk seeks.

  On many operating systems, you can set a filesystem to be updated asynchronously by mounting it with the `-o async` option. If your computer is reasonably stable, this should give you more performance without sacrificing too much reliability. (This flag is on by default on Linux.)

7.6.1. Using Symbolic Links

You can move tables and databases from the database directory to other locations and replace them with symbolic links to the new locations. You might want to do this, for example, to move a database to a file system with more free space or increase the speed of your system by spreading your tables to different disk.

The recommended way to do this is to just symlink databases to a different disk. Symlink tables only as a last resort.

7.6.1.1. Using Symbolic Links for Databases on Unix

On Unix, the way to symlink a database is to first create a directory on some disk where you have free space and then create a symlink to it from the MySQL data directory.

```
shell> mkdir /dr1/databases/test
shell> ln -s /dr1/databases/test /path/to/datadir
```

MySQL doesn't support linking one directory to multiple databases. Replacing a database directory with a symbolic link works fine as long as you don't make a symbolic link between databases. Suppose that you have a database `db1` under the MySQL data directory, and then make a symlink `db2` that points to `db1`:

```
shell> cd /path/to/datadir
shell> ln -s db1 db2
```

For any table `tbl_a` in `db1`, there also appears to be a table `tbl_a` in `db2`. If one client updates `db1.tbl_a` and another client updates `db2.tbl_a`, there are problems.

If you really need to do this, you can change one of the source files. The file to modify depends on your version of MySQL. For MySQL 4.0 and up, look for the following statement in the `mysys/my_symlink.c` file:

```
if (!(MyFlags & MY_RESOLVE_LINK) ||
    (!lstat(filename,&stat_buff) && S_ISLNK(stat_buff.st_mode)))
```

Before MySQL 4.0, look for this statement in the `mysys/mf_format.c` file:

```
if (flag & 32 || (!lstat(to,&stat_buff) && S_ISLNK(stat_buff.st_mode)))
```

Change the statement to this:

```
if (1)
```

On Windows, you can use internal symbolic links to directories by compiling MySQL with `-DUSE_SYMDIR`. This allows you to put different databases on different disks. See Section 7.6.1.3, “Using Symbolic Links for Databases on Windows”. (It is necessary to define `USE_SYMDIR` explicitly only before MySQL 4.0. As of MySQL 4.0, symbolic link support is enabled by default for all Windows servers.)

7.6.1.2. Using Symbolic Links for Tables on Unix

Before MySQL 4.0, you should not symlink tables unless you are *very* careful with them. The problem is that if you run `ALTER TABLE`, `REPAIR TABLE`, or `OPTIMIZE TABLE` on a symlinked table, the symlinks are removed and replaced by the original files. This happens because these statements work by creating a temporary file in the database directory and replacing the original file with the temporary file when the statement operation is complete.

You should not symlink tables on systems that don't have a fully working `realpath()` call. (At least Linux and Solaris support `realpath()`). You can check whether your system supports symbolic links by issuing a `SHOW VARIABLES LIKE 'have_symlink'` statement.

In MySQL 4.0, symlinks are fully supported only for `MyISAM` tables. For other table types, you may get strange problems if you try to use symbolic links on files in the operating system with any of the preceding statements.

The handling of symbolic links for `MyISAM` tables in MySQL 4.0 works the following way:

- In the data directory, you always have the table definition file, the data file, and the index file. The data file and index file can be moved elsewhere and replaced in the data directory by symlinks. The definition file cannot.
- You can symlink the data file and the index file independently to different directories.
- The symlinking can be done manually from the command line with `ln -s` if **mysqld** is not running. With SQL, you can instruct the server to perform the symlinking by using the `DATA DIRECTORY` and `INDEX DIRECTORY` options to `CREATE TABLE`. See Section 13.2.6, “`CREATE TABLE` Syntax”.
- **myisamchk** does not replace a symlink with the data file or index file. It works directly on the file a symlink points to. Any temporary files are created in the directory where the data file or index file is

located.

- When you drop a table that is using symlinks, both the symlink and the file the symlink points to are dropped. This is a good reason why you should *not* run **mysqld** as `root` or allow users to have write access to the MySQL database directories.
- If you rename a table with `ALTER TABLE ... RENAME` and you don't move the table to another database, the symlinks in the database directory are renamed to the new names and the data file and index file are renamed accordingly.
- If you use `ALTER TABLE ... RENAME` to move a table to another database, the table is moved to the other database directory. The old symlinks and the files to which they pointed are deleted. In other words, the new table is not symlinked.
- If you are not using symlinks, you should use the `--skip-symbolic-links` option to **mysqld** to ensure that no one can use **mysqld** to drop or rename a file outside of the data directory.

`SHOW CREATE TABLE` doesn't report if a table has symbolic links prior to MySQL 4.0.15. This is also true for **mysqldump**, which uses `SHOW CREATE TABLE` to generate `CREATE TABLE` statements.

Table symlink operations that are not yet supported:

- `ALTER TABLE` ignores the `DATA DIRECTORY` and `INDEX DIRECTORY` table options.
- `BACKUP TABLE` and `RESTORE TABLE` don't respect symbolic links.
- The `.frm` file must *never* be a symbolic link (as indicated previously, only the data and index files can be symbolic links). Attempting to do this (for example, to make synonyms) produces incorrect results. Suppose that you have a database `db1` under the MySQL data directory, a table `tbl1` in this database, and in the `db1` directory you make a symlink `tbl2` that points to `tbl1`:

  ```
  shell> cd /path/to/datadir/db1
  shell> ln -s tbl1.frm tbl2.frm
  shell> ln -s tbl1.MYD tbl2.MYD
  shell> ln -s tbl1.MYI tbl2.MYI
  ```

  There are problems if one thread reads `db1.tbl1` and another thread updates `db1.tbl2`:

  - The query cache is fooled (it believes `tbl1` has not been updated so it returns outdated results).
  - `ALTER` statements on `tbl2` also fail.

7.6.1.3. Using Symbolic Links for Databases on Windows

Beginning with MySQL 3.23.16, the **mysqld-max** and `mysql-max-nt` servers for Windows are compiled with the `-DUSE_SYMDIR` option. This allows you to put a database directory on a different disk by setting up a symbolic link to it. This is similar to the way that symbolic links work on Unix, although the procedure for setting up the link is different.

As of MySQL 4.0, symbolic links are enabled by default. If you don't need them, you can disable them with the `skip-symbolic-links` option:

```
[mysqld]
skip-symbolic-links
```

Before MySQL 4.0, symbolic links are disabled by default. To enable them, you should put the following entry in your `my.cnf` or `my.ini` file:

```
[mysqld]
symbolic-links
```

On Windows, you make a symbolic link to a MySQL database by creating a file in the data directory that contains the path to the destination directory. The file should be named `db_name.sym`, where *`db_name`* is the database name.

Suppose that the MySQL data directory is `C:\mysql\data` and you want to have database `foo` located at `D:\data\foo`. Set up a symlink like this:

1. Make sure that the `D:\data\foo` directory exists by creating it if necessary. If you have a database directory named `foo` in the data directory, you should move it to `D:\data`. Otherwise, the symbolic link is ineffective. To avoid problems, the server should not be running when you move the database directory.

2. Create a file `C:\mysql\data\foo.sym` that contains the pathname `D:\data\foo\`.

After that, all tables created in the database `foo` are created in `D:\data\foo`. Note that the symbolic link is not used if a directory with the database name exists in the MySQL data directory.

Chapter 8. MySQL Client and Utility Programs

There are many different MySQL client programs that connect to the server to access databases or perform administrative tasks. Other utilities are available as well. These do not communicate with the server but perform MySQL-related operations.

This chapter provides a brief overview of these programs and then a more detailed description of each one. The descriptions indicate how to invoke the programs and the options they understand. See Chapter 4, *Using MySQL Programs* for general information on invoking programs and specifying program options.

8.1. Overview of the Client-Side Scripts and Utilities

The following list briefly describes the MySQL client programs and utilities:

- `myisampack`

  A utility that compresses `MyISAM` tables to produce smaller read-only tables. See Section 8.2, "**myisampack**, the MySQL Compressed Read-only Table Generator".

- `mysql`

  The command-line tool for interactively entering SQL statements or executing them from a file in batch mode. See Section 8.3, "**mysql**, the Command-Line Tool".

- `mysqlaccess`

  A script that checks the access privileges for a host, user, and database combination.

- `mysqladmin`

  A client that performs administrative operations, such as creating or dropping databases, reloading the grant tables, flushing tables to disk, and reopening log files. **mysqladmin** can also be used to retrieve version, process, and status information from the server. See Section 8.4, "**mysqladmin**, Administering a MySQL Server".

- `mysqlbinlog`

  A utility for reading statements from a binary log. The log of executed statements contained in the binary log files can be used to help recover from a crash. See Section 8.5, "The **mysqlbinlog** Binary Log Utility".

- `mysqlcc`

  A client that provides a graphical interface for interacting with the server. See Section 8.6, "**mysqlcc**, the MySQL Control Center".

- `mysqlcheck`

  A table-maintenance client that checks, repairs, analyzes, and optimizes tables. See Section 8.7, “The **mysqlcheck** Table Maintenance and Repair Program”.

- `mysqldump`

  A client that dumps a MySQL database into a file as SQL statements or as tab-separated text files. Enhanced freeware originally by Igor Romanenko. See Section 8.8, “The **mysqldump** Database Backup Program”.

- `mysqlhotcopy`

  A utility that quickly makes backups of `MyISAM` or `ISAM` tables while the server is running. See Section 8.9, “The **mysqlhotcopy** Database Backup Program”.

- `mysqlimport`

  A client that imports text files into their respective tables using `LOAD DATA INFILE`. See Section 8.10, “The **mysqlimport** Data Import Program”.

- `mysqlshow`

  A client that displays information about databases, tables, columns, and indexes. See Section 8.11, “**mysqlshow**, Showing Databases, Tables, and Columns”.

- `perror`

  A utility that displays the meaning of system or MySQL error codes. See Section 8.12, “**perror**, Explaining Error Codes”.

- `replace`

  A utility program that changes strings in place in files or on the standard input. See Section 8.13, “The **replace** String-Replacement Utility”.

Each MySQL program takes many different options. However, every MySQL program provides a `--help` option that you can use to get a full description of the program's different options. For example, try **mysql --help**.

MySQL clients that communicate with the server using the `mysqlclient` library use the following environment variables:

`MYSQL_UNIX_PORT`	The default Unix socket file; used for connections to `localhost`
`MYSQL_TCP_PORT`	The default port number; used for TCP/IP connections
`MYSQL_PWD`	The default password
`MYSQL_DEBUG`	Debug trace options when debugging
`TMPDIR`	The directory where temporary tables and files are created

Use of `MYSQL_PWD` is insecure. See Section 5.6.6, “Keeping Your Password Secure”.

You can override the default option values or values specified in environment variables for all standard programs by specifying options in an option file or on the command line. Section 4.3, “Specifying Program Options”.

8.2. myisampack, the MySQL Compressed Read-only Table Generator

The **myisampack** utility compresses `MyISAM` tables. **myisampack** works by compressing each column in the table separately. Usually, **myisampack** packs the data file 40%-70%.

When the table is used later, the information needed to decompress columns is read into memory. This results in much better performance when accessing individual records, because you only have to uncompress exactly one record, not a much larger disk block as when using Stacker on MS-DOS.

MySQL uses `mmap()` when possible to perform memory mapping on compressed tables. If `mmap()` doesn't work, MySQL falls back to normal read/write file operations.

A similar utility, **pack_isam**, compresses `ISAM` tables. Because `ISAM` tables are deprecated, this section discusses only **myisampack**, but the general procedures for using **myisampack** are also true for **pack_isam** unless otherwise specified.

Please note the following:

- If the **mysqld** server was invoked with the `--skip-external-locking` option, it is not a good idea to invoke **myisampack** if the table might be updated by the server during the packing process.
- After packing a table, it becomes read-only. This is generally intended (such as when accessing packed tables on a CD). Allowing writes to a packed table is on our TODO list, but with low priority.
- **myisampack** can pack `BLOB` or `TEXT` columns. The older **pack_isam** program for `ISAM` tables cannot.

Invoke **myisampack** like this:

```
shell> myisampack [options] filename ...
```

Each filename should be the name of an index (`.MYI`) file. If you are not in the database directory, you should specify the pathname to the file. It is permissible to omit the `.MYI` extension.

myisampack supports the following options:

- `--help, -?`

  Display a help message and exit.

- `--backup, -b`

  Make a backup of the table data file using the name *`tbl_name`*`.OLD`.

- `--debug[=debug_options], -# [debug_options]`

Write a debugging log. The *debug_options* string often is `'d:t:o,file_name'`.

- `--force, -f`

  Produce a packed table even if it becomes larger than the original or if the temporary file from an earlier invocation of **myisampack** exists. (**myisampack** creates a temporary file named *tbl_name*`.TMD` while it compresses the table. If you kill **myisampack**, the `.TMD` file might not be deleted.) Normally, **myisampack** exits with an error if it finds that *tbl_name*`.TMD` exists. With `--force`, **myisampack** packs the table anyway.

- `--join=big_tbl_name, -j big_tbl_name`

  Join all tables named on the command line into a single table *big_tbl_name*. All tables that are to be combined *must* have identical structure (same column names and types, same indexes, and so forth).

- `--packlength=#, -p #`

  Specify the record length storage size, in bytes. The value should be 1, 2, or 3. **myisampack** stores all rows with length pointers of 1, 2, or 3 bytes. In most normal cases, **myisampack** can determine the right length value before it begins packing the file, but it may notice during the packing process that it could have used a shorter length. In this case, **myisampack** prints a note that the next time you pack the same file, you could use a shorter record length.

- `--silent, -s`

  Silent mode. Write output only when errors occur.

- `--test, -t`

  Don't actually pack the table, just test packing it.

- `--tmp_dir=path, -T path`

  Use the named directory as the location in which to write the temporary table.

- `--verbose, -v`

  Verbose mode. Write information about the progress of the packing operation and its result.

- `--version, -V`

  Display version information and exit.

- `--wait, -w`

  Wait and retry if the table is in use. If the **mysqld** server was invoked with the `--skip-external-locking` option, it is not a good idea to invoke **myisampack** if the table might be updated by the server during the packing process.

The following sequence of commands illustrates a typical table compression session:

```
shell> ls -l station.*
-rw-rw-r--   1 monty    my         994128 Apr 17 19:00 station.MYD
-rw-rw-r--   1 monty    my          53248 Apr 17 19:00 station.MYI
-rw-rw-r--   1 monty    my           5767 Apr 17 19:00 station.frm

shell> myisamchk -dvv station
```

```
MyISAM file:     station
Isam-version:  2
Creation time: 1996-03-13 10:08:58
Recover time:  1997-02-02  3:06:43
Data records:              1192  Deleted blocks:              0
Datafile parts:            1192  Deleted data:                0
Datafile pointer (bytes):     2  Keyfile pointer (bytes):     2
Max datafile length:   54657023  Max keyfile length:   33554431
Recordlength:               834
Record format: Fixed length

table description:
Key Start Len Index   Type                 Root  Blocksize    Rec/key
1   2     4   unique  unsigned long        1024       1024          1
2   32    30  multip. text                10240       1024          1

Field Start Length Type
1     1     1
2     2     4
3     6     4
4     10    1
5     11    20
6     31    1
7     32    30
8     62    35
9     97    35
10    132   35
11    167   4
12    171   16
13    187   35
14    222   4
15    226   16
16    242   20
17    262   20
18    282   20
19    302   30
20    332   4
21    336   4
22    340   1
23    341   8
24    349   8
25    357   8
26    365   2
27    367   2
28    369   4
29    373   4
30    377   1
31    378   2
32    380   8
33    388   4
34    392   4
35    396   4
36    400   4
37    404   1
38    405   4
39    409   4
40    413   4
41    417   4
42    421   4
43    425   4
44    429   20
45    449   30
46    479   1
```

```
47    480   1
48    481   79
49    560   79
50    639   79
51    718   79
52    797   8
53    805   1
54    806   1
55    807   20
56    827   4
57    831   4

shell> myisampack station.MYI
Compressing station.MYI: (1192 records)
- Calculating statistics

normal:     20  empty-space:   16  empty-zero:     12  empty-fill:  11
pre-space:   0  end-space:     12  table-lookups:   5  zero:         7
Original trees:  57  After join: 17
- Compressing file
87.14%
Remember to run myisamchk -rq on compressed tables

shell> ls -l station.*
-rw-rw-r--   1 monty    my         127874 Apr 17 19:00 station.MYD
-rw-rw-r--   1 monty    my          55296 Apr 17 19:04 station.MYI
-rw-rw-r--   1 monty    my           5767 Apr 17 19:00 station.frm

shell> myisamchk -dvv station

MyISAM file:     station
Isam-version:  2
Creation time: 1996-03-13 10:08:58
Recover time:  1997-04-17 19:04:26
Data records:               1192  Deleted blocks:              0
Datafile parts:             1192  Deleted data:                0
Datafile pointer (bytes):      3  Keyfile pointer (bytes):     1
Max datafile length:    16777215  Max keyfile length:     131071
Recordlength:                834
Record format: Compressed

table description:
Key Start Len Index   Type                 Root  Blocksize    Rec/key
1   2     4   unique  unsigned long       10240       1024          1
2   32    30  multip. text                54272       1024          1

Field Start Length Type                        Huff tree  Bits
1     1     1      constant                           1     0
2     2     4      zerofill(1)                        2     9
3     6     4      no zeros, zerofill(1)              2     9
4     10    1                                         3     9
5     11    20     table-lookup                       4     0
6     31    1                                         3     9
7     32    30     no endspace, not_always            5     9
8     62    35     no endspace, not_always, no empty  6     9
9     97    35     no empty                           7     9
10    132   35     no endspace, not_always, no empty  6     9
11    167   4      zerofill(1)                        2     9
12    171   16     no endspace, not_always, no empty  5     9
13    187   35     no endspace, not_always, no empty  6     9
14    222   4      zerofill(1)                        2     9
15    226   16     no endspace, not_always, no empty  5     9
16    242   20     no endspace, not_always            8     9
17    262   20     no endspace, no empty              8     9
```

```
18    282   20     no endspace, no empty                5     9
19    302   30     no endspace, no empty                6     9
20    332   4      always zero                          2     9
21    336   4      always zero                          2     9
22    340   1                                           3     9
23    341   8      table-lookup                         9     0
24    349   8      table-lookup                        10     0
25    357   8      always zero                          2     9
26    365   2                                           2     9
27    367   2      no zeros, zerofill(1)                2     9
28    369   4      no zeros, zerofill(1)                2     9
29    373   4      table-lookup                        11     0
30    377   1                                           3     9
31    378   2      no zeros, zerofill(1)                2     9
32    380   8      no zeros                             2     9
33    388   4      always zero                          2     9
34    392   4      table-lookup                        12     0
35    396   4      no zeros, zerofill(1)               13     9
36    400   4      no zeros, zerofill(1)                2     9
37    404   1                                           2     9
38    405   4      no zeros                             2     9
39    409   4      always zero                          2     9
40    413   4      no zeros                             2     9
41    417   4      always zero                          2     9
42    421   4      no zeros                             2     9
43    425   4      always zero                          2     9
44    429   20     no empty                             3     9
45    449   30     no empty                             3     9
46    479   1                                          14     4
47    480   1                                          14     4
48    481   79     no endspace, no empty               15     9
49    560   79     no empty                             2     9
50    639   79     no empty                             2     9
51    718   79     no endspace                         16     9
52    797   8      no empty                             2     9
53    805   1                                          17     1
54    806   1                                           3     9
55    807   20     no empty                             3     9
56    827   4      no zeros, zerofill(2)                2     9
57    831   4      no zeros, zerofill(1)                2     9
```

myisampack displays the following kinds of information:

- `normal`

  The number of columns for which no extra packing is used.

- `empty-space`

  The number of columns containing values that are only spaces; these occupy one bit.

- `empty-zero`

  The number of columns containing values that are only binary zeros; these occupy one bit.

- `empty-fill`

  The number of integer columns that don't occupy the full byte range of their type; these are changed to a smaller type. For example, a `BIGINT` column (eight bytes) can be stored as a `TINYINT` column (one byte) if all its values are in the range from `-128` to `127`.

- `pre-space`

  The number of decimal columns that are stored with leading spaces. In this case, each value contains a count for the number of leading spaces.

- `end-space`

  The number of columns that have a lot of trailing spaces. In this case, each value contains a count for the number of trailing spaces.

- `table-lookup`

  The column had only a small number of different values, which were converted to an `ENUM` before Huffman compression.

- `zero`

  The number of columns for which all values are zero.

- `Original trees`

  The initial number of Huffman trees.

- `After join`

  The number of distinct Huffman trees left after joining trees to save some header space.

After a table has been compressed, **myisamchk -dvv** prints additional information about each column:

- `Type`

  The column type. The value may contain any of the following descriptors:

  - `constant`

    All rows have the same value.

  - `no endspace`

    Don't store endspace.

  - `no endspace, not_always`

    Don't store endspace and don't do endspace compression for all values.

  - `no endspace, no empty`

    Don't store endspace. Don't store empty values.

  - `table-lookup`

    The column was converted to an `ENUM`.

  - `zerofill(n)`

    The most significant *n* bytes in the value are always 0 and are not stored.

  - `no zeros`

Don't store zeros.

- `always zero`

  Zero values are stored using one bit.

- `Huff tree`

  The number of the Huffman tree associated with the column.

- `Bits`

  The number of bits used in the Huffman tree.

After you run **myisampack**, you must run **myisamchk** to re-create any indexes. At this time, you can also sort the index blocks and create statistics needed for the MySQL optimizer to work more efficiently:

```
shell> myisamchk -rq --sort-index --analyze tbl_name.MYI
```

A similar procedure applies for `ISAM` tables. After using **pack_isam**, use **isamchk** to re-create the indexes:

```
shell> isamchk -rq --sort-index --analyze tbl_name.ISM
```

After you have installed the packed table into the MySQL database directory, you should execute **mysqladmin flush-tables** to force **mysqld** to start using the new table.

To unpack a packed table, use the `--unpack` option to **myisamchk** or **isamchk**.

8.3. mysql, the Command-Line Tool

mysql is a simple SQL shell (with GNU `readline` capabilities). It supports interactive and non-interactive use. When used interactively, query results are presented in an ASCII-table format. When used non-interactively (for example, as a filter), the result is presented in tab-separated format. The output format can be changed using command-line options.

If you have problems due to insufficient memory for large result sets, use the `--quick` option. This forces **mysql** to retrieve results from the server a row at a time rather than retrieving the entire result set and buffering it in memory before displaying it. This is done by using `mysql_use_result()` rather than `mysql_store_result()` to retrieve the result set.

Using **mysql** is very easy. Invoke it from the prompt of your command interpreter as follows:

```
shell> mysql db_name
```

Or:

```
shell> mysql --user=user_name --password=your_password db_name
```

Then type an SQL statement, end it with `';'`, `\g`, or `\G` and press Enter.

You can run a script simply like this:

```
shell> mysql db_name < script.sql > output.tab
```

mysql supports the following options:

- `--help, -?`

  Display a help message and exit.

- `--batch, -B`

  Print results using tab as the column separator, with each row on a new line. With this option, **mysql** doesn't use the history file.

- `--character-sets-dir=path`

  The directory where character sets are installed. See Section 5.8.1, “The Character Set Used for Data and Sorting”.

- `--compress, -C`

  Compress all information sent between the client and the server if both support compression.

- `--database=db_name, -D db_name`

  The database to use. This is useful mainly in an option file.

- `--debug[=debug_options], -# [debug_options]`

  Write a debugging log. The *debug_options* string often is `'d:t:o,file_name'`. The default is `'d:t:o,/tmp/mysql.trace'`.

- `--debug-info, -T`

  Print some debugging information when the program exits.

- `--default-character-set=charset`

  Use *charset* as the default character set. See Section 5.8.1, “The Character Set Used for Data and Sorting”.

- `--execute=statement, -e statement`

  Execute the statement and quit. The default output format is like that produced with `--batch`.

- `--force, -f`

  Continue even if an SQL error occurs.

- `--host=host_name, -h host_name`

  Connect to the MySQL server on the given host.

- `--html, -H`

  Produce HTML output.

- `--ignore-space, -i`

  Ignore spaces after function names. The effect of this is described in the discussion for `IGNORE_SPACE` in Section 5.2.2, “The Server SQL Mode”.

- `--local-infile[={0|1}]`

  Enable or disable `LOCAL` capability for `LOAD DATA INFILE`. With no value, the option enables `LOCAL`. It may be given as `--local-infile=0` or `--local-infile=1` to explicitly disable or enable `LOCAL`. Enabling `LOCAL` has no effect if the server does not also support it.

- `--named-commands, -G`

  Named commands are *enabled*. Long format commands are allowed as well as shortened `\*` commands. For example, `quit` and `\q` both are recognized.

- `--no-auto-rehash, -A`

  No automatic rehashing. This option causes **mysql** to start faster, but you must issue the `rehash` command if you want to use table and column name completion.

- `--no-beep, -b`

  Do not beep when errors occur.

- `--no-named-commands, -g`

  Named commands are disabled. Use the `\*` form only, or use named commands only at the beginning of a line ending with a semicolon (`;`). As of MySQL 3.23.22, **mysql** starts with this option *enabled* by default! However, even with this option, long-format commands still work from the first line.

- `--no-pager`

  Do not use a pager for displaying query output. Output paging is discussed further in Section 8.3.1, “**mysql** Commands”.

- `--no-tee`

  Do not copy output to a file. Tee files are discussed further in Section 8.3.1, “**mysql** Commands”.

- `--one-database, -O`

  Ignore statements except those for the default database named on the command line. This is useful for skipping updates to other databases in the binary log.

- `--pager[=command]`

  Use the given command for paging query output. If the command is omitted, the default pager is the value of your `PAGER` environment variable. Valid pagers are **less**, **more**, **cat [> filename]**, and so forth. This option works only on Unix. It does not work in batch mode. Output paging is discussed further in Section 8.3.1, "**mysql** Commands".

- `--password[=password], -p[password]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *`password`* value following the `--password` or `-p` option on the command line, you are prompted for one.

- `--port=port_num, -P port_num`

  The TCP/IP port number to use for the connection.

- `--prompt=format_str`

  Set the prompt to the specified format. The default is `mysql>`. The special sequences that the prompt can contain are described in Section 8.3.1, "**mysql** Commands".

- `--protocol={TCP | SOCKET | PIPE | MEMORY}`

  The connection protocol to use. New in MySQL 4.1.

- `--quick, -q`

  Don't cache each query result, print each row as it is received. This may slow down the server if the output is suspended. With this option, **mysql** doesn't use the history file.

- `--raw, -r`

  Write column values without escape conversion. Often used with the `--batch` option.

- `--reconnect`

  If the connection to the server is lost, automatically try to reconnect. A single reconnect attempt is made each time the connection is lost. To suppress reconnection behavior, use `--skip-reconnect`. New in MySQL 4.1.0.

- `--safe-updates, --i-am-a-dummy, -U`

  Allow only `UPDATE` and `DELETE` statements that specify rows to affect using key values. If you

have this option in an option file, you can override it by using `--safe-updates` on the command line. See Section 8.3.3, "**mysql** Tips" for more information about this option.

- `--sigint-ignore`

  Ignore `SIGINT` signals (typically the result of typing Control-C). This option was added in MySQL 4.1.6.

- `--silent, -s`

  Silent mode. Produce less output. This option can be given multiple times to produce less and less output.

- `--skip-column-names, -N`

  Don't write column names in results.

- `--skip-line-numbers, -L`

  Don't write line numbers for errors. Useful when you want to compare result files that include error messages.

- `--socket=`*`path`*`, -S `*`path`*

  The socket file to use for the connection.

- `--table, -t`

  Display output in table format. This is the default for interactive use, but can be used to produce table output in batch mode.

- `--tee=`*`file_name`*

  Append a copy of output to the given file. This option does not work in batch mode. Tee files are discussed further in Section 8.3.1, "**mysql** Commands".

- `--unbuffered, -n`

  Flush the buffer after each query.

- `--user=`*`user_name`*`, -u `*`user_name`*

  The MySQL username to use when connecting to the server.

- `--verbose, -v`

  Verbose mode. Produce more output. This option can be given multiple times to produce more and more output. (For example, `-v -v -v` produces the table output format even in batch mode.)

- `--version, -V`

  Display version information and exit.

- `--vertical, -E`

  Print the rows of query output vertically. Without this option, you can specify vertical output for individual statements by terminating them with `\G`.

- `--wait, -w`

  If the connection cannot be established, wait and retry instead of aborting.

- `--xml, -X`

  Produce XML output.

You can also set the following variables by using `--var_name=value` options:

- `connect_timeout`

  The number of seconds before connection timeout. (Default value is 0.)

- `max_allowed_packet`

  The maximum packet length to send to or receive from the server. (Default value is 16MB.)

- `max_join_size`

  The automatic limit for rows in a join when using `--safe-updates`. (Default value is 1,000,000.)

- `net_buffer_length`

  The buffer size for TCP/IP and socket communication. (Default value is 16KB.)

- `select_limit`

  The automatic limit for `SELECT` statements when using `--safe-updates`. (Default value is 1,000.)

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. However, this syntax is deprecated as of MySQL 4.0.

On Unix, the **mysql** client writes a record of executed statements to a history file. By default, the history file is named `.mysql_history` and is created in your home directory. To specify a different file, set the value of the `MYSQL_HISTFILE` environment variable.

If you do not want to maintain a history file, first remove `.mysql_history` if it exists, and then use either of the following techniques:

- Set the `MYSQL_HISTFILE` variable to `/dev/null`. To cause this setting to take effect each time you log in, put the setting in one of your shell's startup files.

- Create `.mysql_history` as a symbolic link to `/dev/null`:

  ```
  shell> ln -s /dev/null $HOME/.mysql_history
  ```

  You need do this only once.

8.3.1. mysql Commands

mysql sends SQL statements that you issue to the server to be executed. There is also a set of commands that **mysql** itself interprets. For a list of these commands, type `help` or `\h` at the `mysql>` prompt:

```
mysql> help

MySQL commands:
?         (\h)    Synonym for `help'.
clear     (\c)    Clear command.
connect   (\r)    Reconnect to the server.
                  Optional arguments are db and host.
delimiter (\d)    Set query delimiter.
edit      (\e)    Edit command with $EDITOR.
ego       (\G)    Send command to mysql server,
                  display result vertically.
exit      (\q)    Exit mysql. Same as quit.
go        (\g)    Send command to mysql server.
help      (\h)    Display this help.
nopager   (\n)    Disable pager, print to stdout.
notee     (\t)    Don't write into outfile.
pager     (\P)    Set PAGER [to_pager].
                  Print the query results via PAGER.
print     (\p)    Print current command.
prompt    (\R)    Change your mysql prompt.
quit      (\q)    Quit mysql.
rehash    (\#)    Rebuild completion hash.
source    (\.)    Execute an SQL script file.
                  Takes a file name as an argument.
status    (\s)    Get status information from the server.
system    (\!)    Execute a system shell command.
tee       (\T)    Set outfile [to_outfile].
                  Append everything into given outfile.
use       (\u)    Use another database.
                  Takes database name as argument.
```

Each command has both a long and short form. The long form is not case sensitive; the short form is. The long form can be followed by an optional semicolon terminator, but the short form should not.

The `edit`, `nopager`, `pager`, and `system` commands work only in Unix.

The `status` command provides some information about the connection and the server you are using. If you are running in `--safe-updates` mode, `status` also prints the values for the **mysql** variables that affect your queries.

To log queries and their output, use the `tee` command. All the data displayed on the screen is appended into a given file. This can be very useful for debugging purposes also. You can enable this feature on the command line with the `--tee` option, or interactively with the `tee` command. The `tee` file can be disabled interactively with the `notee` command. Executing `tee` again re-enables logging. Without a parameter, the previous file is used. Note that `tee` flushes query results to the file after each statement, just before **mysql** prints its next prompt.

Browsing or searching query results in interactive mode by using Unix programs such as **less**, **more**, or any other similar program is possible with the `--pager` option. If you specify no value for the option, **mysql** checks the value of the `PAGER` environment variable and sets the pager to that. Output paging can be enabled interactively with the `pager` command and disabled with `nopager`. The command takes an optional argument; if given, the paging program is set to that. With no argument, the pager is set to the pager that was set on the command line, or `stdout` if no pager was specified.

Output paging works only in Unix because it uses the `popen()` function, which doesn't exist on Windows. For Windows, the `tee` option can be used instead to save query output, although this is not as convenient as `pager` for browsing output in some situations.

A few tips about the `pager` command:

- You can use it to write to a file and the results go only to the file:

  ```
  mysql> pager cat > /tmp/log.txt
  ```

  You can also pass any options for the program that you want to use as your pager:

  ```
  mysql> pager less -n -i -S
  ```

- In the preceding example, note the `-S` option. You may find it very useful for browsing wide query results. Sometimes a very wide result set is difficult to read on the screen. The `-S` option to **less** can make the result set much more readable because you can scroll it horizontally using the left-arrow and right-arrow keys. You can also use `-S` interactively within **less** to switch the horizontal-browse mode on and off. For more information, read the **less** manual page:

  ```
  shell> man less
  ```

- You can specify very complex pager commands for handling query output:

  ```
  mysql> pager cat | tee /dr1/tmp/res.txt \
            | tee /dr2/tmp/res2.txt | less -n -i -S
  ```

  In this example, the command would send query results to two files in two different directories on two different filesystems mounted on `/dr1` and `/dr2`, yet still display the results onscreen via **less**.

You can also combine the `tee` and `pager` functions. Have a **tee** file enabled and `pager` set to **less**, and you are able to browse the results using the **less** program and still have everything appended into a file the same time. The difference between the Unix **tee** used with the `pager` command and the **mysql** built-in `tee` command is that the built-in `tee` works even if you don't have the Unix **tee** available. The built-in `tee` also logs everything that is printed on the screen, whereas the Unix **tee** used with `pager` doesn't log quite that much. Additionally, `tee` file logging can be turned on and off interactively from within **mysql**. This is useful when you want to log some queries to a file, but not others.

From MySQL 4.0.2 on, the default `mysql>` prompt can be reconfigured. The string for defining the prompt can contain the following special sequences:

Option	Description
`\v`	The server version
`\d`	The current database
`\h`	The server host

`\p`	The current TCP/IP host
`\u`	Your username
`\U`	Your full *`user_name@host_name`* account name
`\\`	A literal '`\`' backslash character
`\n`	A newline character
`\t`	A tab character
`\`	A space (a space follows the backslash)
`\_`	A space
`\R`	The current time, in 24-hour military time (0-23)
`\r`	The current time, standard 12-hour time (1-12)
`\m`	Minutes of the current time
`\y`	The current year, two digits
`\Y`	The current year, four digits
`\D`	The full current date
`\s`	Seconds of the current time
`\w`	The current day of the week in three-letter format (Mon, Tue, ...)
`\P`	am/pm
`\o`	The current month in numeric format
`\O`	The current month in three-letter format (Jan, Feb, ...)
`\c`	A counter that increments for each statement you issue
`\S`	Semicolon
`\'`	Single quote
`\"`	Double quote

'`\`' followed by any other letter just becomes that letter.

If you specify the `prompt` command with no argument, **mysql** resets the prompt to the default of `mysql>`.

You can set the prompt in several ways:

- Use an environment variable

  You can set the `MYSQL_PS1` environment variable to a prompt string. For example:

  ```
  shell> export MYSQL_PS1="(\u@\h) [\d]> "
  ```

- Use an option file

  You can set the `prompt` option in the `[mysql]` group of any MySQL option file, such as `/etc/my.cnf` or the `.my.cnf` file in your home directory. For example:

  ```
  [mysql]
  prompt=(\\u@\\h) [\\d]>\\_
  ```

  In this example, note that the backslashes are doubled. If you set the prompt using the `prompt` op-

tion in an option file, it is advisable to double the backslashes when using the special prompt options. There is some overlap in the set of allowable prompt options and the set of special escape sequences that are recognized in option files. (These sequences are listed in Section 4.3.2, “Using Option Files”.) The overlap may cause you problems if you use single backslashes. For example, `\s` is interpreted as a space rather than as the current seconds value. The following example shows how to define a prompt within an option file to include the current time in `HH:MM:SS>` format:

```
[mysql]
prompt="\\r:\\m:\\s> "
```

- Use a command-line option

  You can set the `--prompt` option on the command line to **mysql**. For example:

```
shell> mysql --prompt="(\u@\h) [\d]> "
(user@host) [database]>
```

- Interactively

  You can change your prompt interactively by using the `prompt` (or `\R`) command. For example:

```
mysql> prompt (\u@\h) [\d]>\_
PROMPT set to '(\u@\h) [\d]>\_'
(user@host) [database]>
(user@host) [database]> prompt
Returning to default PROMPT of mysql>
mysql>
```

8.3.2. Executing SQL Statements from a Text File

The **mysql** client typically is used interactively, like this:

```
shell> mysql db_name
```

However, it's also possible to put your SQL statements in a file and then tell **mysql** to read its input from that file. To do so, create a text file `text_file` that contains the statements you wish to execute. Then invoke **mysql** as shown here:

```
shell> mysql db_name < text_file
```

You can also start your text file with a `USE db_name` statement. In this case, it is unnecessary to specify the database name on the command line:

```
shell> mysql < text_file
```

If you are running **mysql**, you can execute an SQL script file using the `source` or `\.` command:

```
mysql> source filename
mysql> \. filename
```

Sometimes you may want your script to display progress information to the user; for this you can insert some lines like

```
SELECT '<info_to_display>' AS ' ';
```

which outputs <info_to_display>.

For more information about batch mode, see Section 3.5, “Using **mysql** in Batch Mode”.

8.3.3. mysql Tips

This section describes some techniques that can help you use **mysql** more effectively.

8.3.3.1. Displaying Query Results Vertically

Some query results are much more readable when displayed vertically, instead of in the usual horizontal table format. Queries can be displayed vertically by terminating the query with \G instead of a semicolon. For example, longer text values that include newlines often are much easier to read with vertical output:

```
mysql> SELECT * FROM mails WHERE LENGTH(txt) < 300 LIMIT 300,1\G
*************************** 1. row ***************************
  msg_nro: 3068
     date: 2000-03-01 23:29:50
time_zone: +0200
mail_from: Monty
    reply: monty@no.spam.com
  mail_to: "Thimble Smith" <tim@no.spam.com>
      sbj: UTF-8
      txt: >>>>> "Thimble" == Thimble Smith writes:

Thimble> Hi.  I think this is a good idea.  Is anyone familiar
Thimble> with UTF-8 or Unicode? Otherwise, I'll put this on my
Thimble> TODO list and see what happens.

Yes, please do that.

Regards,
Monty
     file: inbox-jani-1
     hash: 190402944
1 row in set (0.09 sec)
```

8.3.3.2. Using the `--safe-updates` Option

For beginners, a useful startup option is `--safe-updates` (or `--i-am-a-dummy`, which has the same effect). This option was introduced in MySQL 3.23.11. It is helpful for cases when you might have issued a `DELETE FROM` *`tbl_name`* statement but forgotten the `WHERE` clause. Normally, such a statement deletes all rows from the table. With `--safe-updates`, you can delete rows only by specifying the key values that identify them. This helps prevent accidents.

When you use the `--safe-updates` option, **mysql** issues the following statement when it connects to the MySQL server:

```
SET SQL_SAFE_UPDATES=1,SQL_SELECT_LIMIT=1000, SQL_MAX_JOIN_SIZE=1000000;
```

See Section 13.5.3, “`SET` Syntax”.

The `SET` statement has the following effects:

- You are not allowed to execute an `UPDATE` or `DELETE` statement unless you specify a key constraint in the `WHERE` clause or provide a `LIMIT` clause (or both). For example:

  ```
  UPDATE tbl_name SET not_key_column=# WHERE key_column=#;
  ```

  ```
  UPDATE tbl_name SET not_key_column=# LIMIT 1;
  ```

- All large `SELECT` results are automatically limited to 1,000 rows unless the statement includes a `LIMIT` clause.
- Multiple-table `SELECT` statements that probably need to examine more than 1,000,000 row combinations are aborted.

To specify limits other than 1,000 and 1,000,000, you can override the defaults by using `--select_limit` and `--max_join_size` options:

```
shell> mysql --safe-updates --select_limit=500 --max_join_size=10000
```

8.3.3.3. Disabling mysql Auto-Reconnect

If the **mysql** client loses its connection to the server while sending a query, it immediately and automatically tries to reconnect once to the server and send the query again. However, even if **mysql** succeeds in reconnecting, your first connection has ended and all your previous session objects and settings are lost: temporary tables, the autocommit mode, and user and session variables. This behavior may be dangerous for you, as in the following example where the server was shut down and restarted without you knowing it:

```
mysql> SET @a=1;
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO t VALUES(@a);
ERROR 2006: MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    1
Current database: test

Query OK, 1 row affected (1.30 sec)

mysql> SELECT * FROM t;
+------+
| a    |
+------+
| NULL |
+------+
1 row in set (0.05 sec)
```

The `@a` user variable has been lost with the connection, and after the reconnection it is undefined. If it is important to have **mysql** terminate with an error if the connection has been lost, you can start the **mysql** client with the `--skip-reconnect` option.

8.4. mysqladmin, Administering a MySQL Server

mysqladmin is a client for performing administrative operations. You can use it to check the server's

configuration and current status, create and drop databases, and more.

Invoke **mysqladmin** like this:

```
shell> mysqladmin [options] command [command-option] command ...
```

mysqladmin supports the following commands:

- `create db_name`

  Create a new database named *db_name*.

- `debug`

  Tell the server to write debug information to the error log.

- `drop db_name`

  Delete the database named *db_name* and all its tables.

- `extended-status`

  Display the server status variables and their values.

- `flush-hosts`

  Flush all information in the host cache.

- `flush-logs`

  Flush all logs.

- `flush-privileges`

  Reload the grant tables (same as `reload`).

- `flush-status`

  Clear status variables.

- `flush-tables`

  Flush all tables.

- `flush-threads`

  Flush the thread cache. (Added in MySQL 3.23.16.)

- `kill id,id,...`

  Kill server threads.

- `old-password new-password`

  This is like the `password` command but stores the password using the old (pre-4.1) password-hashing format. This command was added in MySQL 4.1.0.

- `password new-password`

Set a new password. This changes the password to `new-password` for the account that you use with **mysqladmin** for connecting to the server.

If *`new-password`* contains spaces or other characters that are special to your command interpreter, you need to enclose it within quotes. On Windows, be sure to use double quotes rather than single quotes; single quotes are not be stripped from the password, they are interpreted as part of the password. For example:

shell> mysqladmin password "my new password"

- `ping`

  Check whether the server is alive. The return status from **mysqladmin** is 0 if the server is running, 1 if it is not. Beginning with MySQL 4.0.22, the status is 0 even in case of an error such as `Access denied`, because that means the server is running but disallowed the connection, which is different from the server not running.

- `processlist`

  Show a list of active server threads. This is like the output of the `SHOW PROCESSLIST` statement. If the `--verbose` option is given, the output is like that of `SHOW FULL PROCESSLIST`.

- `reload`

  Reload the grant tables.

- `refresh`

  Flush all tables and close and open log files.

- `shutdown`

  Stop the server.

- `start-slave`

  Start replication on a slave server. (Added in MySQL 3.23.16.)

- `status`

  Display a short server status message.

- `stop-slave`

  Stop replication on a slave server. (Added in MySQL 3.23.16.)

- `variables`

  Display the server system variables and their values.

- `version`

  Display version information from the server.

All commands can be shortened to any unique prefix. For example:

```
shell> mysqladmin proc stat
+----+-------+-----------+----+-------------+------+-------+------+
```

```
| Id | User  | Host      | db | Command     | Time | State | Info |
+----+-------+-----------+----+-------------+------+-------+------+
| 6  | monty | localhost |    | Processlist | 0    |       |      |
+----+-------+-----------+----+-------------+------+-------+------+
Uptime: 10077  Threads: 1  Questions: 9  Slow queries: 0
Opens: 6 Flush tables: 1  Open tables: 2
Memory in use: 1092K  Max memory used: 1116K
```

The **mysqladmin status** command result displays the following values:

- `Uptime`

  The number of seconds the MySQL server has been running.

- `Threads`

  The number of active threads (clients).

- `Questions`

  The number of questions (queries) from clients since the server was started.

- `Slow queries`

  The number of queries that have taken more than `long_query_time` seconds. See Section 5.9.5, “The Slow Query Log”.

- `Opens`

  The number of tables the server has opened.

- `Flush tables`

  The number of `flush ...`, `refresh`, and `reload` commands the server has executed.

- `Open tables`

  The number of tables that currently are open.

- `Memory in use`

  The amount of memory allocated directly by **mysqld** code. This value is displayed only when MySQL has been compiled with `--with-debug=full`.

- `Maximum memory used`

  The maximum amount of memory allocated directly by **mysqld** code. This value is displayed only when MySQL has been compiled with `--with-debug=full`.

If you execute **mysqladmin shutdown** when connecting to a local server using a Unix socket file,

mysqladmin waits until the server's process ID file has been removed, to ensure that the server has stopped properly.

mysqladmin supports the following options:

- `--help, -?`

  Display a help message and exit.

- `--character-sets-dir=path`

  The directory where character sets are installed. See Section 5.8.1, “The Character Set Used for Data and Sorting”.

- `--compress, -C`

  Compress all information sent between the client and the server if both support compression.

- `--count=#, -c #`

  The number of iterations to make. This works only with `--sleep` (`-i`).

- `--debug[=debug_options], -# [debug_options]`

  Write a debugging log. The *`debug_options`* string often is `'d:t:o,file_name'`. The default is `'d:t:o,/tmp/mysqladmin.trace'`.

- `--default-character-set=charset`

  Use *`charset`* as the default character set. See Section 5.8.1, “The Character Set Used for Data and Sorting”. Added in MySQL 4.1.9.

- `--force, -f`

  Don't ask for confirmation for the `drop database` command. With multiple commands, continue even if an error occurs.

- `--host=host_name, -h host_name`

  Connect to the MySQL server on the given host.

- `--password[=password], -p[password]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *`password`* value following the `--password` or `-p` option on the command line, you are prompted for one.

- `--port=port_num, -P port_num`

  The TCP/IP port number to use for the connection.

- `--protocol={TCP | SOCKET | PIPE | MEMORY}`

  The connection protocol to use. New in MySQL 4.1.

- `--relative, -r`

  Show the difference between the current and previous values when used with `-i`. Currently, this option works only with the `extended-status` command.

- `--silent, -s`

  Exit silently if a connection to the server cannot be established.

- `--sleep=delay, -i delay`

  Execute commands again and again, sleeping for *delay* seconds in between.

- `--socket=path, -S path`

  The socket file to use for the connection.

- `--user=user_name, -u user_name`

  The MySQL username to use when connecting to the server.

- `--verbose, -v`

  Verbose mode. Print out more information on what the program does.

- `--version, -V`

  Display version information and exit.

- `--vertical, -E`

  Print output vertically. This is similar to `--relative`, but prints output vertically.

- `--wait[=#], -w[#]`

  If the connection cannot be established, wait and retry instead of aborting. If an option value is given, it indicates the number of times to retry. The default is one time.

You can also set the following variables by using `--var_name=value` options:

- `connect_timeout`

  The maximum number of seconds before connection timeout. The default value is 43200 (12 hours).

- `shutdown_timeout`

  The maximum number of seconds to wait for shutdown. The default value is 3600 (1 hour).

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. However, this syntax is deprecated as of MySQL 4.0.

8.5. The mysqlbinlog Binary Log Utility

The binary log files that the server generates are written in binary format. To examine these files in text format, use the **mysqlbinlog** utility. It is available as of MySQL 3.23.14.

Invoke **mysqlbinlog** like this:

```
shell> mysqlbinlog [options] log-file ...
```

For example, to display the contents of the binary log `binlog.000003`, use this command:

```
shell> mysqlbinlog binlog.0000003
```

The output includes all statements contained in `binlog.000003`, together with other information such as the time each statement took, the thread ID of the client that issued it, the timestamp when it was issued, and so forth.

Normally, you use **mysqlbinlog** to read binary log files directly and apply them to the local MySQL server. It is also possible to read binary logs from a remote server by using the `--read-from-remote-server` option.

When you read remote binary logs, the connection parameter options can be given to indicate how to connect to the server, but they are ignored unless you also specify the `--read-from-remote-server` option. These options are `--host`, `--password`, `--port`, `--protocol`, `--socket`, and `--user`.

You can also use **mysqlbinlog** to read relay log files written by a slave server in a replication setup. Relay logs have the same format as binary log files.

The binary log is discussed further in Section 5.9.4, “The Binary Log”.

mysqlbinlog supports the following options:

- `--help, -?`

  Display a help message and exit.

- `--database=`*`db_name`*`, -d `*`db_name`*

  List entries for just this database (local log only).

- `--force-read, -f`

  With this option, if **mysqlbinlog** reads a binary log event that it does not recognize, it prints a warning, ignores the event, and continues. Without this option, **mysqlbinlog** stops if it reads such an event.

- `--host=`*`host_name`*`, -h `*`host_name`*

  Get the binary log from the MySQL server on the given host.

- `--local-load=`*`path`*`, -l `*`path`*

  Prepare local temporary files for `LOAD DATA INFILE` in the specified directory.

- `--offset=`*`N`*`, -o `*`N`*

  Skip the first *`N`* entries.

- `--password[=`*`password`*`], -p[`*`password`*`]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *`password`* value following the `--password` or `-p` option on the command line, you are prompted for one.

- `--port=`*`port_num`*`, -P `*`port_num`*

The TCP/IP port number to use for connecting to a remote server.

- `--position=N, -j N`

  Deprecated, use `--start-position` instead (starting from MySQL 4.1.4).

- `--protocol={TCP | SOCKET | PIPE | MEMORY}`

  The connection protocol to use. New in MySQL 4.1.

- `--read-from-remote-server, -R`

  Read the binary log from a MySQL server. Any connection parameter options are ignored unless this option is given as well. These options are `--host`, `--password`, `--port`, `--protocol`, `--socket`, and `--user`.

- `--result-file=name, -r name`

  Direct output to the given file.

- `--short-form, -s`

  Display only the statements contained in the log, without any extra information.

- `--socket=path, -S path`

  The socket file to use for the connection.

- `--start-datetime=datetime`

  Start reading the binary log at the first event having a datetime equal to or later than the `datetime` argument. The *`datetime`* value is relative to the local time zone on the machine where you run **mysqlbinlog**. The value should be in a format accepted for the `DATETIME` or `TIMESTAMP` data types. For example:

  ```
  shell> mysqlbinlog --start-datetime="2004-12-25 11:25:56" binlog.000003
  ```

  This option is available as of MySQL 4.1.4. It is useful for point-in-time recovery.

- `--stop-datetime=datetime`

  Stop reading the binary log at the first event having a datetime equal or posterior to the `datetime` argument. See the description of the `--start-datetime` option for information about the `datetime value`. This option is available as of MySQL 4.1.4. It is useful for point-in-time recovery.

- `--start-position=N`

  Start reading the binary log at the first event having a position equal to the `N` argument. Available as of MySQL 4.1.4 (previously named `--position`).

- `--stop-position=N`

  Stop reading the binary log at the first event having a position equal or greater than the `N` argument. Available as of MySQL 4.1.4.

- `--to-last-log, -t`

Do not stop at the end of the requested binary log of the MySQL server, but rather continue printing until the end of the last binary log. If you send the output to the same MySQL server, this may lead to an endless loop. This option requires `--read-from-remote-server`. Available as of MySQL 4.1.2.

- `--disable-log-bin, -D`

  Disable binary logging. This is useful for avoiding an endless loop if you use the `--to-last-log` option and are sending the output to the same MySQL server. This option also is useful when restoring after a crash to avoid duplication of the statements you have logged. **Note**: This option requires that you have the `SUPER` privilege. Available as of MySQL 4.1.8.

- `--user=`*`user_name`*`, -u `*`user_name`*

  The MySQL username to use when connecting to a remote server.

- `--version, -V`

  Display version information and exit.

You can also set the following variable by using `--`*`var_name=value`* options:

- `open_files_limit`

  Specify the number of open file descriptors to reserve.

You can pipe the output of **mysqlbinlog** into a **mysql** client to execute the statements contained in the binary log. This is used to recover from a crash when you have an old backup (see Section 5.7.1, “Database Backups”):

```
shell> mysqlbinlog hostname-bin.000001 | mysql
```

Or:

```
shell> mysqlbinlog hostname-bin.[0-9]* | mysql
```

You can also redirect the output of **mysqlbinlog** to a text file instead, if you need to modify the statement log first (for example, to remove statements that you don't want to execute for some reason). After editing the file, execute the statements that it contains by using it as input to the **mysql** program.

mysqlbinlog has the `--position` option, which prints only those statements with an offset in the binary log greater than or equal to a given position (the given position must match the start of one event). It also has options to stop or start when it sees an event of a given date and time. This enables you to perform point-in-time recovery using the `--stop-datetime` option (to be able to say, for example, "roll forward my databases to how they were today at 10:30 AM").

If you have more than one binary log to execute on the MySQL server, the safe method is to process them all using a single connection to the server. Here is an example that demonstrates what may be *unsafe*:

```
shell> mysqlbinlog hostname-bin.000001 | mysql # DANGER!!
shell> mysqlbinlog hostname-bin.000002 | mysql # DANGER!!
```

Processing binary logs this way using different connections to the server causes problems if the first log file contains a `CREATE TEMPORARY TABLE` statement and the second log contains a statement that uses the temporary table. When the first **mysql** process terminates, the server drops the temporary table. When the second **mysql** process attempts to use the table, the server reports ``unknown table."

To avoid problems like this, use a single connection to execute the contents of all binary logs that you want to process. Here is one way to do that:

```
shell> mysqlbinlog hostname-bin.000001 hostname-bin.000002 | mysql
```

Another approach is to do this:

```
shell> mysqlbinlog hostname-bin.000001 >  /tmp/statements.sql
shell> mysqlbinlog hostname-bin.000002 >> /tmp/statements.sql
shell> mysql -e "source /tmp/statements.sql"
```

In MySQL 3.23, the binary log did not contain the data to load for `LOAD DATA INFILE` statements. To execute such a statement from a binary log file, the original data file was needed. Starting from MySQL 4.0.14, the binary log does contain the data, so **mysqlbinlog** can produce output that reproduces the `LOAD DATA INFILE` operation without the original data file. **mysqlbinlog** copies the data to a temporary file and writes a `LOAD DATA LOCAL INFILE` statement that refers to the file. The default location of the directory where these files are written is system-specific. To specify a directory explicitly, use the `--local-load` option.

Because **mysqlbinlog** converts `LOAD DATA INFILE` statements to `LOAD DATA LOCAL INFILE` statements (that is, it adds `LOCAL`), both the client and the server that you use to process the statements must be configured to allow `LOCAL` capability. See Section 5.4.4, “Security Issues with `LOAD DATA LOCAL`”.

Warning: The temporary files created for `LOAD DATA LOCAL` statements are *not* automatically deleted because they are needed until you actually execute those statements. You should delete the temporary files yourself after you no longer need the statement log. The files can be found in the temporary file directory and have names like `original_file_name-#-#`.

In the future, we will fix this problem by allowing **mysqlbinlog** to connect directly to a **mysqld** server. Then it is possible to safely remove the log files automatically as soon as the `LOAD DATA INFILE` statements have been executed.

Before MySQL 4.1, **mysqlbinlog** could not prepare output suitable for **mysql** if the binary log contained interlaced statements originating from different clients that used temporary tables of the same name. This is fixed in MySQL 4.1. However, the problem still existed for `LOAD DATA INFILE` statements until it was fixed in MySQL 4.1.8.

8.6. mysqlcc, the MySQL Control Center

mysqlcc, the MySQL Control Center, is a platform-independent client that provides a graphical user interface (GUI) to the MySQL database server. It supports interactive use, including syntax highlighting and tab completion. It provides database and table management, and allows server administration.

mysqlcc is deprecated and it is recommended that users choose the new MySQL Administrator and MySQL Query Browser, found at http://dev.mysql.com/downloads/.

Currently, **mysqlcc** runs on Windows and Linux platforms.

Invoke **mysqlcc** by double-clicking its icon in a graphical environment. From the command line, invoke it like this:

```
shell> mysqlcc [options]
```

mysqlcc supports the following options:

- `--help, -?`

  Display a help message and exit.

- `--blocking_queries, -b`

  Use blocking queries.

- `--compress, -C`

  Compress all information sent between the client and the server if both support compression.

- `--connection_name=name, -c name`

  This option is a synonym for `--server`.

- `--database=db_name, -d db_name`

  The database to use. This is useful mainly in an option file.

- `--history_size=#, -H #`

  The history size for the query window.

- `--host=host_name, -h host_name`

  Connect to the MySQL server on the given host.

- `--local-infile[={0|1}]`

  Enable or disable `LOCAL` capability for `LOAD DATA INFILE`. With no value, the option enables `LOCAL`. It may be given as `--local-infile=0` or `--local-infile=1` to explicitly disable or enable `LOCAL`. Enabling `LOCAL` has no effect if the server does not also support it.

- `--password[=password], -p[password]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *`password`* value following the `--password` or `-p` option on the command line, you are prompted for one.

- `--plugins_path=name, -g name`

  The path to the directory where MySQL Control Center plugins are located.

- `--port=port_num, -P port_num`

The TCP/IP port number to use for the connection.

- `--query, -q`

  Open a query window on startup.

- `--register, -r`

  Open the Register Server dialog on startup.

- `--server=name, -s name`

  The MySQL Control Center connection name.

- `--socket=path, -S path`

  The socket file to use for the connection.

- `--syntax, -y`

  Enable syntax highlighting and completion.

- `--syntax_file=name, -Y name`

  The syntax file for completion.

- `--translations_path=name, -T name`

  The path to the directory where MySQL Control Center translations are located.

- `--user=user_name, -u user_name`

  The MySQL username to use when connecting to the server.

- `--version, -V`

  Display version information and exit.

You can also set the following variables by using `--var_name=value` options:

- `connect_timeout`

  The number of seconds before connection timeout. (Default value is 0.)

- `max_allowed_packet`

  The maximum packet length to send to or receive from the server. (Default value is 16MB.)

- `max_join_size`

The automatic limit for rows in a join. (Default value is 1,000,000.)

- `net_buffer_length`

  The buffer size for TCP/IP and socket communication. (Default value is 16KB.)

- `select_limit`

  The automatic limit for `SELECT` statements. (Default value is 1,000.)

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. However, this syntax is deprecated as of MySQL 4.0.

8.7. The mysqlcheck Table Maintenance and Repair Program

The **mysqlcheck** client checks and repairs `MyISAM` tables. It can also optimize and analyze tables. **mysqlcheck** is available as of MySQL 3.23.38.

mysqlcheck is similar in function to **myisamchk**, but works differently. The main operational difference is that **mysqlcheck** must be used when the **mysqld** server is running, whereas **myisamchk** should be used when it is not. The benefit of using **mysqlcheck** is that you do not have to stop the server to check or repair your tables.

mysqlcheck uses the SQL statements `CHECK TABLE`, `REPAIR TABLE`, `ANALYZE TABLE`, and `OPTIMIZE TABLE` in a convenient way for the user. It determines which statements to use for the operation you want to perform, then sends the statements to the server to be executed.

There are three general ways to invoke **mysqlcheck**:

```
shell> mysqlcheck [options] db_name [tables]
shell> mysqlcheck [options] --databases DB1 [DB2 DB3...]
shell> mysqlcheck [options] --all-databases
```

If you don't name any tables or use the `--databases` or `--all-databases` option, entire databases are checked.

mysqlcheck has a special feature compared to the other clients. The default behavior of checking tables (`--check`) can be changed by renaming the binary. If you want to have a tool that repairs tables by default, you should just make a copy of **mysqlcheck** named **mysqlrepair**, or make a symbolic link to **mysqlcheck** named **mysqlrepair**. If you invoke `mysqlrepair`, it repairs tables on command.

The following names can be used to change **mysqlcheck** default behavior:

mysqlrepair	The default option is `--repair`
mysqlanalyze	The default option is `--analyze`
mysqloptimize	The default option is `--optimize`

mysqlcheck supports the following options:

- `--help, -?`

Display a help message and exit.

- `--all-databases, -A`

  Check all tables in all databases. This is the same as using the `--databases` option and naming all the databases on the command line.

- `--all-in-1, -1`

  Instead of issuing a statement for each table, execute a single statement for each database that names all the tables from that database to be processed.

- `--analyze, -a`

  Analyze the tables.

- `--auto-repair`

  If a checked table is corrupted, automatically fix it. Any necessary repairs are done after all tables have been checked.

- `--character-sets-dir=`*`path`*

  The directory where character sets are installed. See Section 5.8.1, “The Character Set Used for Data and Sorting”.

- `--check, -c`

  Check the tables for errors.

- `--check-only-changed, -C`

  Check only tables that have changed since the last check or that haven't been closed properly.

- `--compress`

  Compress all information sent between the client and the server if both support compression.

- `--databases, -B`

  Process all tables in the named databases. With this option, all name arguments are regarded as database names, not as table names.

- `--debug[=`*`debug_options`*`], -# [`*`debug_options`*`]`

  Write a debugging log. The *`debug_options`* string often is `'d:t:o,`*`file_name`*`'`.

- `--default-character-set=`*`charset`*

  Use *`charset`* as the default character set. See Section 5.8.1, “The Character Set Used for Data and Sorting”.

- `--extended, -e`

  If you are using this option to check tables, it ensures that they are 100% consistent but takes a long time.

  If you are using this option to repair tables, it runs an extended repair that may not only take a long time to execute, but may produce a lot of garbage rows also!

- `--fast, -F`

  Check only tables that haven't been closed properly.

- `--force, -f`

  Continue even if an SQL error occurs.

- `--host=host_name, -h host_name`

  Connect to the MySQL server on the given host.

- `--medium-check, -m`

  Do a check that is faster than an `--extended` operation. This finds only 99.99% of all errors, which should be good enough in most cases.

- `--optimize, -o`

  Optimize the tables.

- `--password[=password], -p[password]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *`password`* value following the `--password` or `-p` option on the command line, you are prompted for one.

- `--port=port_num, -P port_num`

  The TCP/IP port number to use for the connection.

- `--protocol={TCP | SOCKET | PIPE | MEMORY}`

  The connection protocol to use. New in MySQL 4.1.

- `--quick, -q`

  If you are using this option to check tables, it prevents the check from scanning the rows to check for incorrect links. This is the fastest check method.

  If you are using this option to repair tables, it tries to repair only the index tree. This is the fastest repair method.

- `--repair, -r`

  Do a repair that can fix almost anything except unique keys that aren't unique.

- `--silent, -s`

  Silent mode. Print only error messages.

- `--socket=path, -S path`

  The socket file to use for the connection.

- `--tables`

  Overrides the `--databases` or `-B` option. All arguments following the option are regarded as table names.

- `--user=user_name, -u user_name`

  The MySQL username to use when connecting to the server.

- `--verbose, -v`

  Verbose mode. Print information about the various stages of program operation.

- `--version, -V`

  Display version information and exit.

8.8. The mysqldump Database Backup Program

The **mysqldump** client can be used to dump a database or a collection of databases for backup or for transferring the data to another SQL server (not necessarily a MySQL server). The dump contains SQL statements to create the table and/or populate the table.

If you are doing a backup on the server, and your tables all are `MyISAM` tables, you could consider using the **mysqlhotcopy** instead (faster backup, faster restore). See Section 8.9, “The **mysqlhotcopy** Database Backup Program”.

There are three general ways to invoke **mysqldump**:

```
shell> mysqldump [options] db_name [tables]
shell> mysqldump [options] --databases DB1 [DB2 DB3...]
shell> mysqldump [options] --all-databases
```

If you don't name any tables or use the `--databases` or `--all-databases` option, entire databases are dumped.

To get a list of the options your version of **mysqldump** supports, execute **mysqldump --help**.

If you run **mysqldump** without the `--quick` or `--opt` option, **mysqldump** loads the whole result set into memory before dumping the result. This probably is a problem if you are dumping a big database. As of MySQL 4.1, `--opt` is on by default, but can be disabled with `--skip-opt`.

If you are using a recent copy of the **mysqldump** program to generate a dump to be reloaded into a very old MySQL server, you should not use the `--opt` or `-e` options.

Before MySQL 4.1.2, out-of-range numeric values such as `-inf` and `inf`, as well as NaN (not-a-number) values are dumped by **mysqldump** as `NULL`. You can see this using the following sample table:

```
mysql> CREATE TABLE t (f DOUBLE);
mysql> INSERT INTO t VALUES(1e+111111111111111111111);
mysql> INSERT INTO t VALUES(-1e111111111111111111111);
mysql> SELECT f FROM t;
+------+
| f    |
+------+
|  inf |
| -inf |
+------+
```

For this table, **mysqldump** produces the following data output:

```
--
-- Dumping data for table `t`
--

INSERT INTO t VALUES (NULL);
INSERT INTO t VALUES (NULL);
```

The significance of this behavior is that if you dump and restore the table, the new table has contents that differ from the original contents. This problem is fixed as of MySQL 4.1.2; you cannot insert `inf` in the table, so this **mysqldump** behavior is only relevant when you deal with old servers.

mysqldump supports the following options:

- `--help, -?`

  Display a help message and exit.

- `--add-drop-table`

  Add a `DROP TABLE` statement before each `CREATE TABLE` statement.

- `--add-locks`

  Surround each table dump with `LOCK TABLES` and `UNLOCK TABLES` statements. This results in faster inserts when the dump file is reloaded. See Section 7.2.14, “Speed of `INSERT` Statements”.

- `--all-databases, -A`

  Dump all tables in all databases. This is the same as using the `--databases` option and naming all the databases on the command line.

- `--allow-keywords`

  Allow creation of column names that are keywords. This works by prefixing each column name with the table name.

- `--comments[={0|1}]`

  If set to `0`, suppresses additional information in the dump file such as program version, server version, and host. `--skip-comments` has the same effect as `--comments=0`. The default value is `1` to not suppress the extra information. New in MySQL 4.0.17.

- `--compact`

  Produce less verbose output. This option suppresses comments and enables the `--skip-add-drop-table`, `--no-set-names`, `--skip-disable-keys`, and `--skip-add-locks` options. New in MySQL 4.1.2.

- `--compatible=`*`name`*

  Produce output that is compatible with other database systems or with older MySQL servers. The value of `name` can be `ansi`, `mysql323`, `mysql40`, `postgresql`, `oracle`, `mssql`, `db2`, `maxdb`, `no_key_options`, `no_table_options`, or `no_field_options`. To use several values, separate them by commas. These values have the same meaning as the corresponding options for setting the server SQL mode. See Section 5.2.2, “The Server SQL Mode”.

This option requires a server version of 4.1.0 or higher. With older servers, it does nothing.

- `--complete-insert, -c`

  Use complete `INSERT` statements that include column names.

- `--compress, -C`

  Compress all information sent between the client and the server if both support compression.

- `--create-options`

  Include all MySQL-specific table options in the `CREATE TABLE` statements. Before MySQL 4.1.2, use `--all` instead.

- `--databases, -B`

  Dump several databases. Normally, **mysqldump** treats the first name argument on the command line as a database name and following names as table names. With this option, it treats all name arguments as database names. `CREATE DATABASE IF NOT EXISTS` *`db_name`* and `USE` *`db_name`* statements are included in the output before each new database.

- `--debug[=`*`debug_options`*`], -# [`*`debug_options`*`]`

  Write a debugging log. The *`debug_options`* string often is `'d:t:o,`*`file_name`*`'`.

- `--default-character-set=`*`charset`*

  Use *`charset`* as the default character set. See Section 5.8.1, “The Character Set Used for Data and Sorting”. If not specified, **mysqldump** from MySQL 4.1.2 or later uses `utf8`, and earlier versions use `latin1`.

- `--delayed-insert`

  Insert rows using `INSERT DELAYED` statements.

- `--delete-master-logs`

  On a master replication server, delete the binary logs after performing the dump operation. This option automatically enables `--first-slave` before MySQL 4.1.8 and enables `--master-data` thereafter. It was added in MySQL 3.23.57 (for MySQL 3.23) and MySQL 4.0.13 (for MySQL 4.0).

- `--disable-keys, -K`

  For each table, surround the `INSERT` statements with `/*!40000 ALTER TABLE` *`tbl_name`* `DISABLE KEYS */;` and `/*!40000 ALTER TABLE` *`tbl_name`* `ENABLE KEYS */;` statements. This makes loading the dump file into a MySQL 4.0 server faster because the indexes are created after all rows are inserted. This option is effective only for `MyISAM` tables.

- `--extended-insert, -e`

  Use multiple-row `INSERT` syntax that include several `VALUES` lists. This results in a smaller dump file and speeds up inserts when the file is reloaded.

- `--fields-terminated-by=...`, `--fields-enclosed-by=...`, `--fields-optionally-enclosed-by=...`, `--fields-escaped-by=...`, `--lines-terminated-by=...`

  These options are used with the `-T` option and have the same meaning as the corresponding clauses

for `LOAD DATA INFILE`. See Section 13.1.5, "`LOAD DATA INFILE` Syntax".

- `--first-slave, -x`

  Deprecated, renamed to `--lock-all-tables` in MySQL 4.1.8.

- `--flush-logs, -F`

  Flush the MySQL server log files before starting the dump. This option requires the `RELOAD` privilege. Note that if you use this option in combination with the `--all-databases` (or `-A`) option, the logs are flushed *for each database dumped*. The exception is when using `--lock-all-tables` or `--master-data`: In this case, the logs are flushed only once, corresponding to the moment that all tables are locked. If you want your dump and the log flush to happen at exactly the same moment, you should use `--flush-logs` together with either `--lock-all-tables` or `--master-data`.

- `--force, -f`

  Continue even if an SQL error occurs during a table dump.

- `--host=`*`host_name`*`, -h `*`host_name`*

  Dump data from the MySQL server on the given host. The default host is `localhost`.

- `--hex-blob`

  Dump binary string columns using hexadecimal notation (for example, `'abc'` becomes `0x616263`). The affected columns are `BINARY`, `VARBINARY`, and `BLOB` in MySQL 4.1 and up, and `CHAR BINARY`, `VARCHAR BINARY`, and `BLOB` in MySQL 4.0. This option was added in MySQL 4.0.23 and 4.1.8.

- `--lock-all-tables, -x`

  Lock all tables across all databases. This is achieved by acquiring a global read lock for the duration of the whole dump. This option automatically turns off `--single-transaction` and `--lock-tables`. Added in MySQL 4.1.8.

- `--lock-tables, -l`

  Lock all tables before starting the dump. The tables are locked with `READ LOCAL` to allow concurrent inserts in the case of `MyISAM` tables. For `InnoDB` tables, `--single-transaction` is a much better option, because it does not need to lock the tables at all.

  Please note that when dumping multiple databases, `--lock-tables` locks tables for each database separately. So, this option does not guarantee that the tables in the dump file are logically consistent between databases. Tables in different databases may be dumped in completely different states.

- `--master-data[=`*`value`*`]`

  This option causes the binary log position and filename to be written to the output. This option requires the `RELOAD` privilege. If the option value is equal to 1, the position and filename are written to the dump output in the form of a `CHANGE MASTER` statement that makes a slave server start from the correct position in the master's binary logs if you use this SQL dump of the master to set up a slave. If the option value is equal to 2, the `CHANGE MASTER` statement is written as an SQL comment. This is the default action if *`value`* is omitted. *`value`* may be given as of MySQL 4.1.8; before that, do not specify an option value.

  The `--master-data` option turns on `--lock-all-tables`, unless `-`

`-single-transaction` also is specified (in which case, a global read lock is only acquired a short time at the beginning of the dump. See also the description for `--single-transaction`. In all cases, any action on logs happens at the exact moment of the dump. This option automatically turns off `--lock-tables`.

- `--no-create-db, -n`

  This option suppresses the `CREATE DATABASE /*!32312 IF NOT EXISTS*/ db_name` statements that are otherwise included in the output if the `--databases` or `--all-databases` option is given.

- `--no-create-info, -t`

  Don't write `CREATE TABLE` statements that re-create each dumped table.

- `--no-data, -d`

  Don't write any row information for the table. This is very useful if you just want to get a dump of the structure for a table.

- `--opt`

  This option is shorthand; it is the same as specifying `--add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset`. It should give you a fast dump operation and produce a dump file that can be reloaded into a MySQL server quickly. **As of MySQL 4.1, `--opt` is on by default, but can be disabled with `--skip-opt`**. To disable only certain of the options enabled by `--opt`, use their `--skip` forms; for example, `--skip-add-drop-table` or `--skip-quick`.

- `--password[=`*`password`*`], -p[`*`password`*`]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *`password`* value following the `--password` or `-p` option on the command line, you are prompted for one.

- `--port=`*`port_num`*`, -P `*`port_num`*

  The TCP/IP port number to use for the connection.

- `--protocol={TCP | SOCKET | PIPE | MEMORY}`

  The connection protocol to use. New in MySQL 4.1.

- `--quick, -q`

  This option is useful for dumping large tables. It forces **mysqldump** to retrieve rows for a table from the server a row at a time rather than retrieving the entire row set and buffering it in memory before writing it out.

- `--quote-names, -Q`

  Quote database, table, and column names within '`' characters. If the server SQL mode includes the `ANSI_QUOTES` option, names are quoted within '"' characters. As of MySQL 4.1.1, `--quote-names` is on by default, but can be disabled with `--skip-quote-names`.

- `--result-file=`*`file`*`, -r `*`file`*

  Direct output to a given file. This option should be used on Windows, because it prevents newline '`\n`' characters from being converted to '`\r\n`' carriage return/newline sequences.

- `--set-charset`

  Add `SET NAMES` *`default_character_set`* to the output. This option is enabled by default. To suppress the `SET NAMES` statement, use `--skip-set-charset`. This option was added in MySQL 4.1.2.

- `--single-transaction`

  This option issues a `BEGIN` SQL statement before dumping data from the server. It is useful only with `InnoDB` tables, because then it dumps the consistent state of the database at the time when `BEGIN` was issued without blocking any applications.

  When using this option, you should keep in mind that only `InnoDB` tables are dumped in a consistent state. For example, any `MyISAM` or `HEAP` tables dumped while using this option may still change state.

  The `--single-transaction` option was added in MySQL 4.0.2. This option is mutually exclusive with the `--lock-tables` option, because `LOCK TABLES` causes any pending transactions to be committed implicitly.

  To dump big tables, you should combine this option with `--quick`.

- `--socket=`*`path`*`, -S `*`path`*

  The socket file to use when connecting to `localhost` (which is the default host).

- `--skip-comments`

  See the description for the `--comments` option.

- `--tab=`*`path`*`, -T `*`path`*

  Produce tab-separated data files. For each dumped table, **mysqldump** creates a *`tbl_name`*`.sql` file that contains the `CREATE TABLE` statement that creates the table, and a *`tbl_name`*`.txt` file that contains its data. The option value is the directory in which to write the files.

  By default, the `.txt` data files are formatted using tab characters between column values and a newline at the end of each line. The format can be specified explicitly using the `--fields-`*`xxx`* and `--lines--`*`xxx`* options.

  Note: This option should be used only when **mysqldump** is run on the same machine as the **mysqld** server. You must have the `FILE` privilege, and the server must have permission to write files in the directory that you specify.

- `--tables`

  Override the `--databases` or `-B` option. All arguments following the option are regarded as table names.

- `--user=`*`user_name`*`, -u `*`user_name`*

  The MySQL username to use when connecting to the server.

- `--verbose, -v`

  Verbose mode. Print out more information on what the program does.

- `--version, -V`

Display version information and exit.

- `--where='where-condition', -w 'where-condition'`

  Dump only records selected by the given `WHERE` condition. Note that quotes around the condition are mandatory if it contains spaces or characters that are special to your command interpreter.

  Examples:

  ```
  "--where=user='jimf'"
  "-wuserid>1"
  "-wuserid<1"
  ```

- `--xml, -X`

  Write dump output as well-formed XML.

You can also set the following variables by using `--var_name=value` options:

- `max_allowed_packet`

  The maximum size of the buffer for client/server communication. The value of the variable can be up to 16MB before MySQL 4.0, and up to 1GB from MySQL 4.0 on.

- `net_buffer_length`

  The initial size of the buffer for client/server communication. When creating multiple-row-insert statements (as with option `--extended-insert` or `--opt`), **mysqldump** creates rows up to `net_buffer_length` length. If you increase this variable, you should also ensure that the `net_buffer_length` variable in the MySQL server is at least this large.

It is also possible to set variables by using `--set-variable=var_name=value` or `-O var_name=value` syntax. However, this syntax is deprecated as of MySQL 4.0.

The most common use of **mysqldump** is probably for making a backup of an entire database:

```
shell> mysqldump --opt db_name > backup-file.sql
```

You can read the dump file back into the server like this:

```
shell> mysql db_name < backup-file.sql
```

Or like this:

```
shell> mysql -e "source /path-to-backup/backup-file.sql" db_name
```

mysqldump is also very useful for populating databases by copying data from one MySQL server to another:

```
shell> mysqldump --opt db_name | mysql --host=remote_host -C db_name
```

It is possible to dump several databases with one command:

```
shell> mysqldump --databases db_name1 [db_name2 ...] > my_databases.sql
```

If you want to dump all databases, use the `--all-databases` option:

```
shell> mysqldump --all-databases > all_databases.sql
```

If tables are stored in the `InnoDB` storage engine, `mysqldump` provides a way of making an online backup of these (see command below). This backup just needs to acquire a global read lock on all tables (using `FLUSH TABLES WITH READ LOCK`) at the beginning of the dump. As soon as this lock has been acquired, the binary log coordinates are read and lock is released. So if and only if one long updating statement is running when the `FLUSH...` is issued, the MySQL server may get stalled until that long statement finishes, and then the dump becomes lock-free. So if the MySQL server receives only short (in the sense of "short execution time") updating statements, even if there are plenty of them, the initial lock period should not be noticeable.

```
shell> mysqldump --all-databases --single-transaction > all_databases.sql
```

For point-in-time recovery (also known as "roll-forward", when you need to restore an old backup and replay the changes which happened since that backup), it is often useful to rotate the binary log (see Section 5.9.4, “The Binary Log”) or at least know the binary log coordinates to which the dump corresponds:

```
shell> mysqldump --all-databases --master-data=2 > all_databases.sql
or
shell> mysqldump --all-databases --flush-logs --master-data=2 > all_databases.sql
```

The simultaneous use of `--master-data` and `--single-transaction` works as of MySQL 4.1.8. It provides a convenient way to make an online backup suitable for point-in-time recovery, if tables are stored in the `InnoDB` storage engine.

For more information on making backups, see Section 5.7.1, “Database Backups”.

8.9. The mysqlhotcopy Database Backup Program

mysqlhotcopy is a Perl script that was originally written and contributed by Tim Bunce. It uses `LOCK TABLES`, `FLUSH TABLES`, and `cp` or `scp` to quickly make a backup of a database. It's the fastest way to make a backup of the database or single tables, but it can be run only on the same machine where the database directories are located. **mysqlhotcopy** works only for backing up `MyISAM` and `ISAM` tables. It runs on Unix, and as of MySQL 4.0.18 also on NetWare.

```
shell> mysqlhotcopy db_name [/path/to/new_directory]
```

```
shell> mysqlhotcopy db_name_1 ... db_name_n /path/to/new_directory
```

Back up tables in the given database that match a regular expression:

```
shell> mysqlhotcopy db_name./regex/
```

The regular expression for the table name can be negated by prefixing it with a tilde ('~'):

```
shell> mysqlhotcopy db_name./~regex/
```

mysqlhotcopy supports the following options:

- `--help, -?`

  Display a help message and exit.

- `--allowold`

  Don't abort if target exists (rename it by adding an `_old` suffix).

- `--checkpoint=db_name.tbl_name`

  Insert checkpoint entries into the specified *`db_name.tbl_name`*.

- `--debug`

  Enable debug output.

- `--dryrun, -n`

  Report actions without doing them.

- `--flushlog`

  Flush logs after all tables are locked.

- `--keepold`

  Don't delete previous (renamed) target when done.

- `--method=#`

  Method for copy (`cp` or `scp`).

- `--noindices`

  Don't include full index files in the backup. This makes the backup smaller and faster. The indexes can be reconstructed later with **myisamchk -rq** for `MyISAM` tables or **isamchk -rq** for `ISAM` tables.

- `--password=password, -ppassword`

  The password to use when connecting to the server. Note that the password value is not optional for this option, unlike for other MySQL programs.

- `--port=port_num, -P port_num`

  The TCP/IP port number to use when connecting to the local server.

- `--quiet, -q`

  Be silent except for errors.

- `--regexp=expr`

Copy all databases with names matching the given regular expression.

- `--socket=path, -S path`

  The Unix socket file to use for the connection.

- `--suffix=str`

  The suffix for names of copied databases.

- `--tmpdir=path`

  The temporary directory (instead of `/tmp`).

- `--user=user_name, -u user_name`

  The MySQL username to use when connecting to the server.

mysqlhotcopy reads the `[client]` and `[mysqlhotcopy]` option groups from option files.

To execute **mysqlhotcopy**, you must have access to the files for the tables that you are backing up, the `SELECT` privilege for those tables, and the `RELOAD` privilege (to be able to execute `FLUSH TABLES`).

Use `perldoc` for additional **mysqlhotcopy** documentation:

```
shell> perldoc mysqlhotcopy
```

8.10. The mysqlimport Data Import Program

The **mysqlimport** client provides a command-line interface to the `LOAD DATA INFILE` SQL statement. Most options to **mysqlimport** correspond directly to clauses of `LOAD DATA INFILE`. See Section 13.1.5, "`LOAD DATA INFILE` Syntax".

Invoke **mysqlimport** like this:

```
shell> mysqlimport [options] db_name textfile1 [textfile2 ...]
```

For each text file named on the command line, **mysqlimport** strips any extension from the filename and uses the result to determine the name of the table into which to import the file's contents. For example, files named `patient.txt`, `patient.text`, and `patient` all would be imported into a table named `patient`.

mysqlimport supports the following options:

- `--help, -?`

  Display a help message and exit.

- `--columns=column_list, -c column_list`

  This option takes a comma-separated list of column names as its value. The order of the column names indicates how to match up data file columns with table columns.

- `--compress, -C`

Compress all information sent between the client and the server if both support compression.

- `--debug[=debug_options], -# [debug_options]`

  Write a debugging log. The *debug_options* string often is `'d:t:o,file_name'`.

- `--delete, -D`

  Empty the table before importing the text file.

- `--fields-terminated-by=...`, `--fields-enclosed-by=...`, `--fields-optionally-enclosed-by=...`, `--fields-escaped-by=...`, `--lines-terminated-by=...`

  These options have the same meaning as the corresponding clauses for `LOAD DATA INFILE`. See Section 13.1.5, "`LOAD DATA INFILE` Syntax".

- `--force, -f`

  Ignore errors. For example, if a table for a text file doesn't exist, continue processing any remaining files. Without `--force`, **mysqlimport** exits if a table doesn't exist.

- `--host=host_name, -h host_name`

  Import data to the MySQL server on the given host. The default host is `localhost`.

- `--ignore, -i`

  See the description for the `--replace` option.

- `--ignore-lines=n`

  Ignore the first *n* lines of the data file.

- `--local, -L`

  Read input files locally from the client host.

- `--lock-tables, -l`

  Lock *all* tables for writing before processing any text files. This ensures that all tables are synchronized on the server.

- `--password[=password], -p[password]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *password* value following the `--password` or `-p` option on the command line, you are prompted for one.

- `--port=port_num, -P port_num`

  The TCP/IP port number to use for the connection.

- `--protocol={TCP | SOCKET | PIPE | MEMORY}`

  The connection protocol to use. New in MySQL 4.1.

- `--replace, -r`

The `--replace` and `--ignore` options control handling of input records that duplicate existing records on unique key values. If you specify `--replace`, new rows replace existing rows that have the same unique key value. If you specify `--ignore`, input rows that duplicate an existing row on a unique key value are skipped. If you don't specify either option, an error occurs when a duplicate key value is found, and the rest of the text file is ignored.

- `--silent, -s`

  Silent mode. Produce output only when errors occur.

- `--socket=path, -S path`

  The socket file to use when connecting to `localhost` (which is the default host).

- `--user=user_name, -u user_name`

  The MySQL username to use when connecting to the server.

- `--verbose, -v`

  Verbose mode. Print out more information what the program does.

- `--version, -V`

  Display version information and exit.

Here is a sample session that demonstrates use of **mysqlimport**:

```
shell> mysql -e 'CREATE TABLE imptest(id INT, n VARCHAR(30))' test
shell> ed
a
100     Max Sydow
101     Count Dracula
.
w imptest.txt
32
q
shell> od -c imptest.txt
0000000   1   0   0  \t   M   a   x       S   y   d   o   w  \n   1   0
0000020   1  \t   C   o   u   n   t       D   r   a   c   u   l   a  \n
0000040
shell> mysqlimport --local test imptest.txt
test.imptest: Records: 2  Deleted: 0  Skipped: 0  Warnings: 0
shell> mysql -e 'SELECT * FROM imptest' test
+------+---------------+
| id   | n             |
+------+---------------+
|  100 | Max Sydow     |
|  101 | Count Dracula |
+------+---------------+
```

8.11. mysqlshow, Showing Databases, Tables, and Columns

The **mysqlshow** client can be used to quickly look at which databases exist, their tables, and a table's columns or indexes.

mysqlshow provides a command-line interface to several SQL `SHOW` statements. The same information can be obtained by using those statements directly. For example, you can issue them from the **mysql** client program. See Section 13.5.4, “`SHOW` Syntax”.

Invoke **mysqlshow** like this:

```
shell> mysqlshow [options] [db_name [tbl_name [col_name]]]
```

- If no database is given, all matching databases are shown.
- If no table is given, all matching tables in the database are shown.
- If no column is given, all matching columns and column types in the table are shown.

Note that in newer MySQL versions, you see only those database, tables, or columns for which you have some privileges.

If the last argument contains shell or SQL wildcard characters ('`*`', '`?`', '`%`', or '`_`'), only those names that are matched by the wildcard are shown. If a database name contains any underscores, those should be escaped with a backslash (some Unix shells require two) in order to get a list of the proper tables or columns. '`*`' and '`?`' characters are converted into SQL '`%`' and '`_`' wildcard characters. This might cause some confusion when you try to display the columns for a table with a '`_`' in the name, because in this case **mysqlshow** shows you only the table names that match the pattern. This is easily fixed by adding an extra '`%`' last on the command line as a separate argument.

mysqlshow supports the following options:

- `--help, -?`

  Display a help message and exit.

- `--character-sets-dir=path`

  The directory where character sets are installed. See Section 5.8.1, “The Character Set Used for Data and Sorting”.

- `--compress, -C`

  Compress all information sent between the client and the server if both support compression.

- `--debug[=debug_options], -# [debug_options]`

  Write a debugging log. The *`debug_options`* string often is `'d:t:o,file_name'`.

- `--default-character-set=charset`

  Use *`charset`* as the default character set. See Section 5.8.1, “The Character Set Used for Data and Sorting”.

- `--host=host_name, -h host_name`

  Connect to the MySQL server on the given host.

- `--keys, -k`

  Show table indexes.

- `--password[=password], -p[password]`

  The password to use when connecting to the server. If you use the short option form (`-p`), you *cannot* have a space between the option and the password. If you omit the *`password`* value following the `--password` or `-p` option on the command line, you are prompted for one.

- `--port=port_num, -P port_num`

  The TCP/IP port number to use for the connection.

- `--protocol={TCP | SOCKET | PIPE | MEMORY}`

  The connection protocol to use. New in MySQL 4.1.

- `--socket=path, -S path`

  The socket file to use when connecting to `localhost` (which is the default host).

- `--status, -i`

  Display extra information about each table.

- `--user=user_name, -u user_name`

  The MySQL username to use when connecting to the server.

- `--verbose, -v`

  Verbose mode. Print out more information what the program does. This option can be used multiple times to increase the amount of information.

- `--version, -V`

  Display version information and exit.

8.12. perror, Explaining Error Codes

For most system errors, MySQL displays, in addition to an internal text message, the system error code in one of the following styles:

```
message ... (errno: #)
message ... (Errcode: #)
```

You can find out what the error code means by either examining the documentation for your system or by using the **perror** utility.

perror prints a description for a system error code or for a storage engine (table handler) error code.

Invoke **perror** like this:

```
shell> perror [options] errorcode ...
```

Example:

```
shell> perror 13 64
Error code  13:  Permission denied
```

```
Error code  64:  Machine is not on the network
```

Note that the meaning of system error messages may be dependent on your operating system. A given error code may mean different things on different operating systems.

8.13. The replace String-Replacement Utility

The **replace** utility program changes strings in place in files or on the standard input. It uses a finite state machine to match longer strings first. It can be used to swap strings. For example, the following command swaps `a` and `b` in the given files, `file1` and `file2`:

```
shell> replace a b b a -- file1 file2 ...
```

Use the `--` option to indicate where the string-replacement list ends and the filenames begin.

Any file named on the command line is modified in place, so you may want to make a copy of the original before converting it.

If no files are named on the command line, **replace** reads the standard input and writes to the standard output. In this case, no `--` option is needed.

The **replace** program is used by **msql2mysql**. See Section 22.1.1, "**msql2mysql**, Convert mSQL Programs for Use with MySQL".

replace supports the following options:

- `-?, -I`

  Display a help message and exit.

- `-# debug_options`

  Write a debugging log. The *`debug_options`* string often is `'d:t:o,file_name'`.

- `-s`

  Silent mode. Print out less information what the program does.

- `-v`

  Verbose mode. Print out more information what the program does.

- `-V`

  Display version information and exit.

Chapter 9. Language Structure

This chapter discusses the rules for writing the following elements of SQL statements when using MySQL:

- Literal values such as strings and numbers
- Identifiers such as table and column names
- User and system variables
- Comments
- Reserved words

9.1. Literal Values

This section describes how to write literal values in MySQL. These include strings, numbers, hexadecimal values, boolean values, and `NULL`. The section also covers the various nuances and ``gotchas'' that you may run into when dealing with these basic types in MySQL.

9.1.1. Strings

A string is a sequence of characters, surrounded by either single quote (`''`) or double quote (`'"'`) characters. Examples:

```
'a string'
"another string"
```

If the server SQL mode has `ANSI_QUOTES` enabled, string literals can be quoted only with single quotes. A string quoted with double quotes is interpreted as an identifier.

As of MySQL 4.1.1, string literals may have an optional character set introducer and `COLLATE` clause:

```
[_charset_name]'string' [COLLATE collation_name]
```

Examples:

```
SELECT _latin1'string';
SELECT _latin1'string' COLLATE latin1_danish_ci;
```

For more information about these forms of string syntax, see Section 10.3.7, “Character String Literal Character Set and Collation”.

Within a string, certain sequences have special meaning. Each of these sequences begins with a backslash (`'\'`), known as the *escape character*. MySQL recognizes the following escape sequences:

`\0`	An ASCII 0 (`NUL`) character.
`\'`	A single quote (`''`) character.
`\"`	A double quote (`'"'`) character.
`\b`	A backspace character.

`\n`	A newline (linefeed) character.
`\r`	A carriage return character.
`\t`	A tab character.
`\Z`	ASCII 26 (Control-Z). This character can be encoded as '`\Z`' to allow you to work around the problem that ASCII 26 stands for END-OF-FILE on Windows. (ASCII 26 causes problems if you try to use `mysql db_name < file_name`.)
`\\`	A backslash ('`\`') character.
`\%`	A '`%`' character. See note following table.
`\_`	A '`_`' character. See note following table.

These sequences are case sensitive. For example, '`\b`' is interpreted as a backspace, but '`\B`' is interpreted as '`B`'.

The '`\%`' and '`\_`' sequences are used to search for literal instances of '`%`' and '`_`' in pattern-matching contexts where they would otherwise be interpreted as wildcard characters. See Section 12.3.1, “String Comparison Functions”. Note that if you use '`\%`' or '`\_`' in other contexts, they return the strings '`\%`' and '`\_`' and not '`%`' and '`_`'.

In all other escape sequences, backslash is ignored. That is, the escaped character is interpreted as if it was not escaped.

There are several ways to include quotes within a string:

- A '`'`' inside a string quoted with '`'`' may be written as '`''`'.
- A '`"`' inside a string quoted with '`"`' may be written as '`""`'.
- You can precede the quote character with an escape character ('`\`').
- A '`'`' inside a string quoted with '`"`' needs no special treatment and need not be doubled or escaped. In the same way, '`"`' inside a string quoted with '`'`' needs no special treatment.

The following `SELECT` statements demonstrate how quoting and escaping work:

```
mysql> SELECT 'hello', '"hello"', '""hello""', 'hel''lo', '\'hello';
+-------+---------+-----------+--------+--------+
| hello | "hello" | ""hello"" | hel'lo | 'hello |
+-------+---------+-----------+--------+--------+

mysql> SELECT "hello", "'hello'", "''hello''", "hel""lo", "\"hello";
+-------+---------+-----------+--------+--------+
| hello | 'hello' | ''hello'' | hel"lo | "hello |
+-------+---------+-----------+--------+--------+

mysql> SELECT 'This\nIs\nFour\nLines';
+--------------------+
| This
Is
Four
Lines |
+--------------------+

mysql> SELECT 'disappearing\ backslash';
+------------------------+
| disappearing backslash |
```

```
+------------------------+
```

If you want to insert binary data into a string column (such as a `BLOB`), the following characters must be represented by escape sequences:

`NUL`	`NUL` byte (ASCII 0). Represent this character by '`\0`' (a backslash followed by an ASCII '`0`' character).
`\`	Backslash (ASCII 92). Represent this character by '`\\`'.
`'`	Single quote (ASCII 39). Represent this character by '`\'`'.
`"`	Double quote (ASCII 34). Represent this character by '`\"`'.

When writing application programs, any string that might contain any of these special characters must be properly escaped before the string is used as a data value in an SQL statement that is sent to the MySQL server. You can do this in two ways:

- Process the string with a function that escapes the special characters. For example, in a C program, you can use the `mysql_real_escape_string()` C API function to escape characters. See Section 22.2.3.47, "`mysql_real_escape_string()`". The Perl DBI interface provides a `quote` method to convert special characters to the proper escape sequences. See Section 22.4, "MySQL Perl API".
- As an alternative to explicitly escaping special characters, many MySQL APIs provide a placeholder capability that allows you to insert special markers into a query string, and then bind data values to them when you issue the query. In this case, the API takes care of escaping special characters in the values for you.

9.1.2. Numbers

Integers are represented as a sequence of digits. Floats use '`.`' as a decimal separator. Either type of number may be preceded by '`-`' to indicate a negative value.

Examples of valid integers:

```
1221
0
-32
```

Examples of valid floating-point numbers:

```
294.42
-32032.6809e+10
148.00
```

An integer may be used in a floating-point context; it is interpreted as the equivalent floating-point number.

9.1.3. Hexadecimal Values

MySQL supports hexadecimal values. In numeric contexts, these act like integers (64-bit precision). In string contexts, these act like binary strings, where each pair of hex digits is converted to a character:

```
mysql> SELECT x'4D7953514C';
        -> 'MySQL'
mysql> SELECT 0xa+0;
        -> 10
mysql> SELECT 0x5061756c;
        -> 'Paul'
```

In MySQL 4.1 (and in MySQL 4.0 when using the `--new` option), the default type of a hexadecimal value is a string. If you want to ensure that the value is treated as a number, you can use `CAST(... AS UNSIGNED)`:

```
mysql> SELECT 0x41, CAST(0x41 AS UNSIGNED);
        -> 'A', 65
```

The `0x` syntax is based on ODBC. Hexadecimal strings are often used by ODBC to supply values for `BLOB` columns. The `x'hexstring'` syntax is new in 4.0 and is based on standard SQL.

Beginning with MySQL 4.0.1, you can convert a string or a number to a string in hexadecimal format with the `HEX()` function:

```
mysql> SELECT HEX('cat');
        -> '636174'
mysql> SELECT 0x636174;
        -> 'cat'
```

9.1.4. Boolean Values

Beginning with MySQL 4.1, the constant `TRUE` evaluates to `1` and the constant `FALSE` evaluates to `0`. The constant names can be written in any lettercase.

```
mysql> SELECT TRUE, true, FALSE, false;
        -> 1, 1, 0, 0
```

9.1.5. Bit-Field Values

Beginning with MySQL 5.0.3, bit-field values can be written using `b'value'` notation. *`value`* is a binary value written using 0s and 1s.

Bit-field notation is convenient for specifying values to be assigned to `BIT` columns:

```
mysql> CREATE TABLE t (b BIT(8));
mysql> INSERT INTO t SET b = b'11111111';
mysql> INSERT INTO t SET b = b'1010';
+------+----------+----------+----------+
| b+0  | BIN(b+0) | OCT(b+0) | HEX(b+0) |
+------+----------+----------+----------+
|  255 | 11111111 | 377      | FF       |
|   10 | 1010     | 12       | A        |
+------+----------+----------+----------+
```

9.1.6. `NULL` Values

The `NULL` value means ``no data.'' `NULL` can be written in any lettercase.

Be aware that the NULL value is different from values such as 0 for numeric types or the empty string for string types. See Section A.5.3, “Problems with NULL Values”.

For text file import or export operations performed with LOAD DATA INFILE or SELECT ... INTO OUTFILE, NULL is represented by the \N sequence. See Section 13.1.5, “LOAD DATA INFILE Syntax”.

9.2. Database, Table, Index, Column, and Alias Names

Database, table, index, column, and alias names are identifiers. This section describes the allowable syntax for identifiers in MySQL.

The following table describes the maximum length and allowable characters for each type of identifier.

Identifier	Maximum Length (bytes)	Allowed Characters
Database	64	Any character that is allowed in a directory name, except '/', '\', or '.'
Table	64	Any character that is allowed in a filename, except '/', '\', or '.'
Column	64	All characters
Index	64	All characters
Alias	255	All characters

In addition to the restrictions noted in the table, no identifier can contain ASCII 0 or a byte with a value of 255. Database, table, and column names should not end with space characters. Before MySQL 4.1, identifier quote characters should not be used in identifiers.

Beginning with MySQL 4.1, identifiers are stored using Unicode (UTF8). This applies to identifiers in table definitions that stored in .frm files and to identifiers stored in the grant tables in the mysql database. Although Unicode identifiers can include multi-byte characters, note that the maximum lengths shown in the table are byte counts. If an identifier does contain multi-byte characters, the number of *characters* allowed in the identifier is less than the value shown in the table.

An identifier may be quoted or unquoted. If an identifier is a reserved word or contains special characters, you *must* quote it whenever you refer to it. For a list of reserved words, see Section 9.6, “Treatment of Reserved Words in MySQL”. Special characters are those outside the set of alphanumeric characters from the current character set, '_', and '$'.

The identifier quote character is the backtick ('`'):

```
mysql> SELECT * FROM `select` WHERE `select`.id > 100;
```

If the server SQL mode includes the ANSI_QUOTES mode option, it is also allowable to quote identifiers with double quotes:

```
mysql> CREATE TABLE "test" (col INT);
ERROR 1064: You have an error in your SQL syntax. (...)
mysql> SET sql_mode='ANSI_QUOTES';
mysql> CREATE TABLE "test" (col INT);
Query OK, 0 rows affected (0.00 sec)
```

See Section 5.2.2, “The Server SQL Mode”.

As of MySQL 4.1, identifier quote characters can be included within an identifier if you quote the identifier. If the character to be included within the identifier is the same as that used to quote the identifier itself, double the character. The following statement creates a table named `a`b` that contains a column named `c"d`:

```
mysql> CREATE TABLE `a``b` (`c"d` INT);
```

Identifier quoting was introduced in MySQL 3.23.6 to allow use of identifiers that are reserved words or that contain special characters. Before 3.23.6, you cannot use identifiers that require quotes, so the rules for legal identifiers are more restrictive:

- A name may consist of alphanumeric characters from the current character set, '_', and '$'. The default character set is ISO-8859-1 (Latin1). This may be changed with the `--default-character-set` option to **mysqld**. See Section 5.8.1, “The Character Set Used for Data and Sorting”.
- A name may start with any character that is legal in a name. In particular, a name may start with a digit; this differs from many other database systems! However, an unquoted name cannot consist of *only* digits.
- You cannot use the '`.`' character in names because it is used to extend the format by which you can refer to columns (see Section 9.2.1, “Identifier Qualifiers”).

It is recommended that you do not use names like `1e`, because an expression like `1e+1` is ambiguous. It might be interpreted as the expression `1e + 1` or as the number `1e+1`, depending on context.

9.2.1. Identifier Qualifiers

MySQL allows names that consist of a single identifier or multiple identifiers. The components of a multiple-part name should be separated by period ('`.`') characters. The initial parts of a multiple-part name act as qualifiers that affect the context within which the final identifier is interpreted.

In MySQL you can refer to a column using any of the following forms:

Column Reference	Meaning
`col_name`	The column *`col_name`* from whichever table used in the query contains a column of that name.
`tbl_name.col_name`	The column *`col_name`* from table *`tbl_name`* of the default database.
`db_name.tbl_name.col_name`	The column *`col_name`* from table *`tbl_name`* of the database *`db_name`*. This syntax is unavailable before MySQL 3.22.

If any components of a multiple-part name require quoting, quote them individually rather than quoting the name as a whole. For example, `` `my-table`.`my-column` `` is legal, whereas `` `my-table.my-column` `` is not.

You need not specify a *`tbl_name`* or *`db_name.tbl_name`* prefix for a column reference in a statement unless the reference would be ambiguous. Suppose that tables `t1` and `t2` each contain a column `c`, and you retrieve `c` in a `SELECT` statement that uses both `t1` and `t2`. In this case, `c` is ambiguous because it is not unique among the tables used in the statement. You must qualify it with a table name as `t1.c` or `t2.c` to indicate which table you mean. Similarly, to retrieve from a table `t` in database `db1` and from a table `t` in database `db2` in the same statement, you must refer to columns in those tables as

`db1.t.col_name` and `db2.t.col_name`.

The syntax `.tbl_name` means the table `tbl_name` in the current database. This syntax is accepted for ODBC compatibility because some ODBC programs prefix table names with a `'.'` character.

9.2.2. Identifier Case Sensitivity

In MySQL, databases correspond to directories within the data directory. Tables within a database correspond to at least one file within the database directory (and possibly more, depending on the storage engine). Consequently, the case sensitivity of the underlying operating system determines the case sensitivity of database and table names. This means database and table names are not case sensitive in Windows, and case sensitive in most varieties of Unix. One notable exception is Mac OS X, which is Unix-based but uses a default filesystem type (HFS+) that is not case sensitive. However, Mac OS X also supports UFS volumes, which are case sensitive just as on any Unix. See Section 1.5.4, “MySQL Extensions to Standard SQL”.

Note: Although database and table names are not case sensitive on some platforms, you should not refer to a given database or table using different cases within the same query. The following query would not work because it refers to a table both as `my_table` and as `MY_TABLE`:

```
mysql> SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

Column names, index names, and column aliases are not case sensitive on any platform.

Table aliases are case sensitive before MySQL 4.1.1. The following query would not work because it refers to the alias both as `a` and as `A`:

```
mysql> SELECT col_name FROM tbl_name AS a
    -> WHERE a.col_name = 1 OR A.col_name = 2;
```

If you have trouble remembering the allowable lettercase for database and table names, adopt a consistent convention, such as always creating databases and tables using lowercase names.

How table and database names are stored on disk and used in MySQL is defined by the `lower_case_table_names` system variable, which you can set when starting **mysqld**. `lower_case_table_names` can take one of the following values:

Value	Meaning
`0`	Table and database names are stored on disk using the lettercase specified in the `CREATE TABLE` or `CREATE DATABASE` statement. Name comparisons are case sensitive. This is the default on Unix systems. Note that if you force this to 0 with `--lower-case-table-names=0` on a case-insensitive filesystem and access `MyISAM` tablenames using different lettercases, this may lead to index corruption.
`1`	Table names are stored in lowercase on disk and name comparisons are not case sensitive. MySQL converts all table names to lowercase on storage and lookup. This behavior also applies to database names as of MySQL 4.0.2, and to table aliases as of 4.1.1. This value is the default on Windows and Mac OS X systems.
`2`	Table and database names are stored on disk using the lettercase specified in the `CREATE TABLE` or `CREATE DATABASE` statement, but MySQL converts them to lowercase on lookup. Name comparisons are not case sensitive. **Note**: This works *only* on filesystems that are not case sensitive! `InnoDB` table names are stored in lowercase, as for `lower_case_table_names=1`. Setting `lower_case_table_names` to `2` can be done as of MySQL 4.0.18.

If you are using MySQL on only one platform, you don't normally have to change the

lower_case_table_names variable. However, you may encounter difficulties if you want to transfer tables between platforms that differ in filesystem case sensitivity. For example, on Unix, you can have two different tables named my_table and MY_TABLE, but on Windows those names are considered the same. To avoid data transfer problems stemming from database or table name lettercase, you have two options:

- Use lower_case_table_names=1 on all systems. The main disadvantage with this is that when you use SHOW TABLES or SHOW DATABASES, you don't see the names in their original lettercase.
- Use lower_case_table_names=0 on Unix and lower_case_table_names=2 on Windows. This preserves the lettercase of database and table names. The disadvantage of this is that you must ensure that your queries always refer to your database and table names with the correct lettercase on Windows. If you transfer your queries to Unix, where lettercase is significant, they do not work if the lettercase is incorrect.

Note that before setting lower_case_table_names to 1 on Unix, you must first convert your old database and table names to lowercase before restarting **mysqld**.

9.3. User Variables

MySQL supports user variables as of version 3.23.6. You can store a value in a user variable and refer to it later, which allows you to pass values from one statement to another. User variables are connection-specific. That is, a variable defined by one client cannot be seen or used by other clients. All variables for a client connection are automatically freed when the client exits.

User variables are written as *@var_name*, where the variable name *var_name* may consist of alphanumeric characters from the current character set, '.', '_', and '$'. The default character set is ISO-8859-1 (Latin1). This may be changed with the --default-character-set option to **mysqld**. See Section 5.8.1, “The Character Set Used for Data and Sorting”. User variable names are not case sensitive beginning with MySQL 5.0. Before that, they are case sensitive.

One way to set a user variable is by issuing a SET statement:

```
SET @var_name = expr [, @var_name = expr] ...
```

For SET, either = or := can be used as the assignment operator. The *expr* assigned to each variable can evaluate to an integer, real, string, or NULL value.

You can also assign a value to a user variable in statements other than SET. In this case, the assignment operator must be := and not = because = is treated as a comparison operator in non-SET statements:

```
mysql> SET @t1=0, @t2=0, @t3=0;
mysql> SELECT @t1:=(@t2:=1)+@t3:=4,@t1,@t2,@t3;
+----------------------+------+------+------+
| @t1:=(@t2:=1)+@t3:=4 | @t1  | @t2  | @t3  |
+----------------------+------+------+------+
|                    5 |    5 |    1 |    4 |
+----------------------+------+------+------+
```

User variables may be used where expressions are allowed. This does not currently include contexts that explicitly require a number, such as in the LIMIT clause of a SELECT statement, or the IGNORE number LINES clause of a LOAD DATA statement.

If you refer to a variable that has not been initialized, its value is NULL.

Beginning with MySQL 4.1.1, if a user variable is assigned a string value, it also has the same character set and collation as the string. The coercibility of user variables is ``implicit'' as of MySQL 4.1.11 and 5.0.3. (This is the same coercibility as table column values.)

Note: In a `SELECT` statement, each expression is evaluated only when sent to the client. This means that in a `HAVING`, `GROUP BY`, or `ORDER BY` clause, you cannot refer to an expression that involves variables that are set in the `SELECT` list. For example, the following statement does *not* work as expected:

```
mysql> SELECT (@aa:=id) AS a, (@aa+3) AS b FROM tbl_name HAVING b=5;
```

The reference to `b` in the `HAVING` clause refers to an alias for an expression in the `SELECT` list that uses `@aa`. This does not work as expected: `@aa` does not contain the value of the current row, but the value of `id` from the previous selected row.

The general rule is to never assign *and* use the same variable in the same statement.

Another issue with setting a variable and using it in the same statement is that the default result type of a variable is based on the type of the variable at the start of the statement. The following example illustrates this:

```
mysql> SET @a='test';
mysql> SELECT @a,(@a:=20) FROM tbl_name;
```

For this `SELECT` statement, MySQL reports to the client that column one is a string and converts all accesses of `@a` to strings, even though @a is set to a number for the second row. After the `SELECT` statement executes, `@a` is regarded as a number for the next statement.

To avoid problems with this behavior, either do not set and use the same variable within a single statement, or else set the variable to `0`, `0.0`, or `''` to define its type before you use it.

An unassigned variable has a value of `NULL` with a type of string.

9.4. System Variables

Starting from MySQL 4.0.3, we provide better access to a lot of system and connection variables. Many variables can be changed dynamically while the server is running. This allows you to modify server operation without having to stop and restart it.

The **mysqld** server maintains two kinds of variables. Global variables affect the overall operation of the server. Session variables affect its operation for individual client connections.

When the server starts, it initializes all global variables to their default values. These defaults may be changed by options specified in option files or on the command line. After the server starts, those global variables that are dynamic can be changed by connecting to the server and issuing a `SET GLOBAL` *`var_name`* statement. To change a global variable, you must have the `SUPER` privilege.

The server also maintains a set of session variables for each client that connects. The client's session variables are initialized at connect time using the current values of the corresponding global variables. For those session variables that are dynamic, the client can change them by issuing a `SET SESSION` *`var_name`* statement. Setting a session variable requires no special privilege, but a client can change only its own session variables, not those of any other client.

A change to a global variable is visible to any client that accesses that global variable. However, it affects the corresponding session variable that is initialized from the global variable only for clients that connect after the change. It does not affect the session variable for any client that is currently connected (not even that of the client that issues the `SET GLOBAL` statement).

Global or session variables may be set or retrieved using several syntax forms. The following examples use `sort_buffer_size` as a sample variable name.

To set the value of a `GLOBAL` variable, use one of the following syntaxes:

```
mysql> SET GLOBAL sort_buffer_size=value;
mysql> SET @@global.sort_buffer_size=value;
```

To set the value of a `SESSION` variable, use one of the following syntaxes:

```
mysql> SET SESSION sort_buffer_size=value;
mysql> SET @@session.sort_buffer_size=value;
mysql> SET sort_buffer_size=value;
```

`LOCAL` is a synonym for `SESSION`.

If you don't specify `GLOBAL`, `SESSION`, or `LOCAL` when setting a variable, `SESSION` is the default. See Section 13.5.3, "`SET` Syntax".

To retrieve the value of a `GLOBAL` variable, use one of the following statements:

```
mysql> SELECT @@global.sort_buffer_size;
mysql> SHOW GLOBAL VARIABLES like 'sort_buffer_size';
```

To retrieve the value of a `SESSION` variable, use one of the following statements:

```
mysql> SELECT @@sort_buffer_size;
mysql> SELECT @@session.sort_buffer_size;
mysql> SHOW SESSION VARIABLES like 'sort_buffer_size';
```

Here, too, `LOCAL` is a synonym for `SESSION`.

When you retrieve a variable with `SELECT @@var_name` (that is, you do not specify `global.`, `session.`, or `local.`, MySQL returns the `SESSION` value if it exists and the `GLOBAL` value otherwise.

For `SHOW VARIABLES`, if you do not specify `GLOBAL`, `SESSION`, or `LOCAL`, MySQL returns the `SESSION` value.

The reason for requiring the `GLOBAL` keyword when setting `GLOBAL`-only variables but not when retrieving them is to prevent problems in the future. If we remove a `SESSION` variable with the same name as a `GLOBAL` variable, a client with the `SUPER` privilege might accidentally change the `GLOBAL` variable rather than just the `SESSION` variable for its own connection. If we add a `SESSION` variable with the same name as a `GLOBAL` variable, a client that intends to change the `GLOBAL` variable might find only its own `SESSION` variable changed.

Further information about system startup options and system variables can be found in Section 5.2.1, "**mysqld** Command-Line Options" and Section 5.2.3, "Server System Variables". A list of the variables that can be set at runtime is given in Section 5.2.3.1, "Dynamic System Variables".

9.4.1. Structured System Variables

Structured system variables are supported beginning with MySQL 4.1.1. A structured variable differs from a regular system variable in two respects:

- Its value is a structure with components that specify server parameters considered to be closely related.
- There might be several instances of a given type of structured variable. Each one has a different name and refers to a different resource maintained by the server.

Currently, MySQL supports one structured variable type. It specifies parameters that govern the operation of key caches. A key cache structured variable has these components:

- `key_buffer_size`
- `key_cache_block_size`
- `key_cache_division_limit`
- `key_cache_age_threshold`

The purpose of this section is to describe the syntax for referring to structured variables. Key cache variables are used for syntax examples, but specific details about how key caches operate are found elsewhere, in Section 7.4.6, “The `MyISAM` Key Cache”.

To refer to a component of a structured variable instance, you can use a compound name in `instance_name.component_name` format. Examples:

```
hot_cache.key_buffer_size
hot_cache.key_cache_block_size
cold_cache.key_cache_block_size
```

For each structured system variable, an instance with the name of `default` is always predefined. If you refer to a component of a structured variable without any instance name, the `default` instance is used. Thus, `default.key_buffer_size` and `key_buffer_size` both refer to the same system variable.

The naming rules for structured variable instances and components are as follows:

- For a given type of structured variable, each instance must have a name that is unique *within* variables of that type. However, instance names need not be unique *across* structured variable types. For example, each structured variable has an instance named `default`, so `default` is not unique across variable types.
- The names of the components of each structured variable type must be unique across all system variable names. If this were not true (that is, if two different types of structured variables could share component member names), it would not be clear which default structured variable to use for references to member names that are not qualified by an instance name.
- If a structured variable instance name is not legal as an unquoted identifier, refer to it as a quoted identifier using backticks. For example, `hot-cache` is not legal, but `` `hot-cache` `` is.
- `global`, `session`, and `local` are not legal instance names. This avoids a conflict with notation such as `@@global.`*`var_name`* for referring to non-structured system variables.

At the moment, the first two rules have no possibility of being violated because the only structured variable type is the one for key caches. These rules assume greater significance if some other type of structured variable is created in the future.

With one exception, it is allowable to refer to structured variable components using compound names in any context where simple variable names can occur. For example, you can assign a value to a structured variable using a command-line option:

```
shell> mysqld --hot_cache.key_buffer_size=64K
```

In an option file, do this:

```
[mysqld]
hot_cache.key_buffer_size=64K
```

If you start the server with such an option, it creates a key cache named `hot_cache` with a size of 64KB in addition to the default key cache that has a default size of 8MB.

Suppose that you start the server as follows:

```
shell> mysqld --key_buffer_size=256K \
         --extra_cache.key_buffer_size=128K \
         --extra_cache.key_cache_block_size=2048
```

In this case, the server sets the size of the default key cache to 256KB. (You could also have written `--default.key_buffer_size=256K`.) In addition, the server creates a second key cache named `extra_cache` that has a size of 128KB, with the size of block buffers for caching table index blocks set to 2048 bytes.

The following example starts the server with three different key caches having sizes in a 3:1:1 ratio:

```
shell> mysqld --key_buffer_size=6M \
         --hot_cache.key_buffer_size=2M \
         --cold_cache.key_buffer_size=2M
```

Structured variable values may be set and retrieved at runtime as well. For example, to set a key cache named `hot_cache` to a size of 10MB, use either of these statements:

```
mysql> SET GLOBAL hot_cache.key_buffer_size = 10*1024*1024;
mysql> SET @@global.hot_cache.key_buffer_size = 10*1024*1024;
```

To retrieve the cache size, do this:

```
mysql> SELECT @@global.hot_cache.key_buffer_size;
```

However, the following statement does not work. The variable is not interpreted as a compound name, but as a simple string for a `LIKE` pattern-matching operation:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'hot_cache.key_buffer_size';
```

This is the exception to being able to use structured variable names anywhere a simple variable name may occur.

9.5. Comment Syntax

The MySQL server supports three comment styles:

- From a '#' character to the end of the line.
- From a '-- ' sequence to the end of the line. This style is supported as of MySQL 3.23.3. Note that the '-- ' (double-dash) comment style requires the second dash to be followed by at least one space (or by a control character such as a newline). This syntax differs slightly from standard SQL comment syntax, as discussed in Section 1.5.5.7, "'--' as the Start of a Comment".
- From a '/*' sequence to the following '*/' sequence. The closing sequence need not be on the same line, so this syntax allows a comment to extend over multiple lines.

The following example demonstrates all three comment styles:

```
mysql> SELECT 1+1;     # This comment continues to the end of line
mysql> SELECT 1+1;     -- This comment continues to the end of line
mysql> SELECT 1 /* this is an in-line comment */ + 1;
mysql> SELECT 1+
/*
this is a
multiple-line comment
*/
1;
```

The comment syntax just described applies to how the **mysqld** server parses SQL statements. The **mysql** client program also performs some parsing of statements before sending them to the server. (For example, it does this to determine statement boundaries within a multiple-statement input line.) However, there are some limitations on the way that **mysql** parses /* ... */ comments:

- A semicolon within the comment is taken to indicate the end of the current SQL statement and anything following it to indicate the beginning of the next statement. This problem was fixed in MySQL 4.0.13.
- A single quote, double quote, or backtick character is taken to indicate the beginning of a quoted string or identifier, even within a comment. If the quote is not matched by a second quote within the comment, the parser doesn't realize the comment has ended. If you are running **mysql** interactively, you can tell that it has gotten confused like this because the prompt changes from mysql> to '>, ">, or `>. This problem was fixed in MySQL 4.1.1.
- An exclamation point used with this style of comment delimiter (such as /*! ... */) marks portions of SQL statements for conditional execution. For more information and examples, see Section 1.5.4, "MySQL Extensions to Standard SQL".

For affected versions of MySQL, these limitations apply both when you run **mysql** interactively and when you put commands in a file and use **mysql** in batch mode to process the file with **mysql** < ***file_name***.

9.6. Treatment of Reserved Words in MySQL

A common problem stems from trying to use an identifier such as a table or column name that is the name of a built-in MySQL data type or function, such as TIMESTAMP or GROUP. You're allowed to do this (for example, ABS is allowed as a column name). However, by default, no whitespace is allowed in function invocations between the function name and the following '(' character. This requirement allows a function call to be distinguished from a reference to a column name.

A side effect of this behavior is that omitting a space in some contexts causes an identifier to be interpreted as a function name. For example, this statement is legal:

```
mysql> CREATE TABLE abs (val INT);
```

But omitting the space after `abs` causes a syntax error because the statement then appears to invoke the `ABS()` function:

```
mysql> CREATE TABLE abs(val INT);
```

If the server SQL mode includes the `IGNORE_SPACE` mode value, the server allows function invocations to have whitespace between a function name and the following `'('` character. This causes function names to be treated as reserved words. As a result, identifiers that are the same as function names must be quoted as described in Section 9.2, “Database, Table, Index, Column, and Alias Names”. The server SQL mode is controlled as described in Section 5.2.2, “The Server SQL Mode”.

The words in the following table are explicitly reserved in MySQL. Most of them are forbidden by standard SQL as column and/or table names (for example, `GROUP`). A few are reserved because MySQL needs them and (currently) uses a **yacc** parser. A reserved word can be used as an identifier if you quote it.

Word	**Word**	**Word**
ADD	ALL	ALTER
ANALYZE	AND	AS
ASC	ASENSITIVE	BEFORE
BETWEEN	BIGINT	BINARY
BLOB	BOTH	BY
CALL	CASCADE	CASE
CHANGE	CHAR	CHARACTER
CHECK	COLLATE	COLUMN
CONDITION	CONNECTION	CONSTRAINT
CONTINUE	CONVERT	CREATE
CROSS	CURRENT_DATE	CURRENT_TIME
CURRENT_TIMESTAMP	CURRENT_USER	CURSOR
DATABASE	DATABASES	DAY_HOUR
DAY_MICROSECOND	DAY_MINUTE	DAY_SECOND
DEC	DECIMAL	DECLARE
DEFAULT	DELAYED	DELETE
DESC	DESCRIBE	DETERMINISTIC
DISTINCT	DISTINCTROW	DIV
DOUBLE	DROP	DUAL
EACH	ELSE	ELSEIF
ENCLOSED	ESCAPED	EXISTS
EXIT	EXPLAIN	FALSE
FETCH	FLOAT	FOR
FORCE	FOREIGN	FROM
FULLTEXT	GOTO	GRANT
GROUP	HAVING	HIGH_PRIORITY
HOUR_MICROSECOND	HOUR_MINUTE	HOUR_SECOND

IF	IGNORE	IN
INDEX	INFILE	INNER
INOUT	INSENSITIVE	INSERT
INT	INTEGER	INTERVAL
INTO	IS	ITERATE
JOIN	KEY	KEYS
KILL	LEADING	LEAVE
LEFT	LIKE	LIMIT
LINES	LOAD	LOCALTIME
LOCALTIMESTAMP	LOCK	LONG
LONGBLOB	LONGTEXT	LOOP
LOW_PRIORITY	MATCH	MEDIUMBLOB
MEDIUMINT	MEDIUMTEXT	MIDDLEINT
MINUTE_MICROSECOND	MINUTE_SECOND	MOD
MODIFIES	NATURAL	NOT
NO_WRITE_TO_BINLOG	NULL	NUMERIC
ON	OPTIMIZE	OPTION
OPTIONALLY	OR	ORDER
OUT	OUTER	OUTFILE
PRECISION	PRIMARY	PROCEDURE
PURGE	READ	READS
REAL	REFERENCES	REGEXP
RENAME	REPEAT	REPLACE
REQUIRE	RESTRICT	RETURN
REVOKE	RIGHT	RLIKE
SCHEMA	SCHEMAS	SECOND_MICROSECOND
SELECT	SENSITIVE	SEPARATOR
SET	SHOW	SMALLINT
SONAME	SPATIAL	SPECIFIC
SQL	SQLEXCEPTION	SQLSTATE
SQLWARNING	SQL_BIG_RESULT	SQL_CALC_FOUND_ROWS
SQL_SMALL_RESULT	SSL	STARTING
STRAIGHT_JOIN	TABLE	TERMINATED
THEN	TINYBLOB	TINYINT
TINYTEXT	TO	TRAILING
TRIGGER	TRUE	UNDO
UNION	UNIQUE	UNLOCK
UNSIGNED	UPDATE	USAGE
USE	USING	UTC_DATE
UTC_TIME	UTC_TIMESTAMP	VALUES
VARBINARY	VARCHAR	VARCHARACTER
VARYING	WHEN	WHERE

WHILE	WITH	WRITE
XOR	YEAR_MONTH	ZEROFILL

MySQL allows some keywords to be used as unquoted identifiers because many people previously used them. Examples are those in the following list:

- ACTION
- BIT
- DATE
- ENUM
- NO
- TEXT
- TIME
- TIMESTAMP

Chapter 10. Character Set Support

Improved support for character set handling was added to MySQL in Version 4.1. The features described here are as implemented in MySQL 4.1.1. (MySQL 4.1.0 has some but not all of these features, and some of them are implemented differently.)

This chapter discusses the following topics:

- What are character sets and collations?
- The multiple-level default system
- New syntax in MySQL 4.1
- Affected functions and operations
- Unicode support
- The meaning of each individual character set and collation

Character set support currently is included in the `MyISAM`, `MEMORY` (`HEAP`), and (as of MySQL 4.1.2) `InnoDB` storage engines. The `ISAM` storage engine does not include character set support; there are no plans to change this, because `ISAM` is deprecated.

10.1. Character Sets and Collations in General

A **character set** is a set of symbols and encodings. A **collation** is a set of rules for comparing characters in a character set. Let's make the distinction clear with an example of an imaginary character set.

Suppose that we have an alphabet with four letters: 'A', 'B', 'a', 'b'. We give each letter a number: 'A' = 0, 'B' = 1, 'a' = 2, 'b' = 3. The letter 'A' is a symbol, the number 0 is the **encoding** for 'A', and the combination of all four letters and their encodings is a **character set**.

Suppose that we want to compare two string values, 'A' and 'B'. The simplest way to do this is to look at the encodings: 0 for 'A' and 1 for 'B'. Because 0 is less than 1, we say 'A' is less than 'B'. What we've just done is apply a collation to our character set. The collation is a set of rules (only one rule in this case): ``compare the encodings.'' We call this simplest of all possible collations a **binary** collation.

But what if we want to say that the lowercase and uppercase letters are equivalent? Then we would have at least two rules: (1) treat the lowercase letters 'a' and 'b' as equivalent to 'A' and 'B'; (2) then compare the encodings. We call this a **case-insensitive** collation. It's a little more complex than a binary collation.

In real life, most character sets have many characters: not just 'A' and 'B' but whole alphabets, sometimes multiple alphabets or eastern writing systems with thousands of characters, along with many special symbols and punctuation marks. Also in real life, most collations have many rules: not just case insensitivity but also accent insensitivity (an ``accent'' is a mark attached to a character as in German 'Ö') and multiple-character mappings (such as the rule that 'Ö' = 'OE' in one of the two German collations).

MySQL 4.1 can do these things for you:

- Store strings using a variety of character sets
- Compare strings using a variety of collations

- Mix strings with different character sets or collations in the same server, the same database, or even the same table
- Allow specification of character set and collation at any level

In these respects, not only is MySQL 4.1 far more flexible than MySQL 4.0, it also is far ahead of other DBMSs. However, to use the new features effectively, you need to learn what character sets and collations are available, how to change their defaults, and what the various string operators do with them.

10.2. Character Sets and Collations in MySQL

The MySQL server can support multiple character sets. To list the available character sets, use the `SHOW CHARACTER SET` statement:

```
mysql> SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |      2 |
| dec8     | DEC West European           | dec8_swedish_ci     |      1 |
| cp850    | DOS West European           | cp850_general_ci    |      1 |
| hp8      | HP West European            | hp8_english_ci      |      1 |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |      1 |
| latin1   | ISO 8859-1 West European    | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |      1 |
| swe7     | 7bit Swedish                | swe7_swedish_ci     |      1 |
| ascii    | US ASCII                    | ascii_general_ci    |      1 |
| ujis     | EUC-JP Japanese             | ujis_japanese_ci    |      3 |
| sjis     | Shift-JIS Japanese          | sjis_japanese_ci    |      2 |
| hebrew   | ISO 8859-8 Hebrew           | hebrew_general_ci   |      1 |
| tis620   | TIS620 Thai                 | tis620_thai_ci      |      1 |
| euckr    | EUC-KR Korean               | euckr_korean_ci     |      2 |
...
```

Any given character set always has at least one collation. It may have several collations.

To list the collations for a character set, use the `SHOW COLLATION` statement. For example, to see the collations for the `latin1` (``ISO-8859-1 West European'') character set, use this statement to find those collation names that begin with `latin1`:

```
mysql> SHOW COLLATION LIKE 'latin1%';
+-------------------+---------+----+---------+----------+---------+
| Collation         | Charset | Id | Default | Compiled | Sortlen |
+-------------------+---------+----+---------+----------+---------+
| latin1_german1_ci | latin1  |  5 |         |          |       0 |
| latin1_swedish_ci | latin1  |  8 | Yes     | Yes      |       1 |
| latin1_danish_ci  | latin1  | 15 |         |          |       0 |
| latin1_german2_ci | latin1  | 31 |         | Yes      |       2 |
| latin1_bin        | latin1  | 47 |         | Yes      |       1 |
| latin1_general_ci | latin1  | 48 |         |          |       0 |
| latin1_general_cs | latin1  | 49 |         |          |       0 |
| latin1_spanish_ci | latin1  | 94 |         |          |       0 |
+-------------------+---------+----+---------+----------+---------+
```

The `latin1` collations have the following meanings:

Collation	Meaning

latin1_bin	Binary according to latin1 encoding
latin1_danish_ci	Danish/Norwegian
latin1_general_ci	Multilingual
latin1_general_cs	Multilingual, case sensitive
latin1_german1_ci	German DIN-1
latin1_german2_ci	German DIN-2
latin1_spanish_ci	Modern Spanish
latin1_swedish_ci	Swedish/Finnish

Collations have these general characteristics:

- Two different character sets cannot have the same collation.
- Each character set has one collation that is the *default collation*. For example, the default collation for latin1 is latin1_swedish_ci.
- There is a convention for collation names: They start with the name of the character set with which they are associated, they usually include a language name, and they end with _ci (case insensitive), _cs (case sensitive), or _bin (binary).

10.3. Determining the Default Character Set and Collation

There are default settings for character sets and collations at four levels: server, database, table, and connection. The following description may appear complex, but it has been found in practice that multiple-level defaulting leads to natural and obvious results.

10.3.1. Server Character Set and Collation

The MySQL Server has a server character set and a server collation, which may not be null.

MySQL determines the server character set and server collation thus:

- According to the option settings in effect when the server starts
- According to the values set at runtime

At the server level, the decision is simple. The server character set and collation depend initially on the options that you use when you start **mysqld**. You can use --default-character-set for the character set, and along with it you can add --default-collation for the collation. If you don't specify a character set, that is the same as saying --default-character-set=latin1. If you specify only a character set (for example, latin1) but not a collation, that is the same as saying --default-charset=latin1 --default-collation=latin1_swedish_ci because latin1_swedish_ci is the default collation for latin1. Therefore, the following three commands all have the same effect:

```
shell> mysqld
shell> mysqld --default-character-set=latin1
```

```
shell> mysqld --default-character-set=latin1 \
           --default-collation=latin1_swedish_ci
```

One way to change the settings is by recompiling. If you want to change the default server character set and collation when building from sources, use: `--with-charset` and `--with-collation` as arguments for **configure**. For example:

```
shell> ./configure --with-charset=latin1
```

Or:

```
shell> ./configure --with-charset=latin1 \
           --with-collation=latin1_german1_ci
```

Both **mysqld** and **configure** verify that the character set/collation combination is valid. If not, each program displays an error message and terminates.

The current server character set and collation are available as the values of the `character_set_server` and `collation_server` system variables. These variables can be changed at runtime.

10.3.2. Database Character Set and Collation

Every database has a database character set and a database collation, which may not be null. The `CREATE DATABASE` and `ALTER DATABASE` statements have optional clauses for specifying the database character set and collation:

```
CREATE DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]

ALTER DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]
```

Example:

```
CREATE DATABASE db_name
    DEFAULT CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

MySQL chooses the database character set and database collation thus:

- If both `CHARACTER SET` *X* and `COLLATE` *Y* were specified, then character set *X* and collation *Y*.
- If `CHARACTER SET` *X* was specified without `COLLATE`, then character set *X* and its default collation.
- Otherwise, the server character set and server collation.

MySQL's `CREATE DATABASE ... DEFAULT CHARACTER SET ...` syntax is analogous to the standard SQL `CREATE SCHEMA ... CHARACTER SET ...` syntax. Because of this, it is possible to create databases with different character sets and collations on the same MySQL server.

The database character set and collation are used as default values if the table character set and collation are not specified in `CREATE TABLE` statements. They have no other purpose.

The character set and collation for the default database are available as the values of the `character_set_database` and `collation_database` system variables. The server sets these variables whenever the default database changes. If there is no default database, the variables have the same value as the corresponding server-level variables, `character_set_server` and `collation_server`.

10.3.3. Table Character Set and Collation

Every table has a table character set and a table collation, which may not be null. The `CREATE TABLE` and `ALTER TABLE` statements have optional clauses for specifying the table character set and collation:

```
CREATE TABLE tbl_name (column_list)
    [DEFAULT CHARACTER SET charset_name [COLLATE collation_name]]

ALTER TABLE tbl_name
    [DEFAULT CHARACTER SET charset_name] [COLLATE collation_name]
```

Example:

```
CREATE TABLE t1 ( ... )
    DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

MySQL chooses the table character set and collation thus:

- If both `CHARACTER SET` *X* and `COLLATE` *Y* were specified, then character set *X* and collation *Y*.
- If `CHARACTER SET` *X* was specified without `COLLATE`, then character set *X* and its default collation.
- Otherwise, the database character set and collation.

The table character set and collation are used as default values if the column character set and collation are not specified in individual column definitions. The table character set and collation are MySQL extensions; there are no such things in standard SQL.

10.3.4. Column Character Set and Collation

Every ``character'' column (that is, a column of type `CHAR`, `VARCHAR`, or `TEXT`) has a column character set and a column collation, which may not be null. Column definition syntax has optional clauses for specifying the column character set and collation:

```
col_name {CHAR | VARCHAR | TEXT} (col_length)
    [CHARACTER SET charset_name [COLLATE collation_name]]
```

Example:

```
CREATE TABLE Table1
(
    column1 VARCHAR(5) CHARACTER SET latin1 COLLATE latin1_german1_ci
);
```

MySQL chooses the column character set and collation thus:

- If both `CHARACTER SET X` and `COLLATE Y` were specified, then character set *X* and collation *Y*.
- If `CHARACTER SET X` was specified without `COLLATE`, then character set `X` and its default collation.
- Otherwise, the table character set and collation.

The `CHARACTER SET` and `COLLATE` clauses are standard SQL.

10.3.5. Examples of Character Set and Collation Assignment

The following examples show how MySQL determines default character set and collation values.

Example 1: Table + Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10) CHARACTER SET latin1 COLLATE latin1_german1_ci
) DEFAULT CHARACTER SET latin2 COLLATE latin2_bin;
```

Here we have a column with a `latin1` character set and a `latin1_german1_ci` collation. The definition is explicit, so that's straightforward. Notice that there's no problem storing a `latin1` column in a `latin2` table.

Example 2: Table + Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10) CHARACTER SET latin1
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

This time we have a column with a `latin1` character set and a default collation. Although it might seem natural, the default collation is not taken from the table level. Instead, because the default collation for `latin1` is always `latin1_swedish_ci`, column `c1` has a collation of `latin1_swedish_ci` (not `latin1_danish_ci`).

Example 3: Table + Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10)
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

We have a column with a default character set and a default collation. In this circumstance, MySQL looks up to the table level for inspiration in determining the column character set and collation. So, the character set for column `c1` is `latin1` and its collation is `latin1_danish_ci`.

Example 4: Database + Table + Column Definition

```
CREATE DATABASE d1
    DEFAULT CHARACTER SET latin2 COLLATE latin2_czech_ci;
```

```
USE d1;
CREATE TABLE t1
(
    c1 CHAR(10)
);
```

We create a column without specifying its character set and collation. We're also not specifying a character set and a collation at the table level. In this circumstance, MySQL looks up to the database level for inspiration. (The database's settings become the table's settings, and thereafter become the column's setting.) So, the character set for column `c1` is `latin2` and its collation is `latin2_czech_ci`.

10.3.6. Connection Character Sets and Collations

Several character set and collation system variables relate to a client's interaction with the server. Some of these have been mentioned in earlier sections:

- The server character set and collation are available as the values of the `character_set_server` and `collation_server` variables.
- The character set and collation of the default database are available as the values of the `character_set_database` and `collation_database` variables.

Additional character set and collation variables are involved in handling traffic for the connection between a client and the server. Every client has connection-related character set and collation variables.

Consider what a ``connection'' is: It's what you make when you connect to the server. The client sends SQL statements, such as queries, over the connection to the server. The server sends responses, such as result sets, over the connection back to the client. This leads to several questions about character set and collation handling for client connections, each of which can be answered in terms of system variables:

- What character set is the query in when it leaves the client?

  The server takes the `character_set_client` variable to be the character set in which queries are sent by the client.

- What character set should the server translate a query to after receiving it?

  For this, `character_set_connection` and `collation_connection` are used by the server. It converts queries sent by the client from `character_set_client` to `character_set_connection` (except for string literals that have an introducer such as `_latin1` or `_utf8`). `collation_connection` is important for comparisons of literal strings. For comparisons of strings with column values, it does not matter because columns have a higher collation precedence.

- What character set should the server translate to before shipping result sets or error messages back to the client?

  The `character_set_results` variable indicates the character set in which the server returns query results to the client. This includes result data such as column values, and result metadata such as column names.

You can fine-tune the settings for these variables, or you can depend on the defaults (in which case, you can skip this section).

There are two statements that affect the connection character sets:

```
SET NAMES 'charset_name'
SET CHARACTER SET charset_name
```

`SET NAMES` indicates what is in the SQL statements that the client sends. Thus, `SET NAMES 'cp1251'` tells the server ``future incoming messages from this client are in character set `cp1251`." It also specifies the character set for results that the server sends back to the client. (For example, it indicates what character set column values are if you use a `SELECT` statement.)

A `SET NAMES 'x'` statement is equivalent to these three statements:

```
mysql> SET character_set_client = x;
mysql> SET character_set_results = x;
mysql> SET character_set_connection = x;
```

Setting `character_set_connection` to `x` also sets `collation_connection` to the default collation for `x`.

`SET CHARACTER SET` is similar but sets the connection character set and collation to be those of the default database. A `SET CHARACTER SET x` statement is equivalent to these three statements:

```
mysql> SET character_set_client = x;
mysql> SET character_set_results = x;
mysql> SET collation_connection = @@collation_database;
```

When a client connects, it sends to the server the name of the character set that it wants to use. The server sets the `character_set_client`, `character_set_results`, and `character_set_connection` variables to that character set. (In effect, the server performs a `SET NAMES` operation using the character set.)

With the **mysql** client, it is not necessary to execute `SET NAMES` every time you start up if you want to use a character set different from the default. You can add the `--default-character-set` option setting to your **mysql** statement line, or in your option file. For example, the following option file setting changes the three character set variables set to `koi8r` each time you run **mysql**:

```
[mysql]
default-character-set=koi8r
```

Example: Suppose that `column1` is defined as `CHAR(5) CHARACTER SET latin2`. If you do not say `SET NAMES` or `SET CHARACTER SET`, then for `SELECT column1 FROM t`, the server sends back all the values for `column1` using the character set that the client specified when it connected. On the other hand, if you say `SET NAMES 'latin1'` or `SET CHARACTER SET latin1`, then just before sending results back, the server converts the `latin2` values to `latin1`. Conversion may be lossy if there are characters that are not in both character sets.

If you do not want the server to perform any conversion, set `character_set_results` to `NULL`:

```
mysql> SET character_set_results = NULL;
```

10.3.7. Character String Literal Character Set and Collation

Every character string literal has a character set and a collation, which may not be null.

A character string literal may have an optional character set introducer and `COLLATE` clause:

```
[_charset_name]'string' [COLLATE collation_name]
```

Examples:

```
SELECT 'string';
SELECT _latin1'string';
SELECT _latin1'string' COLLATE latin1_danish_ci;
```

For the simple statement `SELECT 'string'`, the string has the character set and collation defined by the `character_set_connection` and `collation_connection` system variables.

The `_charset_name` expression is formally called an *introducer*. It tells the parser, ``the string that is about to follow is in character set `X`." Because this has confused people in the past, we emphasize that an introducer does not cause any conversion, it is strictly a signal that does not change the string's value. An introducer is also legal before standard hex literal and numeric hex literal notation (`x'literal'` and `0xnnnn`), and before `?` (parameter substitution when using prepared statements within a programming language interface).

Examples:

```
SELECT _latin1 x'AABBCC';
SELECT _latin1 0xAABBCC;
SELECT _latin1 ?;
```

MySQL determines a literal's character set and collation thus:

- If both `_X` and `COLLATE Y` were specified, then character set `X` and collation `Y`
- If `_X` is specified but `COLLATE` is not specified, then character set `X` and its default collation
- Otherwise, the character set and collation given by the `character_set_connection` and `collation_connection` system variables

Examples:

- A string with `latin1` character set and `latin1_german1_ci` collation:

  ```
  SELECT _latin1'Müller' COLLATE latin1_german1_ci;
  ```

- A string with `latin1` character set and its default collation (that is, `latin1_swedish_ci`):

  ```
  SELECT _latin1'Müller';
  ```

- A string with the connection default character set and collation:

  ```
  SELECT 'Müller';
  ```

Character set introducers and the `COLLATE` clause are implemented according to standard SQL specifications.

10.3.8. Using COLLATE in SQL Statements

With the COLLATE clause, you can override whatever the default collation is for a comparison. COLLATE may be used in various parts of SQL statements. Here are some examples:

- With ORDER BY:

  ```
  SELECT k
  FROM t1
  ORDER BY k COLLATE latin1_german2_ci;
  ```

- With AS:

  ```
  SELECT k COLLATE latin1_german2_ci AS k1
  FROM t1
  ORDER BY k1;
  ```

- With GROUP BY:

  ```
  SELECT k
  FROM t1
  GROUP BY k COLLATE latin1_german2_ci;
  ```

- With aggregate functions:

  ```
  SELECT MAX(k COLLATE latin1_german2_ci)
  FROM t1;
  ```

- With DISTINCT:

  ```
  SELECT DISTINCT k COLLATE latin1_german2_ci
  FROM t1;
  ```

- With WHERE:

  ```
  SELECT *
  FROM t1
  WHERE _latin1 'Müller' COLLATE latin1_german2_ci = k;
  ```

- With HAVING:

  ```
  SELECT k
  FROM t1
  GROUP BY k
  HAVING k = _latin1 'Müller' COLLATE latin1_german2_ci;
  ```

10.3.9. COLLATE Clause Precedence

The COLLATE clause has high precedence (higher than ||), so the following two expressions are equivalent:

```
x || y COLLATE z
```

```
x || (y COLLATE z)
```

10.3.10. BINARY Operator

The BINARY operator is a shorthand for a COLLATE clause. BINARY 'x' is equivalent to 'x' COLLATE y, where y is the name of the binary collation for the character set of 'x'. Every character set has a binary collation. For example, the binary collation for the latin1 character set is latin1_bin, so if the column a is of character set latin1, the following two statements have the same effect:

```
SELECT * FROM t1 ORDER BY BINARY a;
SELECT * FROM t1 ORDER BY a COLLATE latin1_bin;
```

10.3.11. Some Special Cases Where the Collation Determination Is Tricky

In the great majority of queries, it is obvious what collation MySQL uses to resolve a comparison operation. For example, in the following cases, it should be clear that the collation is ``the column collation of column x'':

```
SELECT x FROM T ORDER BY x;
SELECT x FROM T WHERE x = x;
SELECT DISTINCT x FROM T;
```

However, when multiple operands are involved, there can be ambiguity. For example:

```
SELECT x FROM T WHERE x = 'Y';
```

Should this query use the collation of the column x, or of the string literal 'Y'?

Standard SQL resolves such questions using what used to be called ``coercibility'' rules. The essence is: Because x and 'Y' both have collations, whose collation takes precedence? It's complex, but the following rules take care of most situations:

- An explicit COLLATE clause has a coercibility of 0. (Not coercible at all.)
- A concatenation of two strings with different collations has a coercibility of 1.
- A column's collation has a coercibility of 2.
- A ``system constant'' (the string returned by functions such as USER() or VERSION()) has a coercibility of 3.
- A literal's collation has a coercibility of 4.
- NULL or an expression that is derived from NULL has a coercibility of 5.

The preceding coercibility values are current as of MySQL 4.1.11 and 5.0.3. See the note later in this section for additional version-related information.

Those rules resolve ambiguities thus:

- Use the collation with the lowest coercibility value.
- If both sides have the same coercibility, then it is an error if the collations aren't the same.

Examples:

`column1 = 'A'`	Use collation of `column1`
`column1 = 'A' COLLATE x`	Use collation of `'A'`
`column1 COLLATE x = 'A' COLLATE y`	Error

The `COERCIBILITY()` function can be used to determine the coercibility of a string expression:

```
mysql> SELECT COERCIBILITY('A' COLLATE latin1_swedish_ci);
        -> 0
mysql> SELECT COERCIBILITY(VERSION());
        -> 3
mysql> SELECT COERCIBILITY('A');
        -> 4
```

See Section 12.8.3, “Information Functions”.

Before MySQL 4.1.11 and 5.0.3, there is no system constant or ignorable coercibility. Functions such as `USER()` have a coercibility of 2 rather than 3, and literals have a coercibility of 3 rather than 4.

10.3.12. Collations Must Be for the Right Character Set

Recall that each character set has one or more collations, and each collation is associated with one and only one character set. Therefore, the following statement causes an error message because the `latin2_bin` collation is not legal with the `latin1` character set:

```
mysql> SELECT _latin1 'x' COLLATE latin2_bin;
ERROR 1251: COLLATION 'latin2_bin' is not valid
for CHARACTER SET 'latin1'
```

In some cases, expressions that worked before MySQL 4.1 fail in early versions of MySQL 4.1 if you do not take character set and collation into account. For example, before 4.1, this statement works as is:

```
mysql> SELECT SUBSTRING_INDEX(USER(),'@',1);
+-------------------------------+
| SUBSTRING_INDEX(USER(),'@',1) |
+-------------------------------+
| root                          |
+-------------------------------+
```

The statement also works as is in MySQL 4.1 as of 4.1.8: In MySQL 4.1, usernames are stored using the `utf8` character set (see Section 10.6, “UTF8 for Metadata”). The literal string `'@'` has the server character set (`latin1` by default). Although the character sets are different, MySQL can coerce the `latin1` string to the character set (and collation) of `USER()` without data loss. It does so, performs the substring operation, and returns a result that has a character set of `utf8`.

However, in versions of MySQL 4.1 before 4.1.8, the statement fails:

```
mysql> SELECT SUBSTRING_INDEX(USER(),'@',1);
ERROR 1267 (HY000): Illegal mix of collations
```

```
(utf8_general_ci,IMPLICIT) and (latin1_swedish_ci,COERCIBLE)
for operation 'substr_index'
```

This happens because the automatic character set conversion of `'@'` does not occur and the string operands have different character sets (and thus different collations):

```
mysql> SELECT COLLATION(USER()), COLLATION('@');
+-------------------+-------------------+
| COLLATION(USER()) | COLLATION('@')    |
+-------------------+-------------------+
| utf8_general_ci   | latin1_swedish_ci |
+-------------------+-------------------+
```

One way to deal with this is to upgrade to MySQL 4.1.8 or later. If that is not possible, you can tell MySQL to interpret the literal string as `utf8`:

```
mysql> SELECT SUBSTRING_INDEX(USER(),_utf8'@',1);
+-------------------------------------+
| SUBSTRING_INDEX(USER(),_utf8'@',1) |
+-------------------------------------+
| root                                |
+-------------------------------------+
```

Another way is to change the connection character set and collation to `utf8`. You can do that with `SET NAMES 'utf8'` or by setting the `character_set_connection` and `collation_connection` system variables directly.

10.3.13. An Example of the Effect of Collation

Suppose that column `X` in table `T` has these `latin1` column values:

```
Muffler
Müller
MX Systems
MySQL
```

And suppose that the column values are retrieved using the following statement:

```
SELECT X FROM T ORDER BY X COLLATE collation_name;
```

The resulting order of the values for different collations is shown in this table:

latin1_swedish_ci	latin1_german1_ci	latin1_german2_ci
Muffler	Muffler	Müller
MX Systems	Müller	Muffler
Müller	MX Systems	MX Systems
MySQL	MySQL	MySQL

The table is an example that shows what the effect would be if we used different collations in an `ORDER BY` clause. The character that causes the different sort orders in this example is the U with two dots over it, which the Germans call U-umlaut, but we'll call it U-diaeresis.

- The first column shows the result of the `SELECT` using the Swedish/Finnish collating rule, which says that U-diaeresis sorts with Y.
- The second column shows the result of the `SELECT` using the German DIN-1 rule, which says that U-diaeresis sorts with U.
- The third column shows the result of the `SELECT` using the German DIN-2 rule, which says that U-diaeresis sorts with UE.

Three different collations, three different results. That's what MySQL is here to handle. By using the appropriate collation, you can choose the sort order you want.

10.4. Operations Affected by Character Set Support

This section describes operations that take character set information into account as of MySQL 4.1.

10.4.1. Result Strings

MySQL has many operators and functions that return a string. This section answers the question: What is the character set and collation of such a string?

For simple functions that take string input and return a string result as output, the output's character set and collation are the same as those of the principal input value. For example, `UPPER(X)` returns a string whose character string and collation are the same as that of *X*. The same applies for `INSTR()`, `LCASE()`, `LOWER()`, `LTRIM()`, `MID()`, `REPEAT()`, `REPLACE()`, `REVERSE()`, `RIGHT()`, `RPAD()`, `RTRIM()`, `SOUNDEX()`, `SUBSTRING()`, `TRIM()`, `UCASE()`, and `UPPER()`. (Also note: The `REPLACE()` function, unlike all other functions, ignores the collation of the string input and performs a case-insensitive comparison every time.)

For operations that combine multiple string inputs and return a single string output, the ``aggregation rules'' of standard SQL apply:

- If an explicit `COLLATE X` occurs, then use `X`
- If an explicit `COLLATE X` and `COLLATE Y` occur, then error
- Otherwise, if all collations are `X`, then use `X`
- Otherwise, the result has no collation

For example, with `CASE ... WHEN a THEN b WHEN b THEN c COLLATE X END`, the resultant collation is `X`. The same applies for `CASE`, `UNION`, `||`, `CONCAT()`, `ELT()`, `GREATEST()`, `IF()`, and `LEAST()`.

For operations that convert to character data, the character set and collation of the strings that result from the operations are defined by the `character_set_connection` and `collation_connection` system variables. This applies for `CAST()`, `CHAR()`, `CONV()`, `FORMAT()`, `HEX()`, and `SPACE()`.

10.4.2. `CONVERT()`

`CONVERT()` provides a way to convert data between different character sets. The syntax is:

```
CONVERT(expr USING transcoding_name)
```

In MySQL, transcoding names are the same as the corresponding character set names.

Examples:

```
SELECT CONVERT(_latin1'Müller' USING utf8);
INSERT INTO utf8table (utf8column)
    SELECT CONVERT(latin1field USING utf8) FROM latin1table;
```

`CONVERT(... USING ...)` is implemented according to the standard SQL specification.

10.4.3. CAST()

You may also use `CAST()` to convert a string to a different character set. The syntax is:

```
CAST(character_string AS character_data_type CHARACTER SET charset_name)
```

Example:

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8);
```

If you use `CAST()` without specifying `CHARACTER SET`, the resulting character set and collation are defined by the `character_set_connection` and `collation_connection` system variables. If you use `CAST()` with `CHARACTER SET X`, then the resulting character set and collation are `X` and the default collation of `X`.

You may not use a `COLLATE` clause inside a `CAST()`, but you may use it outside. That is, `CAST(... COLLATE ...)` is illegal, but `CAST(...) COLLATE ...` is legal.

Example:

```
SELECT CAST(_latin1'test' AS CHAR CHARACTER SET utf8) COLLATE utf8_bin;
```

10.4.4. SHOW Statements

Several `SHOW` statements are new or modified in MySQL 4.1 to provide additional character set information. `SHOW CHARACTER SET`, `SHOW COLLATION`, and `SHOW CREATE DATABASE` are new. `SHOW CREATE TABLE` and `SHOW COLUMNS` are modified.

The `SHOW CHARACTER SET` command shows all available character sets. It takes an optional `LIKE` clause that indicates which character set names to match. For example:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+---------+-----------------------------+-------------------+--------+
| Charset | Description                 | Default collation | Maxlen |
+---------+-----------------------------+-------------------+--------+
| latin1  | ISO 8859-1 West European    | latin1_swedish_ci |      1 |
| latin2  | ISO 8859-2 Central European | latin2_general_ci |      1 |
| latin5  | ISO 8859-9 Turkish          | latin5_turkish_ci |      1 |
| latin7  | ISO 8859-13 Baltic          | latin7_general_ci |      1 |
+---------+-----------------------------+-------------------+--------+
```

See Section 13.5.4.1, “`SHOW CHARACTER SET` Syntax”.

The output from SHOW COLLATION includes all available character sets. It takes an optional LIKE clause that indicates which collation names to match. For example:

```
mysql> SHOW COLLATION LIKE 'latin1%';
+-------------------+---------+----+---------+----------+---------+
| Collation         | Charset | Id | Default | Compiled | Sortlen |
+-------------------+---------+----+---------+----------+---------+
| latin1_german1_ci | latin1  |  5 |         |          |       0 |
| latin1_swedish_ci | latin1  |  8 | Yes     | Yes      |       0 |
| latin1_danish_ci  | latin1  | 15 |         |          |       0 |
| latin1_german2_ci | latin1  | 31 |         | Yes      |       2 |
| latin1_bin        | latin1  | 47 |         | Yes      |       0 |
| latin1_general_ci | latin1  | 48 |         |          |       0 |
| latin1_general_cs | latin1  | 49 |         |          |       0 |
| latin1_spanish_ci | latin1  | 94 |         |          |       0 |
+-------------------+---------+----+---------+----------+---------+
```

See Section 13.5.4.2, “SHOW COLLATION Syntax”.

SHOW CREATE DATABASE displays the CREATE DATABASE statement that creates a given database. The result includes all database options. DEFAULT CHARACTER SET and COLLATE are supported. All database options are stored in a text file named db.opt that can be found in the database directory.

```
mysql> SHOW CREATE DATABASE a\G
*************************** 1. row ***************************
       Database: a
Create Database: CREATE DATABASE `a`
                 /*!40100 DEFAULT CHARACTER SET macce */
```

See Section 13.5.4.4, “SHOW CREATE DATABASE Syntax”.

SHOW CREATE TABLE is similar, but displays the CREATE TABLE statement to create a given table. The column definitions indicate any character set specifications, and the table options include character set information.

See Section 13.5.4.5, “SHOW CREATE TABLE Syntax”.

The SHOW COLUMNS statement displays the collations of a table's columns when invoked as SHOW FULL COLUMNS. Columns with CHAR, VARCHAR, or TEXT data types have non-NULL collations. Numeric and other non-character types have NULL collations. For example:

```
mysql> SHOW FULL COLUMNS FROM t;
+-------+---------+------------+------+-----+---------+-------+
| Field | Type    | Collation  | Null | Key | Default | Extra |
+-------+---------+------------+------+-----+---------+-------+
| a     | char(1) | latin1_bin | YES  |     | NULL    |       |
| b     | int(11) | NULL       | YES  |     | NULL    |       |
+-------+---------+------------+------+-----+---------+-------+
```

The character set is not part of the display. (The character set name is implied by the collation name.)

See Section 13.5.4.3, “SHOW COLUMNS Syntax”.

10.5. Unicode Support

As of MySQL version 4.1, there are two new character sets for storing Unicode data:

- `ucs2`, the UCS-2 Unicode character set.
- `utf8`, the UTF8 encoding of the Unicode character set.

In UCS-2 (binary Unicode representation), every character is represented by a two-byte Unicode code with the most significant byte first. For example: "LATIN CAPITAL LETTER A" has the code 0x0041 and it's stored as a two-byte sequence: 0x00 0x41. "CYRILLIC SMALL LETTER YERU" (Unicode 0x044B) is stored as a two-byte sequence: 0x04 0x4B. For Unicode characters and their codes, please refer to the Unicode Home Page [http://www.unicode.org/].

A temporary restriction is that UCS-2 cannot yet be used as a client character set. That means that `SET NAMES 'ucs2'` does not work.

The UTF8 character set (transform Unicode representation) is an alternative way to store Unicode data. It is implemented according to RFC 3629. The idea of the UTF8 character set is that various Unicode characters are encoded using byte sequences of different lengths:

- Basic Latin letters, digits, and punctuation signs use one byte.
- Most European and Middle East script letters fit into a two-byte sequence: extended Latin letters (with tilde, macron, acute, grave and other accents), Cyrillic, Greek, Armenian, Hebrew, Arabic, Syriac, and others.
- Korean, Chinese, and Japanese ideographs use three-byte sequences.

RFC 3629 describes encoding sequences that take from one to four bytes. Currently, MySQL UTF8 support does not include four-byte sequences. (An older standard for UTF8 encoding is given by RFC 2279, which describes UTF8 sequences that take from one to six bytes. RFC 3629 renders RFC 2279 obsolete and sequences with five and six bytes now are not used.)

Tip: To save space with UTF8, use `VARCHAR` instead of `CHAR`. Otherwise, MySQL has to reserve 30 bytes for a `CHAR(10) CHARACTER SET utf8` column, because that's the maximum possible length.

10.6. UTF8 for Metadata

The metadata is the data about the data. Anything that describes the database, as opposed to being the contents of the database, is metadata. Thus column names, database names, usernames, version names, and most of the string results from `SHOW` are metadata.

Representation of metadata must satisfy these requirements:

- All metadata must be in the same character set. Otherwise, `SHOW` wouldn't work properly because different rows in the same column would be in different character sets.
- Metadata must include all characters in all languages. Otherwise, users wouldn't be able to name columns and tables in their own languages.

In order to satisfy both requirements, MySQL stores metadata in a Unicode character set, namely UTF8. This does not cause any disruption if you never use accented characters. But if you do, you should be aware that metadata is in UTF8.

This means that the `USER()`, `CURRENT_USER()`, `DATABASE()`, and `VERSION()` functions have

the UTF8 character set by default, as do synonyms such as `SESSION_USER()` and `SYSTEM_USER()`.

The server sets the `character_set_system` system variable to the name of the metadata character set:

```
mysql> SHOW VARIABLES LIKE 'character_set_system';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
| character_set_system | utf8  |
+----------------------+-------+
```

Storage of metadata using Unicode does *not* mean that the headers of columns and the results of `DESCRIBE` functions are in the `character_set_system` character set by default. When you say `SELECT column1 FROM t`, the name `column1` itself is returned from the server to the client in the character set as determined by the `SET NAMES` statement. More specifically, the character set used is determined by the value of the `character_set_results` system variable. If this variable is set to `NULL`, no conversion is performed and the server returns metadata using its original character set (the set indicated by `character_set_system`).

If you want the server to pass metadata results back in a non-UTF8 character set, then use `SET NAMES` to force the server to perform character set conversion (see Section 10.3.6, “Connection Character Sets and Collations”), or else set the client to do the conversion. It is always more efficient to set the client to do the conversion, but this option is not available for many clients until late in the MySQL 4.x product cycle.

If you are just using, for example, the `USER()` function for comparison or assignment within a single statement, don't worry. MySQL does some automatic conversion for you.

```
SELECT * FROM Table1 WHERE USER() = latin1_column;
```

This works because the contents of `latin1_column` are automatically converted to UTF8 before the comparison.

```
INSERT INTO Table1 (latin1_column) SELECT USER();
```

This works because the contents of `USER()` are automatically converted to `latin1` before the assignment. Automatic conversion is not fully implemented yet, but should work correctly in a later version.

Although automatic conversion is not in the SQL standard, the SQL standard document does say that every character set is (in terms of supported characters) a ``subset'' of Unicode. Since it is a well-known principle that ``what applies to a superset can apply to a subset,'' we believe that a collation for Unicode can apply for comparisons with non-Unicode strings.

10.7. Compatibility with Other DBMSs

For MaxDB compatibility these two statements are the same:

```
CREATE TABLE t1 (f1 CHAR(n) UNICODE);
CREATE TABLE t1 (f1 CHAR(n) CHARACTER SET ucs2);
```

10.8. New Character Set Configuration File Format

In MySQL 4.1, character set configuration is stored in XML files, one file per character set. In previous versions, this information was stored in `.conf` files.

10.9. National Character Set

Before MySQL 4.1, `NCHAR` and `CHAR` were synonymous. ANSI defines `NCHAR` or `NATIONAL CHAR` as a way to indicate that a `CHAR` column should use some predefined character set. MySQL 4.1 and up uses `utf8` as that predefined character set. For example, these column type declarations are equivalent:

```
CHAR(10) CHARACTER SET utf8
NATIONAL CHARACTER(10)
NCHAR(10)
```

As are these:

```
VARCHAR(10) CHARACTER SET utf8
NATIONAL VARCHAR(10)
NCHAR VARCHAR(10)
NATIONAL CHARACTER VARYING(10)
NATIONAL CHAR VARYING(10)
```

You can use `N'literal'` to create a string in the national character set. These two statements are equivalent:

```
SELECT N'some text';
SELECT _utf8'some text';
```

10.10. Upgrading Character Sets from MySQL 4.0

What about upgrading from older versions of MySQL? MySQL 4.1 is almost upward compatible with MySQL 4.0 and earlier for the simple reason that almost all the features are new, so there's nothing in earlier versions to conflict with. However, there are some differences and a few things to be aware of.

Most important: The ``MySQL 4.0 character set'' has the properties of both ``MySQL 4.1 character sets'' and ``MySQL 4.1 collations.'' You have to unlearn this. Henceforth, we does not bundle character set/collation properties in the same conglomerate object.

There is a special treatment of national character sets in MySQL 4.1. `NCHAR` is not the same as `CHAR`, and `N'...'` literals are not the same as `'...'` literals.

Finally, there is a different file format for storing information about character sets and collations. Make sure that you have reinstalled the `/share/mysql/charsets/` directory containing the new configuration files.

If you want to start **mysqld** from a 4.1.x distribution with data created by MySQL 4.0, you should start the server with the same character set and collation. In this case, you won't need to reindex your data.

There are two ways to do so:

```
shell> ./configure --with-charset=... --with-collation=...
shell> ./mysqld --default-character-set=... --default-collation=...
```

If you used **mysqld** with, for example, the MySQL 4.0 `danish` character set, you should use the `latin1` character set and the `latin1_danish_ci` collation:

```
shell> ./configure --with-charset=latin1 \
           --with-collation=latin1_danish_ci
shell> ./mysqld --default-character-set=latin1 \
           --default-collation=latin1_danish_ci
```

Use the table shown in Section 10.10.1, "4.0 Character Sets and Corresponding 4.1 Character Set/Collation Pairs" to find old 4.0 character set names and their 4.1 character set/collation pair equivalents.

If you have non-`latin1` data stored in a 4.0 `latin1` table and want to convert the table column definitions to reflect the actual character set of the data, use the instructions in Section 10.10.2, "Converting 4.0 Character Columns to 4.1 Format".

10.10.1. 4.0 Character Sets and Corresponding 4.1 Character Set/Collation Pairs

ID	4.0 Character Set	4.1 Character Set	4.1 Collation
1	big5	big5	big5_chinese_ci
2	czech	latin2	latin2_czech_ci
3	dec8	dec8	dec8_swedish_ci
4	dos	cp850	cp850_general_ci
5	german1	latin1	latin1_german1_ci
6	hp8	hp8	hp8_english_ci
7	koi8_ru	koi8r	koi8r_general_ci
8	latin1	latin1	latin1_swedish_ci
9	latin2	latin2	latin2_general_ci
10	swe7	swe7	swe7_swedish_ci
11	usa7	ascii	ascii_general_ci
12	ujis	ujis	ujis_japanese_ci
13	sjis	sjis	sjis_japanese_ci
14	cp1251	cp1251	cp1251_bulgarian_ci
15	danish	latin1	latin1_danish_ci
16	hebrew	hebrew	hebrew_general_ci
17	win1251	(removed)	(removed)
18	tis620	tis620	tis620_thai_ci
19	euc_kr	euckr	euckr_korean_ci
20	estonia	latin7	latin7_estonian_ci
21	hungarian	latin2	latin2_hungarian_ci
22	koi8_ukr	koi8u	koi8u_ukrainian_ci
23	win1251ukr	cp1251	cp1251_ukrainian_ci
24	gb2312	gb2312	gb2312_chinese_ci
25	greek	greek	greek_general_ci
26	win1250	cp1250	cp1250_general_ci

27	croat	latin2	latin2_croatian_ci
28	gbk	gbk	gbk_chinese_ci
29	cp1257	cp1257	cp1257_lithuanian_ci
30	latin5	latin5	latin5_turkish_ci
31	latin1_de	latin1	latin1_german2_ci

10.10.2. Converting 4.0 Character Columns to 4.1 Format

Normally, the server runs using the latin1 character set by default. If you have been storing column data that actually is in some other character set that the 4.1 server supports directly, you can convert the column. However, you should avoid trying to convert directly from latin1 to the "real" character set. This may result in data loss. Instead, convert the column to a binary column type, and then from the binary type to a non-binary type with the desired character set. Conversion to and from binary involves no attempt at character value conversion and preserves your data intact. For example, suppose that you have a 4.0 table with three columns that are used to store values represented in latin1, latin2, and utf8:

```
CREATE TABLE t
(
    latin1_col CHAR(50),
    latin2_col CHAR(100),
    utf8_col CHAR(150)
);
```

After upgrading to MySQL 4.1, you want to convert this table to leave latin1_col alone but change the latin2_col and utf8_col columns to have character sets of latin2 and utf8. First, back up your table, then convert the columns as follows:

```
ALTER TABLE t MODIFY latin2_col BINARY(100);
ALTER TABLE t MODIFY utf8_col BINARY(150);
ALTER TABLE t MODIFY latin2_col CHAR(100) CHARACTER SET latin2;
ALTER TABLE t MODIFY utf8_col CHAR(150) CHARACTER SET utf8;
```

The first two statements ``remove'' the character set information from the latin2_col and utf8_col columns. The second two statements assign the proper character sets to the two columns.

If you like, you can combine the to-binary conversions and from-binary conversions into single statements:

```
ALTER TABLE t
    MODIFY latin2_col BINARY(100),
    MODIFY utf8_col BINARY(150);
ALTER TABLE t
    MODIFY latin2_col CHAR(100) CHARACTER SET latin2,
    MODIFY utf8_col CHAR(150) CHARACTER SET utf8;
```

If you specified attributes when creating a column initially, you should also specify them when altering the table with ALTER TABLE. For example, if you specified NOT NULL and an explicit DEFAULT value, you should also provide them in the ALTER TABLE statement. Otherwise, the resulting column definition will not include those attributes.

10.11. Character Sets and Collations That

MySQL Supports

Here is an annotated list of character sets and collations that MySQL supports. Because options and installation settings differ, some sites might not have all items listed, and some sites might have items not listed.

MySQL supports 70+ collations for 30+ character sets. The character sets and their default collations are displayed by the `SHOW CHARACTER SET` statement. (The output actually includes another column that is not shown so that the example fits better on the page.)

```
mysql> SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+
| Charset  | Description                 | Default collation   |
+----------+-----------------------------+---------------------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |
| dec8     | DEC West European           | dec8_swedish_ci     |
| cp850    | DOS West European           | cp850_general_ci    |
| hp8      | HP West European            | hp8_english_ci      |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |
| latin1   | ISO 8859-1 West European    | latin1_swedish_ci   |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |
| swe7     | 7bit Swedish                | swe7_swedish_ci     |
| ascii    | US ASCII                    | ascii_general_ci    |
| ujis     | EUC-JP Japanese             | ujis_japanese_ci    |
| sjis     | Shift-JIS Japanese          | sjis_japanese_ci    |
| hebrew   | ISO 8859-8 Hebrew           | hebrew_general_ci   |
| tis620   | TIS620 Thai                 | tis620_thai_ci      |
| euckr    | EUC-KR Korean               | euckr_korean_ci     |
| koi8u    | KOI8-U Ukrainian            | koi8u_general_ci    |
| gb2312   | GB2312 Simplified Chinese   | gb2312_chinese_ci   |
| greek    | ISO 8859-7 Greek            | greek_general_ci    |
| cp1250   | Windows Central European    | cp1250_general_ci   |
| gbk      | GBK Simplified Chinese      | gbk_chinese_ci      |
| latin5   | ISO 8859-9 Turkish          | latin5_turkish_ci   |
| armscii8 | ARMSCII-8 Armenian          | armscii8_general_ci |
| utf8     | UTF-8 Unicode               | utf8_general_ci     |
| ucs2     | UCS-2 Unicode               | ucs2_general_ci     |
| cp866    | DOS Russian                 | cp866_general_ci    |
| keybcs2  | DOS Kamenicky Czech-Slovak  | keybcs2_general_ci  |
| macce    | Mac Central European        | macce_general_ci    |
| macroman | Mac West European           | macroman_general_ci |
| cp852    | DOS Central European        | cp852_general_ci    |
| latin7   | ISO 8859-13 Baltic          | latin7_general_ci   |
| cp1251   | Windows Cyrillic            | cp1251_general_ci   |
| cp1256   | Windows Arabic              | cp1256_general_ci   |
| cp1257   | Windows Baltic              | cp1257_general_ci   |
| binary   | Binary pseudo charset       | binary              |
| geostd8  | GEOSTD8 Georgian            | geostd8_general_ci  |
| cp932    | SJIS for Windows Japanese   | cp932_japanese_ci   |
| eucjpms  | UJIS for Windows Japanese   | eucjpms_japanese_ci |
+----------+-----------------------------+---------------------+
```

10.11.1. Unicode Character Sets

MySQL has two Unicode character sets. You can store texts in about 650 languages using these character sets. We have added several collations for these two new sets, with more to come.

- `ucs2` (UCS-2 Unicode) collations:

  - `ucs2_bin`
  - `ucs2_czech_ci`
  - `ucs2_danish_ci`
  - `ucs2_estonian_ci`
  - `ucs2_general_ci` (default)
  - `ucs2_icelandic_ci`
  - `ucs2_latvian_ci`
  - `ucs2_lithuanian_ci`
  - `ucs2_persian_ci`
  - `ucs2_polish_ci`
  - `ucs2_roman_ci`
  - `ucs2_romanian_ci`
  - `ucs2_slovak_ci`
  - `ucs2_slovenian_ci`
  - `ucs2_spanish2_ci`
  - `ucs2_spanish_ci`
  - `ucs2_swedish_ci`
  - `ucs2_turkish_ci`
  - `ucs2_unicode_ci`
- `utf8` (UTF-8 Unicode) collations:
  - `utf8_bin`
  - `utf8_czech_ci`
  - `utf8_danish_ci`
  - `utf8_estonian_ci`
  - `utf8_general_ci` (default)
  - `utf8_icelandic_ci`
  - `utf8_latvian_ci`
  - `utf8_lithuanian_ci`
  - `utf8_persian_ci`
  - `utf8_polish_ci`

- utf8_roman_ci
- utf8_romanian_ci
- utf8_slovak_ci
- utf8_slovenian_ci
- utf8_spanish2_ci
- utf8_spanish_ci
- utf8_swedish_ci
- utf8_turkish_ci
- utf8_unicode_ci

The utf8_unicode_ci collation is implemented according to the Unicode Collation Algorithm (UCA) described at http://www.unicode.org/reports/tr10/. The collation uses the default UCA weight keys: http://www.unicode.org/Public/UCA/latest/allkeys.txt. (The following discussion uses utf8_unicode_ci, but it is also true for ucs2_unicode_ci.)

Currently, the utf8_unicode_ci collation has only partial support for the Unicode Collation Algorithm. Some characters are not supported yet. Also, combining marks are not fully supported. This affects primarily Vietnamese and some minority languages in Russia such as Udmurt, Tatar, Bashkir, and Mari.

The most significant feature in utf8_unicode_ci is that it supports expansions, that is, when one character compares equal to several characters. For example, in German and some other languages 'ß' is equal to 'ss'.

utf8_general_ci is a legacy collation that does not support expansions. It can only compare characters one to one. This means that comparisons for the utf8_general_ci collation are faster, but slightly less correct, than comparisons for utf8_unicode_ci).

For example, the following equalities hold in both utf8_general_ci and utf8_unicode_ci:

```
Ä = A
Ö = O
Ü = U
```

A difference between the collations is that this is true for utf8_general_ci:

```
ß = s
```

Whereas this is true for utf8_unicode_ci:

```
ß = ss
```

Language-specific collations for the utf8 character set are implemented only if the default order provided by utf8_unicode_ci does not work for a language well. For example, utf8_unicode_ci works fine for German and French, so there is no need to create special utf8 collations for these two languages.

utf8_general_ci is fine for both German and French, too, with an exception that 'ß' is equal to 's', not to 'ss'. If that is acceptable for your application, use utf8_general_ci because it is faster. Otherwise, use utf8_unicode_ci because it is ``smarter."

utf8_swedish_ci, like other utf8 language-specific collations, is derived from utf8_unicode_ci with additional language rules. For example, in Swedish, the following relationship holds, which is not something expected by a German or French speaker:

```
Ü = Y < Ö
```

The utf8_spanish_ci and utf8_spanish2_ci collations correspond to modern Spanish and traditional Spanish, respectively. In both collations, 'ñ' (n-tilde) is a separate letter between 'n' and 'o'. In addition, for traditional Spanish, 'ch' is a separate letter between 'c' and d, and 'll' is a separate letter between 'l' and 'm'

10.11.2. West European Character Sets

West European Character Sets cover most West European languages, such as French, Spanish, Catalan, Basque, Portuguese, Italian, Albanian, Dutch, German, Danish, Swedish, Norwegian, Finnish, Faroese, Icelandic, Irish, Scottish, and English.

- ascii (US ASCII) collations:
  - ascii_bin
  - ascii_general_ci (default)
- cp850 (DOS West European) collations:
  - cp850_bin
  - cp850_general_ci (default)
- dec8 (DEC West European) collations:
  - dec8_bin
  - dec8_swedish_ci (default)
- hp8 (HP West European) collations:
  - hp8_bin
  - hp8_english_ci (default)
- latin1 (ISO 8859-1 West European) collations:
  - latin1_bin
  - latin1_danish_ci
  - latin1_general_ci
  - latin1_general_cs
  - latin1_german1_ci

- latin1_german2_ci
- latin1_spanish_ci
- latin1_swedish_ci (default)

The latin1 is the default character set. The latin1_swedish_ci collation is the default that probably is used by the majority of MySQL customers. It is constantly stated that this is based on the Swedish/Finnish collation rules, but you can find Swedes and Finns who disagree with that statement.

The latin1_german1_ci and latin1_german2_ci collations are based on the DIN-1 and DIN-2 standards, where DIN stands for Deutsches Institut für Normung (that is, the German answer to ANSI). DIN-1 is called the dictionary collation and DIN-2 is called the phone-book collation.

- latin1_german1_ci (dictionary) rules:

```
Ä = A
Ö = O
Ü = U
ß = s
```

- latin1_german2_ci (phone-book) rules:

```
Ä = AE
Ö = OE
Ü = UE
ß = ss
```

In the latin1_spanish_ci collation, 'ñ' (n-tilde) is a separate letter between 'n' and 'o'.

- macroman (Mac West European) collations:
  - macroman_bin
  - macroman_general_ci (default)
- swe7 (7bit Swedish) collations:
  - swe7_bin
  - swe7_swedish_ci (default)

10.11.3. Central European Character Sets

We have some support for character sets used in the Czech Republic, Slovakia, Hungary, Romania, Slovenia, Croatia, and Poland.

- cp1250 (Windows Central European) collations:
  - cp1250_bin
  - cp1250_czech_cs
  - cp1250_general_ci (default)

- `cp852` (DOS Central European) collations:
  - `cp852_bin`
  - `cp852_general_ci` (default)
- `keybcs2` (DOS Kamenicky Czech-Slovak) collations:
  - `keybcs2_bin`
  - `keybcs2_general_ci` (default)
- `latin2` (ISO 8859-2 Central European) collations:
  - `latin2_bin`
  - `latin2_croatian_ci`
  - `latin2_czech_cs`
  - `latin2_general_ci` (default)
  - `latin2_hungarian_ci`
- `macce` (Mac Central European) collations:
  - `macce_bin`
  - `macce_general_ci` (default)

10.11.4. South European and Middle East Character Sets

- `armscii8` (ARMSCII-8 Armenian) collations:
  - `armscii8_bin`
  - `armscii8_general_ci` (default)
- `cp1256` (Windows Arabic) collations:
  - `cp1256_bin`
  - `cp1256_general_ci` (default)
- `geostd8` (GEOSTD8 Georgian) collations:
  - `geostd8_bin`
  - `geostd8_general_ci` (default)
- `greek` (ISO 8859-7 Greek) collations:
  - `greek_bin`
  - `greek_general_ci` (default)
- `hebrew` (ISO 8859-8 Hebrew) collations:

  - `hebrew_bin`
  - `hebrew_general_ci` (default)
- `latin5` (ISO 8859-9 Turkish) collations:
  - `latin5_bin`
  - `latin5_turkish_ci` (default)

10.11.5. Baltic Character Sets

The Baltic character sets cover Estonian, Latvian, and Lithuanian languages. There are two Baltic character sets currently supported:

- `cp1257` (Windows Baltic) collations:
  - `cp1257_bin`
  - `cp1257_general_ci` (default)
  - `cp1257_lithuanian_ci`
- `latin7` (ISO 8859-13 Baltic) collations:
  - `latin7_bin`
  - `latin7_estonian_cs`
  - `latin7_general_ci` (default)
  - `latin7_general_cs`

10.11.6. Cyrillic Character Sets

Here are the Cyrillic character sets and collations for use with Belarusian, Bulgarian, Russian, and Ukrainian languages.

- `cp1251` (Windows Cyrillic) collations:
  - `cp1251_bin`
  - `cp1251_bulgarian_ci`
  - `cp1251_general_ci` (default)
  - `cp1251_general_cs`
  - `cp1251_ukrainian_ci`
- `cp866` (DOS Russian) collations:
  - `cp866_bin`

  - `cp866_general_ci` (default)
- `koi8r` (KOI8-R Relcom Russian) collations:
  - `koi8r_bin`
  - `koi8r_general_ci` (default)
- `koi8u` (KOI8-U Ukrainian) collations:
  - `koi8u_bin`
  - `koi8u_general_ci` (default)

10.11.7. Asian Character Sets

The Asian character sets that we support include Chinese, Japanese, Korean, and Thai. These can be complicated. For example, the Chinese sets must allow for thousands of different characters.

- `big5` (Big5 Traditional Chinese) collations:
  - `big5_bin`
  - `big5_chinese_ci` (default)
- `cp932` (SJIS for Windows Japanese) collations:
  - `cp932_bin`
  - `cp932_japanese_ci` (default)
- `eucjpms` (UJIS for Windows Japanese) collations:
  - `eucjpms_bin`
  - `eucjpms_japanese_ci` (default)
- `euckr` (EUC-KR Korean) collations:
  - `euckr_bin`
  - `euckr_korean_ci` (default)
- `gb2312` (GB2312 Simplified Chinese) collations:
  - `gb2312_bin`
  - `gb2312_chinese_ci` (default)
- `gbk` (GBK Simplified Chinese) collations:
  - `gbk_bin`
  - `gbk_chinese_ci` (default)
- `sjis` (Shift-JIS Japanese) collations:

  - `sjis_bin`
  - `sjis_japanese_ci` (default)
- `tis620` (TIS620 Thai) collations:
  - `tis620_bin`
  - `tis620_thai_ci` (default)
- `ujis` (EUC-JP Japanese) collations:
  - `ujis_bin`
  - `ujis_japanese_ci` (default)

Chapter 11. Column Types

MySQL supports a number of column types in several categories: numeric types, date and time types, and string (character) types. This chapter first gives an overview of these column types, and then provides a more detailed description of the properties of the types in each category, and a summary of the column type storage requirements. The overview is intentionally brief. The more detailed descriptions should be consulted for additional information about particular column types, such as the allowable formats in which you can specify values.

MySQL versions 4.1 and up support extensions for handing spatial data. Information about spatial types is provided in Chapter 18, *Spatial Extensions in MySQL*.

Several of the column type descriptions use these conventions:

- *M*

  Indicates the maximum display width. The maximum legal display width is 255.

- *D*

  Applies to floating-point and fixed-point types and indicates the number of digits following the decimal point. The maximum possible value is 30, but should be no greater than *M*#2.

-

  Square brackets ('[' and ']') indicate parts of type specifiers that are optional.

11.1. Column Type Overview

11.1.1. Overview of Numeric Types

A summary of the numeric column types follows. For additional information, see Section 11.2, “Numeric Types”. Column storage requirements are given in Section 11.5, “Column Type Storage Requirements”.

M indicates the maximum display width. The maximum legal display width is 255. Display width is unrelated to the storage size or range of values a type can contain, as described in Section 11.2, “Numeric Types”.

If you specify `ZEROFILL` for a numeric column, MySQL automatically adds the `UNSIGNED` attribute to the column.

`SERIAL` is an alias for `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT`.

`SERIAL DEFAULT VALUE` in the definition of an integer column is an alias for `NOT NULL AUTO_INCREMENT UNIQUE`.

Warning: You should be aware that when you use subtraction between integer values where one is of type `UNSIGNED`, the result is unsigned. See Section 12.7, “Cast Functions and Operators”.

- `BIT[(M)]`

  A bit-field type. *M* indicates the number of bits per value, from 1 to 64. The default is 1 if *M* is omit-

ted.

This data type was added in MySQL 5.0.3. Before 5.0.3, `BIT` is a synonym for `TINYINT(1)`.

- `TINYINT[(M)] [UNSIGNED] [ZEROFILL]`

  A very small integer. The signed range is `-128` to `127`. The unsigned range is `0` to `255`.

- `BOOL`, `BOOLEAN`

  These are synonyms for `TINYINT(1)`. The `BOOLEAN` synonym was added in MySQL 4.1.0. A value of zero is considered false. Non-zero values are considered true.

  In the future, full boolean type handling will be introduced in accordance with standard SQL.

- `SMALLINT[(M)] [UNSIGNED] [ZEROFILL]`

  A small integer. The signed range is `-32768` to `32767`. The unsigned range is `0` to `65535`.

- `MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]`

  A medium-size integer. The signed range is `-8388608` to `8388607`. The unsigned range is `0` to `16777215`.

- `INT[(M)] [UNSIGNED] [ZEROFILL]`

  A normal-size integer. The signed range is `-2147483648` to `2147483647`. The unsigned range is `0` to `4294967295`.

- `INTEGER[(M)] [UNSIGNED] [ZEROFILL]`

  This is a synonym for `INT`.

- `BIGINT[(M)] [UNSIGNED] [ZEROFILL]`

  A large integer. The signed range is `-9223372036854775808` to `9223372036854775807`. The unsigned range is `0` to `18446744073709551615`.

  Some things you should be aware of with respect to `BIGINT` columns:

  - All arithmetic is done using signed `BIGINT` or `DOUBLE` values, so you shouldn't use unsigned big integers larger than `9223372036854775807` (63 bits) except with bit functions! If you do that, some of the last digits in the result may be wrong because of rounding errors when converting a `BIGINT` value to a `DOUBLE`.

    MySQL 4.0 can handle `BIGINT` in the following cases:

  - When using integers to store big unsigned values in a `BIGINT` column.
  - In `MIN(col_name)` or `MAX(col_name)`, where `col_name` refers to a `BIGINT` column.
  - When using operators (`+`, `-`, `*`, and so on) where both operands are integers.
- You can always store an exact integer value in a `BIGINT` column by storing it using a string. In this case, MySQL performs a string-to-number conversion that involves no intermediate double-precision representation.
- The `-`, `+`, and `*` operators use `BIGINT` arithmetic when both operands are integer values. This means that if you multiply two big integers (or results from functions that return integers), you may get unexpected results when the result is larger than `9223372036854775807`.

- `FLOAT(p) [UNSIGNED] [ZEROFILL]`

  A floating-point number. `p` represents the precision. It can be from 0 to 24 for a single-precision floating-point number and from 25 to 53 for a double-precision floating-point number. These types are like the `FLOAT` and `DOUBLE` types described immediately following. `FLOAT(p)` has the same range as the corresponding `FLOAT` and `DOUBLE` types, but the display width and number of decimals are undefined.

  As of MySQL 3.23, this is a true floating-point value. In earlier MySQL versions, `FLOAT(p)` always has two decimals.

  This syntax is provided for ODBC compatibility.

  Using `FLOAT` might give you some unexpected problems because all calculations in MySQL are done with double precision. See Section A.5.7, “Solving Problems with No Matching Rows”.

- `FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]`

  A small (single-precision) floating-point number. Allowable values are `-3.402823466E+38` to `-1.175494351E-38`, `0`, and `1.175494351E-38` to `3.402823466E+38`. If `UNSIGNED` is specified, negative values are disallowed. `M` is the display width and `D` is the number of decimals. `FLOAT` without arguments or `FLOAT(p)` (where `p` is in the range from 0 to 24) stands for a single-precision floating-point number.

- `DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]`

  A normal-size (double-precision) floating-point number. Allowable values are `-1.7976931348623157E+308` to `-2.2250738585072014E-308`, `0`, and `2.2250738585072014E-308` to `1.7976931348623157E+308`. If `UNSIGNED` is specified, negative values are disallowed. `M` is the display width and `D` is the number of decimals. `DOUBLE` without arguments or `FLOAT(p)` (where `p` is in the range from 25 to 53) stands for a double-precision floating-point number.

- `DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL]`, `REAL[(M,D)] [UNSIGNED] [ZEROFILL]`

  These are synonyms for `DOUBLE`. Exception: If the server SQL mode includes the

REAL_AS_FLOAT option, REAL is a synonym for FLOAT rather than DOUBLE.

- DECIMAL[(*M*[,*D*])] [UNSIGNED] [ZEROFILL]

  An unpacked fixed-point number. Behaves like a CHAR column; ``unpacked'' means the number is stored as a string, using one character for each digit of the value. *M* is the total number of digits and *D* is the number of decimals. The decimal point and (for negative numbers) the '-' sign are not counted in *M*, although space for them is reserved. If *D* is 0, values have no decimal point or fractional part. The maximum range of DECIMAL values is the same as for DOUBLE, but the actual range for a given DECIMAL column may be constrained by the choice of *M* and *D*. If UNSIGNED is specified, negative values are disallowed.

  If *D* is omitted, the default is 0. If *M* is omitted, the default is 10.

  Prior to MySQL 3.23, the *M* argument must be large enough to include the space needed for the sign and the decimal point.

- DEC[(*M*[,*D*])] [UNSIGNED] [ZEROFILL] , NUMERIC[(*M*[,*D*])] [UNSIGNED] [ZEROFILL] , FIXED[(*M*[,*D*])] [UNSIGNED] [ZEROFILL]

  These are synonyms for DECIMAL. The FIXED synonym was added in MySQL 4.1.0 for compatibility with other servers.

11.1.2. Overview of Date and Time Types

A summary of the temporal column types follows. For additional information, see Section 11.3, “Date and Time Types”. Column storage requirements are given in Section 11.5, “Column Type Storage Requirements”.

- DATE

  A date. The supported range is '1000-01-01' to '9999-12-31'. MySQL displays DATE values in 'YYYY-MM-DD' format, but allows you to assign values to DATE columns using either strings or numbers.

- DATETIME

  A date and time combination. The supported range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. MySQL displays DATETIME values in 'YYYY-MM-DD HH:MM:SS' format, but allows you to assign values to DATETIME columns using either strings or numbers.

- TIMESTAMP[(*M*)]

  A timestamp. The range is '1970-01-01 00:00:00' to partway through the year 2037.

  A TIMESTAMP column is useful for recording the date and time of an INSERT or UPDATE operation. The first TIMESTAMP column in a table is automatically set to the date and time of the most

recent operation if you don't assign it a value yourself. You can also set any TIMESTAMP column to the current date and time by assigning it a NULL value.

From MySQL 4.1 on, TIMESTAMP is returned as a string with the format 'YYYY-MM-DD HH:MM:SS'. If you want to obtain the value as a number, you should add +0 to the timestamp column. Different timestamp display widths are not supported.

In MySQL 4.0 and earlier, TIMESTAMP values are displayed in YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD, or YYMMDD format, depending on whether *M* is 14 (or missing), 12, 8, or 6, but allows you to assign values to TIMESTAMP columns using either strings or numbers. The *M* argument affects only how a TIMESTAMP column is displayed, not storage. Its values always are stored using four bytes each. From MySQL 4.0.12, the --new option can be used to make the server behave as in MySQL 4.1.

Note that TIMESTAMP(*M*) columns where *M* is 8 or 14 are reported to be numbers, whereas other TIMESTAMP(*M*) columns are reported to be strings. This is just to ensure that you can reliably dump and restore the table with these types.

- TIME

  A time. The range is '-838:59:59' to '838:59:59'. MySQL displays TIME values in 'HH:MM:SS' format, but allows you to assign values to TIME columns using either strings or numbers.

- YEAR[(2|4)]

  A year in two-digit or four-digit format. The default is four-digit format. In four-digit format, the allowable values are 1901 to 2155, and 0000. In two-digit format, the allowable values are 70 to 69, representing years from 1970 to 2069. MySQL displays YEAR values in YYYY format, but allows you to assign values to YEAR columns using either strings or numbers. The YEAR type is unavailable prior to MySQL 3.22.

11.1.3. Overview of String Types

A summary of the string column types follows. For additional information, see Section 11.4, “String Types”. Column storage requirements are given in Section 11.5, “Column Type Storage Requirements”.

In some cases, MySQL may change a string column to a type different from that given in a CREATE TABLE or ALTER TABLE statement. See Section 13.2.6.1, “Silent Column Specification Changes”.

As of MySQL 4.1, several changes affect string data types:

- Column definitions for many string data types can include a CHARACTER SET attribute to specify the character set and, optionally, a collation. This applies to CHAR, VARCHAR, the TEXT types, ENUM, and SET. For example:

  ```
  CREATE TABLE t
  (
      c1 CHAR(20) CHARACTER SET utf8,
      c2 CHAR(20) CHARACTER SET latin1 COLLATE latin1_bin
  );
  ```

This table definition creates a column named `c1` that has a character set of `utf8` with the default collation for that character set, and a column named `c2` that has a character set of `latin1` and the binary collation for the character set. The binary collation is not case sensitive.

- MySQL 4.1, MySQL interprets length specifications in character column definitions in character units. Earlier versions interpret lengths in byte units.
- For `CHAR`, `VARCHAR`, and the `TEXT` types, the `BINARY` attribute causes the column to be assigned the binary collation of the column character set. For earlier versions, `BINARY` causes `CHAR` and `VARCHAR` to be treated as binary strings and is disallowed for the `TEXT` types.
- Character column sorting and comparison are based on the character set assigned to the column. For earalier versions, sorting and comparison are based on the collation of the server character set. For `CHAR` and `VARCHAR` columns, you can declare the column with the `BINARY` attribute to cause sorting and comparison to use the underlying character code values rather then a lexical ordering.

For more details about character set support in MySQL 4.1 and up, see Chapter 10, *Character Set Support*.

- `[NATIONAL] CHAR(M) [BINARY | ASCII | UNICODE]`

  A fixed-length string that is always right-padded with spaces to the specified length when stored. *M* represents the column length. The range of *M* is 0 to 255 characters (1 to 255 prior to MySQL 3.23).

  Note: Trailing spaces are removed when `CHAR` values are retrieved.

  From MySQL 4.1.0 to 5.0.2, a `CHAR` column with a length specification greater than 255 is converted to the smallest `TEXT` type that can hold values of the given length. For example, `CHAR(500)` is converted to `TEXT`, and `CHAR(200000)` is converted to `MEDIUMTEXT`. This is a compatibility feature. However, this conversion causes the column to become a variable-length column, and also affects trailing-space removal.

  `CHAR` is shorthand for `CHARACTER`. `NATIONAL CHAR` (or its equivalent short form, `NCHAR`) is the standard SQL way to define that a `CHAR` column should use the default character set. This is the default in MySQL.

  As of MySQL 4.1.2, the `BINARY` attribute is shorthand for specifying the binary collation of the column character set. Sorting and comparison is based on numeric character values. Before 4.1.2, `BINARY` attribute causes the column to be treated as a binary string. Sorting and comparison is based on numeric byte values.

  From MySQL 4.1.0 on, column type `CHAR BYTE` is an alias for `CHAR BINARY`. This is a compatibility feature.

  From MySQL 4.1.0 on, the `ASCII` attribute can be specified for `CHAR`. It assigns the `latin1` character set.

  From MySQL 4.1.1 on, the `UNICODE` attribute can be specified for `CHAR`. It assigns the `ucs2` character set.

  MySQL allows you to create a column of type `CHAR(0)`. This is mainly useful when you have to be compliant with some old applications that depend on the existence of a column but that do not actually use the value. This is also quite nice when you need a column that can take only two values: A `CHAR(0)` column that is not defined as `NOT NULL` occupies only one bit and can take only the values `NULL` and `''` (the empty string).

- `CHAR`

  This is a synonym for `CHAR(1)`.

- `[NATIONAL] VARCHAR(M) [BINARY]`

  A variable-length string. *M* represents the maximum column length. The range of *M* is 1 to 255 before MySQL 4.0.2, 0 to 255 as of MySQL 4.0.2, and 0 to 65,535 as of MySQL 5.0.3. (The maximum actual length of a `VARCHAR` in MySQL 5.0 is determined by the maximum row size and the character set you use. The maximum effective length is 65,532 bytes.)

  Note: Before 5.0.3, trailing spaces were removed when `VARCHAR` values were stored, which differs from the standard SQL specification.

  From MySQL 4.1.0 to 5.0.2, a `VARCHAR` column with a length specification greater than 255 is converted to the smallest `TEXT` type that can hold values of the given length. For example, `VARCHAR(500)` is converted to `TEXT`, and `VARCHAR(200000)` is converted to `MEDIUMTEXT`. This is a compatibility feature. However, this conversion affects trailing-space removal.

  `VARCHAR` is shorthand for `CHARACTER VARYING`.

  As of MySQL 4.1.2, the `BINARY` attribute is shorthand for specifying the binary collation of the column character set. Sorting and comparison is based on numeric character values. Before 4.1.2, `BINARY` attribute causes the column to be treated as a binary string. Sorting and comparison is based on numeric byte values.

  Starting from MySQL 5.0.3, `VARCHAR` is stored with a one-byte or two-byte length prefix + data. The length prefix is two bytes if the `VARCHAR` column is declared with a length greater than 255.

- `BINARY(M)`

  The `BINARY` type is similar to the `CHAR` type, but stores binary byte strings rather than non-binary character strings.

  This type was added in MySQL 4.1.2.

- `VARBINARY(M)`

  The `VARBINARY` type is similar to the `VARCHAR` type, but stores binary byte strings rather than non-binary character strings.

  This type was added in MySQL 4.1.2.

- `TINYBLOB`

  A `BLOB` column with a maximum length of 255 (2^8 # 1) bytes.

- `TINYTEXT`

A TEXT column with a maximum length of 255 (2^8 # 1) characters.

- BLOB

  A BLOB column with a maximum length of 65,535 (2^16 # 1) bytes.

- TEXT

  A TEXT column with a maximum length of 65,535 (2^16 # 1) characters.

- MEDIUMBLOB

  A BLOB column with a maximum length of 16,777,215 (2^24 # 1) bytes.

- MEDIUMTEXT

  A TEXT column with a maximum length of 16,777,215 (2^24 # 1) characters.

- LONGBLOB

  A BLOB column with a maximum length of 4,294,967,295 or 4GB (2^32 # 1) bytes. Up to MySQL 3.23, the client/server protocol and MyISAM tables had a limit of 16MB per communication packet / table row. From MySQL 4.0, the maximum allowed length of LONGBLOB columns depends on the configured maximum packet size in the client/server protocol and available memory.

- LONGTEXT

  A TEXT column with a maximum length of 4,294,967,295 or 4GB (2^32 # 1) characters. Up to MySQL 3.23, the client/server protocol and MyISAM tables had a limit of 16MB per communication packet / table row. From MySQL 4.0, the maximum allowed length of LONGTEXT columns depends on the configured maximum packet size in the client/server protocol and available memory.

- ENUM('*value1*','*value2*',...)

  An enumeration. A string object that can have only one value, chosen from the list of values '*value1*', '*value2*', ..., NULL or the special '' error value. An ENUM column can have a maximum of 65,535 distinct values. ENUM values are represented internally as integers.

- SET('*value1*','*value2*',...)

  A set. A string object that can have zero or more values, each of which must be chosen from the list of values '*value1*', '*value2*', ... A SET column can have a maximum of 64 members. SET values are represented internally as integers.

11.2. Numeric Types

MySQL supports all of the standard SQL numeric data types. These types include the exact numeric data types (`INTEGER`, `SMALLINT`, `DECIMAL`, and `NUMERIC`), as well as the approximate numeric data types (`FLOAT`, `REAL`, and `DOUBLE PRECISION`). The keyword `INT` is a synonym for `INTEGER`, and the keyword `DEC` is a synonym for `DECIMAL`.

As of MySQL 5.0.3, a `BIT` data type is available for storing bit-field values. (Before 5.0.3, MySQL interprets `BIT` as `TINYINT(1)`.)

As an extension to the SQL standard, MySQL also supports the integer types `TINYINT`, `MEDIUMINT`, and `BIGINT`. The following table shows the required storage and range for each of the integer types.

Type	**Bytes**	**Minimum Value**	**Maximum Value**
		(Signed/Unsigned)	**(Signed/Unsigned)**
`TINYINT`	1	`-128`	`127`
		`0`	`255`
`SMALLINT`	2	`-32768`	`32767`
		`0`	`65535`
`MEDIUMINT`	3	`-8388608`	`8388607`
		`0`	`16777215`
`INT`	4	`-2147483648`	`2147483647`
		`0`	`4294967295`
`BIGINT`	8	`-9223372036854775808`	`9223372036854775807`
		`0`	`18446744073709551615`

Another extension is supported by MySQL for optionally specifying the display width of an integer value in parentheses following the base keyword for the type (for example, `INT(4)`). This optional display width specification is used to left-pad the display of values having a width less than the width specified for the column.

The display width does not constrain the range of values that can be stored in the column, nor the number of digits that are displayed for values having a width exceeding that specified for the column.

When used in conjunction with the optional extension attribute `ZEROFILL`, the default padding of spaces is replaced with zeros. For example, for a column declared as `INT(5) ZEROFILL`, a value of `4` is retrieved as `00004`. Note that if you store larger values than the display width in an integer column, you may experience problems when MySQL generates temporary tables for some complicated joins, because in these cases MySQL trusts that the data did fit into the original column width.

All integer types can have an optional (non-standard) attribute `UNSIGNED`. Unsigned values can be used when you want to allow only non-negative numbers in a column and you need a bigger upper numeric range for the column.

As of MySQL 4.0.2, floating-point and fixed-point types also can be `UNSIGNED`. As with integer types, this attribute prevents negative values from being stored in the column. However, unlike the integer types, the upper range of column values remains the same.

If you specify `ZEROFILL` for a numeric column, MySQL automatically adds the `UNSIGNED` attribute to the column.

For floating-point column types, MySQL uses four bytes for single-precision values and eight bytes for double-precision values.

The FLOAT type is used to represent approximate numeric data types. The SQL standard allows an optional specification of the precision (but not the range of the exponent) in bits following the keyword FLOAT in parentheses. The MySQL implementation also supports this optional precision specification, but the precision value is used only to determine storage size. A precision from 0 to 23 results in four-byte single-precision FLOAT column. A precision from 24 to 53 results in eight-byte double-precision DOUBLE column.

When the keyword FLOAT is used for a column type without a precision specification, MySQL uses four bytes to store the values. MySQL also supports variant syntax with two numbers given in parentheses following the FLOAT keyword. The first number represents the display width and the second number specifies the number of digits to be stored and displayed following the decimal point (as with DECIMAL and NUMERIC). When MySQL is asked to store a number for such a column with more decimal digits following the decimal point than specified for the column, the value is rounded to eliminate the extra digits when the value is stored.

In standard SQL, the REAL and DOUBLE PRECISION types do not accept precision specifications. MySQL supports a variant syntax with two numbers given in parentheses following the type name. The first number represents the display width and the second number specifies the number of digits to be stored and displayed following the decimal point. As an extension to the SQL standard, MySQL recognizes DOUBLE as a synonym for the DOUBLE PRECISION type. In contrast with the standard's requirement that the precision for REAL be smaller than that used for DOUBLE PRECISION, MySQL implements both as eight-byte double-precision floating-point values (unless the server SQL mode includes the REAL_AS_FLOAT option).

For maximum portability, code requiring storage of approximate numeric data values should use FLOAT or DOUBLE PRECISION with no specification of precision or number of decimal points.

The DECIMAL and NUMERIC types are implemented as the same type by MySQL. They are used to store values for which it is important to preserve exact precision, for example with monetary data. When declaring a column of one of these types, the precision and scale can be (and usually is) specified; for example:

```
salary DECIMAL(5,2)
```

In this example, 5 is the precision and 2 is the scale. The precision represents the number of significant decimal digits that are stored for values, and the scale represents the number of digits that can be stored following the decimal point.

MySQL stores DECIMAL and NUMERIC values as strings, rather than as binary floating-point numbers, in order to preserve the decimal precision of those values. One character is used for each digit of the value, the decimal point (if the scale is greater than 0), and the '-' sign (for negative numbers). If the scale is 0, DECIMAL and NUMERIC values contain no decimal point or fractional part.

Standard SQL requires that the salary column be able to store any value with five digits and two decimals. In this case, therefore, the range of values that can be stored in the salary column is from -999.99 to 999.99. MySQL varies from this in two ways:

- On the positive end of the range, the column actually can store numbers up to 9999.99. For positive numbers, MySQL uses the byte reserved for the sign to extend the upper end of the range.
- DECIMAL columns in MySQL before 3.23 are stored differently and cannot represent all the values required by standard SQL. This is because for a type of DECIMAL(*M*,*D*), the value of *M* includes the bytes for the sign and the decimal point. The range of the salary column before MySQL 3.23 would be -9.99 to 99.99.

In standard SQL, the syntax DECIMAL(*M*) is equivalent to DECIMAL(*M*,0). Similarly, the syntax

DECIMAL is equivalent to DECIMAL(*M*,0), where the implementation is allowed to decide the value of *M*. As of MySQL 3.23.6, both of these variant forms of the DECIMAL and NUMERIC data types are supported. The default value of *M* is 10. Before 3.23.6, *M* and *D* both must be specified explicitly.

The maximum range of DECIMAL and NUMERIC values is the same as for DOUBLE, but the actual range for a given DECIMAL or NUMERIC column can be constrained by the precision or scale for a given column. When such a column is assigned a value with more digits following the decimal point than are allowed by the specified scale, the value is converted to that scale. (The precise behavior is operating system-specific, but generally the effect is truncation to the allowable number of digits.)

As of MySQL 5.0.3, the BIT data type can be used to store bit-field values. A type of BIT(*M*) allows for storage of *M*-bit values. *M* can range from 1 to 64.

To specify bit values, b'*value*' notation can be used. *value* is a binary value written using 0s and 1s. For example, b'111' and b'100000000' represent 7 and 128, respectively. See Section 9.1.5, “Bit-Field Values”.

When asked to store a value in a numeric column that is outside the column type's allowable range, MySQL clips the value to the appropriate endpoint of the range and stores the resulting value instead.

For example, the range of an INT column is -2147483648 to 2147483647. If you try to insert -9999999999 into an INT column, MySQL clips the value to the lower endpoint of the range and stores -2147483648 instead. Similarly, if you try to insert 9999999999, MySQL clips the value to the upper endpoint of the range and stores 2147483647 instead.

If the INT column is UNSIGNED, the size of the column's range is the same but its endpoints shift up to 0 and 4294967295. If you try to store -9999999999 and 9999999999, the values stored in the column are 0 and 4294967296.

When a floating-point or fixed-point column is assigned a value that exceeds the range implied by the specified (or default) precision and scale, MySQL stores the value representing the corresponding end point of that range.

Conversions that occur due to clipping are reported as ``warnings'' for ALTER TABLE, LOAD DATA INFILE, UPDATE, and multiple-row INSERT statements.

11.3. Date and Time Types

The date and time types for representing temporal values are DATETIME, DATE, TIMESTAMP, TIME, and YEAR. Each temporal type has a range of legal values, as well as a ``zero'' value that is used when you specify an illegal value that MySQL cannot represent. The TIMESTAMP type has special automatic updating behavior, described later on.

Starting from MySQL 5.0.2, MySQL gives warnings/errors if you try to insert an illegal date. You can get MySQL to accept certain dates, such as '1999-11-31', by using the ALLOW_INVALID_DATES SQL mode. (Before 5.0.2, this mode was the default behavior for MySQL). This is useful when you want to store the ``possibly wrong'' value the user has specified (for example, in a web form) in the database for future processing. Under this mode, MySQL verifies only that the month is in the range from 0 to 12 and that the day is in the range from 0 to 31. These ranges are defined to include zero because MySQL allows you to store dates where the day or month and day are zero in a DATE or DATETIME column. This is extremely useful for applications that need to store a birthdate for which you don't know the exact date. In this case, you simply store the date as '1999-00-00' or '1999-01-00'. If you store dates such as these, you should not expect to get correct results for functions such as DATE_SUB() or DATE_ADD that require complete dates. (If you don't want to allow zero in dates, you can use the NO_ZERO_IN_DATE SQL mode).

MySQL also allows you to store '0000-00-00' as a ``dummy date'' (if you are not using the NO_ZERO_DATE SQL mode). This is in some cases is more convenient (and uses less space in data and

index) than using NULL values.

By setting the sql_mode system variable to the appropriate mode values, You can more exactly what kind of dates you want MySQL to support. See Section 5.2.2, “The Server SQL Mode”.

Here are some general considerations to keep in mind when working with date and time types:

- MySQL retrieves values for a given date or time type in a standard output format, but it attempts to interpret a variety of formats for input values that you supply (for example, when you specify a value to be assigned to or compared to a date or time type). Only the formats described in the following sections are supported. It is expected that you supply legal values, and unpredictable results may occur if you use values in other formats.
- Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using the following rules:
  - Year values in the range 00-69 are converted to 2000-2069.
  - Year values in the range 70-99 are converted to 1970-1999.
- Although MySQL tries to interpret values in several formats, dates always must be given in year-month-day order (for example, '98-09-04'), rather than in the month-day-year or day-month-year orders commonly used elsewhere (for example, '09-04-98', '04-09-98').
- MySQL automatically converts a date or time type value to a number if the value is used in a numeric context and vice versa.
- When MySQL encounters a value for a date or time type that is out of range or otherwise illegal for the type (as described at the beginning of this section), it converts the value to the ``zero'' value for that type. The exception is that out-of-range TIME values are clipped to the appropriate endpoint of the TIME range.

  The following table shows the format of the ``zero'' value for each type. Note that the use of these values produces warnings if the NO_ZERO_DATE SQL mode is enabled.

Column Type	``Zero'' Value
DATETIME	'0000-00-00 00:00:00'
DATE	'0000-00-00'
TIMESTAMP	00000000000000
TIME	'00:00:00'
YEAR	0000

- The ``zero'' values are special, but you can store or refer to them explicitly using the values shown in the table. You can also do this using the values '0' or 0, which are easier to write.
- ``Zero'' date or time values used through MyODBC are converted automatically to NULL in MyODBC 2.50.12 and above, because ODBC can't handle such values.

11.3.1. The DATETIME, DATE, and TIMESTAMP Types

The DATETIME, DATE, and TIMESTAMP types are related. This section describes their characteristics, how they are similar, and how they differ.

The DATETIME type is used when you need values that contain both date and time information.

MySQL retrieves and displays DATETIME values in 'YYYY-MM-DD HH:MM:SS' format. The supported range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59'. (``Supported'' means that although earlier values might work, there is no guarantee)

The DATE type is used when you need only a date value, without a time part. MySQL retrieves and displays DATE values in 'YYYY-MM-DD' format. The supported range is '1000-01-01' to '9999-12-31'.

The TIMESTAMP column type has varying properties, depending on the MySQL version and the SQL mode the server is running in. These properties are described later in this section.

You can specify DATETIME, DATE, and TIMESTAMP values using any of a common set of formats:

- As a string in either 'YYYY-MM-DD HH:MM:SS' or 'YY-MM-DD HH:MM:SS' format. A ``relaxed'' syntax is allowed: Any punctuation character may be used as the delimiter between date parts or time parts. For example, '98-12-31 11:30:45', '98.12.31 11+30+45', '98/12/31 11*30*45', and '98@12@31 11^30^45' are equivalent.
- As a string in either 'YYYY-MM-DD' or 'YY-MM-DD' format. A ``relaxed'' syntax is allowed here, too. For example, '98-12-31', '98.12.31', '98/12/31', and '98@12@31' are equivalent.
- As a string with no delimiters in either 'YYYYMMDDHHMMSS' or 'YYMMDDHHMMSS' format, provided that the string makes sense as a date. For example, '19970523091528' and '970523091528' are interpreted as '1997-05-23 09:15:28', but '971122129015' is illegal (it has a nonsensical minute part) and becomes '0000-00-00 00:00:00'.
- As a string with no delimiters in either 'YYYYMMDD' or 'YYMMDD' format, provided that the string makes sense as a date. For example, '19970523' and '970523' are interpreted as '1997-05-23', but '971332' is illegal (it has nonsensical month and day parts) and becomes '0000-00-00'.
- As a number in either YYYYMMDDHHMMSS or YYMMDDHHMMSS format, provided that the number makes sense as a date. For example, 19830905132800 and 830905132800 are interpreted as '1983-09-05 13:28:00'.
- As a number in either YYYYMMDD or YYMMDD format, provided that the number makes sense as a date. For example, 19830905 and 830905 are interpreted as '1983-09-05'.
- As the result of a function that returns a value that is acceptable in a DATETIME, DATE, or TIMESTAMP context, such as NOW() or CURRENT_DATE.

Illegal DATETIME, DATE, or TIMESTAMP values are converted to the ``zero'' value of the appropriate type ('0000-00-00 00:00:00', '0000-00-00', or 00000000000000).

For values specified as strings that include date part delimiters, it is not necessary to specify two digits for month or day values that are less than 10. '1979-6-9' is the same as '1979-06-09'. Similarly, for values specified as strings that include time part delimiters, it is not necessary to specify two digits for hour, minute, or second values that are less than 10. '1979-10-30 1:2:3' is the same as '1979-10-30 01:02:03'.

Values specified as numbers should be 6, 8, 12, or 14 digits long. If a number is 8 or 14 digits long, it is assumed to be in YYYYMMDD or YYYYMMDDHHMMSS format and that the year is given by the first 4 digits. If the number is 6 or 12 digits long, it is assumed to be in YYMMDD or YYMMDDHHMMSS format and that the year is given by the first 2 digits. Numbers that are not one of these lengths are interpreted as though padded with leading zeros to the closest length.

Values specified as non-delimited strings are interpreted using their length as given. If the string is 8 or

14 characters long, the year is assumed to be given by the first 4 characters. Otherwise, the year is assumed to be given by the first 2 characters. The string is interpreted from left to right to find year, month, day, hour, minute, and second values, for as many parts as are present in the string. This means you should not use strings that have fewer than 6 characters. For example, if you specify `'9903'`, thinking that represents March, 1999, MySQL inserts a ``zero" date into your table. This is because the year and month values are `99` and `03`, but the day part is completely missing, so the value is not a legal date. However, as of MySQL 3.23, you can explicitly specify a value of zero to represent missing month or day parts. For example, you can use `'990300'` to insert the value `'1999-03-00'`.

You can to some extent assign values of one date type to an object of a different date type. However, there may be some alteration of the value or loss of information:

- If you assign a `DATE` value to a `DATETIME` or `TIMESTAMP` object, the time part of the resulting value is set to `'00:00:00'` because the `DATE` value contains no time information.
- If you assign a `DATETIME` or `TIMESTAMP` value to a `DATE` object, the time part of the resulting value is deleted because the `DATE` type stores no time information.
- Remember that although `DATETIME`, `DATE`, and `TIMESTAMP` values all can be specified using the same set of formats, the types do not all have the same range of values. For example, `TIMESTAMP` values cannot be earlier than `1970` or later than `2037`. This means that a date such as `'1968-01-01'`, while legal as a `DATETIME` or `DATE` value, is not a valid `TIMESTAMP` value and is converted to `0` if assigned to such an object.

Be aware of certain pitfalls when specifying date values:

- The relaxed format allowed for values specified as strings can be deceiving. For example, a value such as `'10:11:12'` might look like a time value because of the `':'` delimiter, but if used in a date context is interpreted as the year `'2010-11-12'`. The value `'10:45:15'` is converted to `'0000-00-00'` because `'45'` is not a legal month.
- The MySQL server performs only basic checking on the validity of a date: The ranges for year, month, and day are 1000 to 9999, 00 to 12, and 00 to 31, respectively. Any date containing parts not within these ranges is subject to conversion to `'0000-00-00'`. Please note that this still allows you to store invalid dates such as `'2002-04-31'`. To ensure that a date is valid, perform a check in your application.
- Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using the following rules:
  - Year values in the range `00-69` are converted to `2000-2069`.
  - Year values in the range `70-99` are converted to `1970-1999`.

11.3.1.1. `TIMESTAMP` Properties Prior to MySQL 4.1

The `TIMESTAMP` column type provides a type that you can use to automatically mark `INSERT` or `UPDATE` operations with the current date and time. If you have multiple `TIMESTAMP` columns in a table, only the first one is updated automatically. (From MySQL 4.1.2 on, you can specify which `TIMESTAMP` column updates; see Section 11.3.1.2, “`TIMESTAMP` Properties as of MySQL 4.1”.)

Automatic updating of the first `TIMESTAMP` column in a table occurs under any of the following conditions:

- You explicitly set the column to NULL.
- The column is not specified explicitly in an INSERT or LOAD DATA INFILE statement.
- The column is not specified explicitly in an UPDATE statement and some other column changes value. An UPDATE that sets a column to the value it does not cause the TIMESTAMP column to be updated; if you set a column to its current value, MySQL ignores the update for efficiency.

TIMESTAMP columns other than the first can also be set to the current date and time. Just set the column to NULL or to any function that produces the current date and time (NOW(), CURRENT_TIMESTAMP).

You can set any TIMESTAMP column to a value different from the current date and time by setting it explicitly to the desired value. This is true even for the first TIMESTAMP column. You can use this property if, for example, you want a TIMESTAMP to be set to the current date and time when you create a row, but not to be changed whenever the row is updated later:

- Let MySQL set the column when the row is created. This initializes it to the current date and time.
- When you perform subsequent updates to other columns in the row, set the TIMESTAMP column explicitly to its current value:

```
UPDATE tbl_name
    SET timestamp_col = timestamp_col,
        other_col1 = new_value1,
        other_col2 = new_value2, ...
```

Another way to maintain a column that records row-creation time is to use a DATETIME column that you initialize to NOW() when the row is created and leave alone for subsequent updates.

TIMESTAMP values may range from the beginning of 1970 to partway through the year 2037, with a resolution of one second. Values are displayed as numbers. When you store a value in a TIMESTAMP column, it is assumed to be represented in the current time zone, and is converted to UTC for storage. When you retrieve the value, it is converted from UTC back to the local time zone for display. Before MySQL 4.1.3, the server has a single time zone. As of 4.1.3, clients can set their time zone on a per-connection basis, as described in Time zone support.

The format in which MySQL retrieves and displays TIMESTAMP values depends on the display size, as illustrated by the following table. The ``full'' TIMESTAMP format is 14 digits, but TIMESTAMP columns may be created with shorter display sizes:

Column Type	Display Format
TIMESTAMP(14)	YYYYMMDDHHMMSS
TIMESTAMP(12)	YYMMDDHHMMSS
TIMESTAMP(10)	YYMMDDHHMM
TIMESTAMP(8)	YYYYMMDD
TIMESTAMP(6)	YYMMDD
TIMESTAMP(4)	YYMM
TIMESTAMP(2)	YY

All TIMESTAMP columns have the same storage size, regardless of display size. The most common dis-

play sizes are 6, 8, 12, and 14. You can specify an arbitrary display size at table creation time, but values of 0 or greater than 14 are coerced to 14. Odd-valued sizes in the range from 1 to 13 are coerced to the next higher even number.

`TIMESTAMP` columns store legal values using the full precision with which the value was specified, regardless of the display size. This has several implications:

- Always specify year, month, and day, even if your column types are `TIMESTAMP(4)` or `TIMESTAMP(2)`. Otherwise, the value is not a legal date and `0` is stored.
- If you use `ALTER TABLE` to widen a narrow `TIMESTAMP` column, information is displayed that previously was ``hidden."
- Similarly, narrowing a `TIMESTAMP` column does not cause information to be lost, except in the sense that less information is shown when the values are displayed.
- If you are planning to use `mysqldump` for the database, do not use `TIMESTAMP(4)` or `TIMESTAMP(2)`. The display format for these column types are not legal dates and `0` will be stored instead. This inconsistency is fixed starting with MySQL 4.1, where display width is ignored. To prepare for transition to versions after 4.0, you should change to use display widths of 6 or more, which will produce a legal display format. You can change the display width of `TIMESTAMP` column types, without losing any information, by using `ALTER TABLE` as indicated above.

  If you need to print the timestamps for external applications, you can use `MID()` to extract the relevant part of the timestamp: for example, to imitate the `TIMESTAMP(4)` display format.
- Although `TIMESTAMP` values are stored to full precision, the only function that operates directly on the underlying stored value is `UNIX_TIMESTAMP()`. Other functions operate on the formatted retrieved value. This means you cannot use a function such as `HOUR()` or `SECOND()` unless the relevant part of the `TIMESTAMP` value is included in the formatted value. For example, the `HH` part of a `TIMESTAMP` column is not displayed unless the display size is at least 10, so trying to use `HOUR()` on shorter `TIMESTAMP` values produces a meaningless result.

11.3.1.2. `TIMESTAMP` Properties as of MySQL 4.1

In MySQL 4.1 and up, the properties of the `TIMESTAMP` column type change in the ways described in this section.

From MySQL 4.1.0 on, `TIMESTAMP` display format differs from that of earlier MySQL releases:

- `TIMESTAMP` columns are displayed in the same format as `DATETIME` columns.
- Display widths (used as described in the preceding section) are no longer supported. In other words, for declarations such as `TIMESTAMP(2)`, `TIMESTAMP(4)`, and so on, the display width is ignored.

Beginning with MySQL 4.1.1, the MySQL server can be run in `MAXDB` mode. When the server runs in this mode, `TIMESTAMP` is identical with `DATETIME`. That is, if the server is running in `MAXDB` mode at the time that a table is created, `TIMESTAMP` columns are created as `DATETIME` columns. As a result, such columns use `DATETIME` display format, have the same range of values, and there is no automatic initialization or updating to the current date and time.

To enable `MAXDB` mode, set the server SQL mode to `MAXDB` at startup using the `--sql-mode=MAXDB` server option or by setting the global `sql_mode` variable at runtime:

```
mysql> SET GLOBAL sql_mode=MAXDB;
```

A client can cause the server to run in MAXDB mode for its own connection as follows:

```
mysql> SET SESSION sql_mode=MAXDB;
```

As of MySQL 5.0.2, MySQL does not accept timestamp values that include a zero in the day or month column or values that are not a valid date. (The exception is the special value '0000-00-00 00:00:00'.)

Beginning with MySQL 4.1.2, you have more flexible control over when automatic TIMESTAMP initialization and updating occur and which column should have those behaviors:

- You can assign the current timestamp as the default value and the auto-update value, as before. But it is possible to have just one automatic behavior or the other, or neither of them. (It is not possible to have one behavior for one column and the other for another column.)
- You can specify which TIMESTAMP column to automatically initialize or update to the current date and time. This no longer need be the first TIMESTAMP column.

The following discussion describes the revised syntax and behavior. Note that this information applies only to TIMESTAMP columns for tables not created with MAXDB mode enabled. As noted earlier in this section, MAXDB mode causes columns to be created as DATETIME columns.

The following items summarize the pre-4.1.2 properties for TIMESTAMP initialization and updating:

The first TIMESTAMP column in table row automatically is set to the current timestamp when the record is created if the column is set to NULL or is not specified at all.

The first TIMESTAMP column in table row automatically is updated to the current timestamp when the value of any other column in the row is changed, unless the TIMESTAMP column explicitly is assigned a value other than NULL.

If a DEFAULT value is specified for the first TIMESTAMP column when the table is created, it is silently ignored.

Other TIMESTAMP columns in the table can be set to the current TIMESTAMP by assigning NULL to them, but they do not update automatically.

As of 4.1.2, you have more flexibility in deciding which TIMESTAMP column automatically is initialized and updated to the current timestamp. The rules are as follows:

If a DEFAULT value is specified for the first TIMESTAMP column in a table, it is not ignored. The default can be CURRENT_TIMESTAMP or a constant date and time value.

DEFAULT NULL is the same as DEFAULT CURRENT_TIMESTAMP for the *first* TIMESTAMP column. For any other TIMESTAMP column, DEFAULT NULL is treated as DEFAULT 0.

Any single TIMESTAMP column in a table can be set to be the one that is initialized to the current timestamp and/or updated automatically.

In a CREATE TABLE statement, the first TIMESTAMP column can be declared in any of the following ways:

- With both DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP clauses, the column has the current timestamp for its default value, and is automatically updated.

- With neither DEFAULT nor ON UPDATE clauses, it is the same as DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP.
- With a DEFAULT CURRENT_TIMESTAMP clause and no ON UPDATE clause, the column has the current timestamp for its default value but is not automatically updated.
- With no DEFAULT clause and with an ON UPDATE CURRENT_TIMESTAMP clause, the column has a default of 0 and is automatically updated.
- With a constant DEFAULT value, the column has the given default. If the column has an ON UPDATE CURRENT_TIMESTAMP clause, it is automatically updated, otherwise not.

In other words, you can use the current timestamp for both the initial value and the auto-update value, or either one, or neither. (For example, you can specify ON UPDATE to get auto-update without also having the column auto-initialized.)

Any of CURRENT_TIMESTAMP, CURRENT_TIMESTAMP(), or NOW() can be used in the DEFAULT and ON UPDATE clauses. They all have the same effect.

The order of the two attributes does not matter. If both DEFAULT and ON UPDATE are specified for a TIMESTAMP column, either can precede the other.

Example. These statements are equivalent:

```
CREATE TABLE t (ts TIMESTAMP);
CREATE TABLE t (ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                             ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE t (ts TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
                             DEFAULT CURRENT_TIMESTAMP);
```

To specify automatic default or updating for a TIMESTAMP column other than the first one, you must suppress the automatic initialization and update behaviors for the first TIMESTAMP column by explicitly assigning it a constant DEFAULT value (for example, DEFAULT 0 or DEFAULT '2003-01-01 00:00:00'). Then for the other TIMESTAMP column, the rules are the same as for the first TIMESTAMP column, except that you cannot omit both of the DEFAULT and ON UPDATE clauses. If you do that, no automatic initialization or updating occurs.

Example. These statements are equivalent:

```
CREATE TABLE t (
    ts1 TIMESTAMP DEFAULT 0,
    ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                  ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE t (
    ts1 TIMESTAMP DEFAULT 0,
    ts2 TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
                  DEFAULT CURRENT_TIMESTAMP);
```

Beginning with MySQL 4.1.3, you can set the current time zone on a per-connection basis, as described in Time zone support. TIMESTAMP values still are stored in UTC, but are converted from the current time zone for storage, and converted back to the current time zone for retrieval. As long as the time zone setting remains the same, you get back the same value you store. If you store a TIMESTAMP value, then change the time zone and retrieve the value, it is different than the value you stored. This occurs because the same time zone is not used for conversion in both directions. The current time zone is available as the value of the time_zone system variable.

Beginning with MySQL 4.1.6, you can include the NULL attribute in the definition of a TIMESTAMP

column to allow the column to contain NULL values. For example:

```
CREATE TABLE t
(
  ts1 TIMESTAMP NULL DEFAULT NULL,
  ts2 TIMESTAMP NULL DEFAULT 0,
  ts3 TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP
);
```

Before MySQL 4.1.6 (and even as of 4.1.6 if the NULL attribute is not specified), setting the column to NULL sets it to the current timestamp.

11.3.2. The TIME Type

MySQL retrieves and displays TIME values in 'HH:MM:SS' format (or 'HHH:MM:SS' format for large hours values). TIME values may range from '-838:59:59' to '838:59:59'. The reason the hours part may be so large is that the TIME type may be used not only to represent a time of day (which must be less than 24 hours), but also elapsed time or a time interval between two events (which may be much greater than 24 hours, or even negative).

You can specify TIME values in a variety of formats:

- As a string in 'D HH:MM:SS.fraction' format. You can also use one of the following ``relaxed'' syntaxes: 'HH:MM:SS.fraction', 'HH:MM:SS', 'HH:MM', 'D HH:MM:SS', 'D HH:MM', 'D HH', or 'SS'. Here D represents days and can have a value from 0 to 34. Note that MySQL doesn't yet store the fraction part.
- As a string with no delimiters in 'HHMMSS' format, provided that it makes sense as a time. For example, '101112' is understood as '10:11:12', but '109712' is illegal (it has a nonsensical minute part) and becomes '00:00:00'.
- As a number in HHMMSS format, provided that it makes sense as a time. For example, 101112 is understood as '10:11:12'. The following alternative formats are also understood: SS, MMSS, HHMMSS, HHMMSS.fraction. Note that MySQL doesn't yet store the fraction part.
- As the result of a function that returns a value that is acceptable in a TIME context, such as CURRENT_TIME.

For TIME values specified as strings that include a time part delimiter, it is not necessary to specify two digits for hours, minutes, or seconds values that are less than 10. '8:3:2' is the same as '08:03:02'.

Be careful about assigning ``short'' TIME values to a TIME column. Without colons, MySQL interprets values using the assumption that the rightmost digits represent seconds. (MySQL interprets TIME values as elapsed time rather than as time of day.) For example, you might think of '1112' and 1112 as meaning '11:12:00' (12 minutes after 11 o'clock), but MySQL interprets them as '00:11:12' (11 minutes, 12 seconds). Similarly, '12' and 12 are interpreted as '00:00:12'. TIME values with colons, by contrast, are always treated as time of the day. That is '11:12' mean '11:12:00', not '00:11:12'.

Values that lie outside the TIME range but are otherwise legal are clipped to the closest endpoint of the range. For example, '-850:00:00' and '850:00:00' are converted to '-838:59:59' and '838:59:59'.

Illegal TIME values are converted to '00:00:00'. Note that because '00:00:00' is itself a legal TIME value, there is no way to tell, from a value of '00:00:00' stored in a table, whether the original value was specified as '00:00:00' or whether it was illegal.

11.3.3. The YEAR Type

The YEAR type is a one-byte type used for representing years.

MySQL retrieves and displays YEAR values in YYYY format. The range is 1901 to 2155.

You can specify YEAR values in a variety of formats:

- As a four-digit string in the range '1901' to '2155'.
- As a four-digit number in the range 1901 to 2155.
- As a two-digit string in the range '00' to '99'. Values in the ranges '00' to '69' and '70' to '99' are converted to YEAR values in the ranges 2000 to 2069 and 1970 to 1999.
- As a two-digit number in the range 1 to 99. Values in the ranges 1 to 69 and 70 to 99 are converted to YEAR values in the ranges 2001 to 2069 and 1970 to 1999. Note that the range for two-digit numbers is slightly different from the range for two-digit strings, because you cannot specify zero directly as a number and have it be interpreted as 2000. You must specify it as a string '0' or '00' or it is interpreted as 0000.
- As the result of a function that returns a value that is acceptable in a YEAR context, such as NOW().

Illegal YEAR values are converted to 0000.

11.3.4. Y2K Issues and Date Types

MySQL itself is year 2000 (Y2K) safe (see Section 1.2.5, “Year 2000 Compliance”), but input values presented to MySQL may not be. Any input containing two-digit year values is ambiguous, because the century is unknown. Such values must be interpreted into four-digit form because MySQL stores years internally using four digits.

For DATETIME, DATE, TIMESTAMP, and YEAR types, MySQL interprets dates with ambiguous year values using the following rules:

- Year values in the range 00-69 are converted to 2000-2069.
- Year values in the range 70-99 are converted to 1970-1999.

Remember that these rules provide only reasonable guesses as to what your data values mean. If the heuristics used by MySQL do not produce the correct values, you should provide unambiguous input containing four-digit year values.

ORDER BY properly sorts TIMESTAMP or YEAR values that have two-digit years.

Some functions like MIN() and MAX() convert a TIMESTAMP or YEAR to a number. This means that a value with a two-digit year does not work properly with these functions. The fix in this case is to convert the TIMESTAMP or YEAR to four-digit year format or use something like MIN(DATE_ADD(timestamp,INTERVAL 0 DAYS)).

11.4. String Types

The string types are CHAR, VARCHAR, BINARY, VARBINARY, BLOB, TEXT, ENUM, and SET. This section describes how these types work and how to use them in your queries.

11.4.1. The CHAR and VARCHAR Types

The CHAR and VARCHAR types are similar, but differ in the way they are stored and retrieved. As of MySQL 5.0.3, they also differ in maximum length and in whether trailing spaces are retained.

The CHAR and VARCHAR types are declared with a length that indicates the maximum number of characters you want to store. For example, CHAR(30) can hold up to 30 characters. (Before MySQL 4.1, the length is interpreted as number of bytes.)

The length of a CHAR column is fixed to the length that you declare when you create the table. The length can be any value from 0 to 255. (Before MySQL 3.23, the length of CHAR may be from 1 to 255.) When CHAR values are stored, they are right-padded with spaces to the specified length. When CHAR values are retrieved, trailing spaces are removed.

Values in VARCHAR columns are variable-length strings. The length can be specified as 1 to 255 before MySQL 4.0.2, 0 to 255 as of MySQL 4.0.2, and 0 to 65,535 as of MySQL 5.0.3. (The maximum actual length of a VARCHAR in MySQL 5.0 is determined by the maximum row size and the character set you use. The maximum effective length is 65,532 bytes.)

In contrast to CHAR, VARCHAR values are stored using only as many characters as are needed, plus one byte to record the length (two bytes for columns that are declared with a length longer than 255).

VARCHAR values are not padded when they are stored. Handling of trailing spaces is version-dependent. As of MySQL 5.0.3, trailing spaces are retained when values are stored and retrieved, in conformance with standard SQL. Before MySQL 5.0.3, trailing spaces are removed from values when they are stored into a VARCHAR column; this means the spaces also are absent from retrieved values.

No lettercase conversion takes place during storage or retrieval.

If you assign a value to a CHAR or VARCHAR column that exceeds the column's maximum length, the value is truncated to fit. If the truncated characters are not spaces, a warning is generated. You can cause an error to occur rather than an error by using ``strict'' SQL mode. See Section 5.2.2, “The Server SQL Mode”.

Before MySQL 5.0.3, if you need a data type for which trailing spaces are not removed, consider using a BLOB or TEXT type. Also, if you want to store binary values such as results from an encryption or compression function that might contain arbitrary byte values, use a BLOB column rather than a CHAR or VARCHAR column, to avoid potential problems with trailing space removal that would change data values.

The following table illustrates the differences between the two types of columns by showing the result of storing various string values into CHAR(4) and VARCHAR(4) columns:

Value	CHAR(4)	**Storage Required**	VARCHAR(4)	**Storage Required**
''	' '	4 bytes	''	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

The values retrieved from the CHAR(4) and VARCHAR(4) columns are the same in each case, because trailing spaces are removed from CHAR columns upon retrieval.

As of MySQL 4.1, values in CHAR and VARCHAR columns are sorted and compared according to the collation of the character set assigned to the column. Before MySQL 4.1, sorting and comparison are based on the collation of the server character set; you can declare the column with the BINARY attribute to cause sorting and comparison to be based on the numeric values of the bytes in column values. BIN-

ARY doesn't affect how column values are stored or retrieved.

The BINARY attribute is sticky. This means that if a column marked BINARY is used in an expression, the whole expression is treated as a BINARY value.

From MySQL 4.1.0 on, column type CHAR BYTE is an alias for CHAR BINARY. This is a compatibility feature.

From MySQL 4.1.0 on, the ASCII attribute can be specified for CHAR. It assigns the latin1 character set.

From MySQL 4.1.1 on, the UNICODE attribute can be specified for CHAR. It assigns the ucs2 character set.

MySQL may silently change the type of a CHAR or VARCHAR column at table creation time. See Section 13.2.6.1, “Silent Column Specification Changes”.

11.4.2. The BINARY and VARBINARY Types

The BINARY and VARBINARY types are similar to CHAR and VARCHAR, except that they contain binary strings rather than non-binary strings. That is, they contain byte strings rather than character strings. This means they have no character set, and sorting and comparison is based on the numeric values of the bytes in column values.

The allowable maximum length and handling of trailing spaces is the same for BINARY and VARBINARY as it is for CHAR and VARCHAR, except that the length for BINARY and VARBINARY is a length in bytes rather than in characters.

Before MySQL 4.1.2, BINARY(*M*) and VARBINARY(*M*) are treated as CHAR(*M*) BINARY and VARCHAR(*M*) BINARY. As of MySQL 4.1.2, BINARY and VARBINARY are available as distinct data types, and for CHAR(*M*) BINARY and VARCHAR(*M*) BINARY, the BINARY attribute does not cause the column to be treated as a binary string column. Instead, it causes the binary collation for the column character set to be used, but the column itself contains non-binary character strings rather than binary byte strings. For example, in 4.1 and up, CHAR(5) BINARY is treated as CHAR(5) CHARACTER SET latin1 COLLATE latin1_bin, assuming that the default character set is latin1.

11.4.3. The BLOB and TEXT Types

A BLOB is a binary large object that can hold a variable amount of data. The four BLOB types, TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB, differ only in the maximum length of the values they can hold.

See Section 11.5, “Column Type Storage Requirements”.

The four TEXT types, TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT, correspond to the four BLOB types and have the same maximum lengths and storage requirements.

BLOB columns are treated as binary strings (byte strings). TEXT columns are treated as non-binary strings (character strings). BLOB columns have no character set, and sorting and comparison is based on the numeric values of the bytes in column values. TEXT columns have a character set, and values are sorted and compared based on the collation of the character set assigned to the column as of MySQL 4.1. Before 4.1, TEXT sorting and comparison are based on the collation of the server character set.

No lettercase conversion takes place during storage or retrieval.

If you assign a value to a BLOB or TEXT column that exceeds the column type's maximum length, the value is truncated to fit. If the truncated characters are not spaces, a warning is generated. You can cause an error to occur rather than an error by using ``strict'' SQL mode. See Section 5.2.2, “The Server SQL

Mode".

In most respects, you can regard a BLOB column as a VARBINARY column that can be as big as you like. Similarly, you can regard a TEXT column as a VARCHAR column. BLOB and TEXT differ from VARBINARY and VARCHAR in the following ways::

- There is no trailing-space removal for BLOB and TEXT columns when values are stored or retrieved. Before MySQL 5.0.3, this differs from VARBINARY and VARCHAR, for which trailing spaces are removed when values are stored.
- You can have indexes on BLOB and TEXT columns only as of MySQL 3.23.2 for MyISAM tables or MySQL 4.0.14 for InnoDB tables. Older versions of MySQL did not support indexing these column types.
- For indexes on BLOB and TEXT columns, you must specify an index prefix length. For CHAR and VARCHAR, a prefix length is optional.
- BLOB and TEXT columns cannot have DEFAULT values.

From MySQL 4.1.0 on, LONG and LONG VARCHAR map to the MEDIUMTEXT data type. This is a compatibility feature. If you use the BINARY attribute with a TEXT column type, the column is assigned the binary collation of the column character set.

MySQL Connector/ODBC defines BLOB values as LONGVARBINARY and TEXT values as LONGVARCHAR.

Because BLOB and TEXT values may be extremely long, you may encounter some constraints in using them:

- Only the first max_sort_length bytes of the column are used when sorting. The default value of max_sort_length is 1024; this value can be changed using the --max_sort_length option when starting the **mysqld** server. See Section 5.2.3, "Server System Variables".

  As of MySQL 4.0.3, you can make more bytes significant in sorting or grouping by increasing the value of max_sort_length at runtime. Any client can change the value of its session max_sort_length variable:

  ```
  mysql> SET max_sort_length = 2000;
  mysql> SELECT id, comment FROM tbl_name
      -> ORDER BY comment;
  ```

  Another way to use GROUP BY or ORDER BY on a BLOB or TEXT column containing long values when you want more than max_sort_length bytes to be significant is to convert the column value into a fixed-length object. The standard way to do this is with the SUBSTRING function. For example, the following statement causes 2000 bytes of the comment column to be taken into account for sorting:

  ```
  mysql> SELECT id, SUBSTRING(comment,1,2000) FROM tbl_name
      -> ORDER BY SUBSTRING(comment,1,2000);
  ```

  Before MySQL 3.23.2, you can group on an expression involving BLOB or TEXT values by using a column alias or by specifying the column position:

  ```
  mysql> SELECT id, SUBSTRING(comment,1,2000) AS b
      -> FROM tbl_name GROUP BY b;
  mysql> SELECT id, SUBSTRING(comment,1,2000)
  ```

```
-> FROM tbl_name GROUP BY 2;
```

- The maximum size of a BLOB or TEXT object is determined by its type, but the largest value you actually can transmit between the client and server is determined by the amount of available memory and the size of the communications buffers. You can change the message buffer size by changing the value of the max_allowed_packet variable, but you must do so for both the server and your client program. For example, both **mysql** and **mysqldump** allow you to change the client-side max_allowed_packet value. See Section 7.5.2, “Tuning Server Parameters”, Section 8.3, “**mysql**, the Command-Line Tool”, and Section 8.8, “The **mysqldump** Database Backup Program”.

Each BLOB or TEXT value is represented internally by a separately allocated object. This is in contrast to all other column types, for which storage is allocated once per column when the table is opened.

11.4.4. The ENUM Type

An ENUM is a string object with a value chosen from a list of allowed values that are enumerated explicitly in the column specification at table creation time.

The value may also be the empty string (''') or NULL under certain circumstances:

- If you insert an invalid value into an ENUM (that is, a string not present in the list of allowed values), the empty string is inserted instead as a special error value. This string can be distinguished from a ``normal'' empty string by the fact that this string has the numerical value 0. More about this later.
- If an ENUM column is declared to allow NULL, the NULL value is a legal value for the column, and the default value is NULL. If an ENUM column is declared NOT NULL, its default value is the first element of the list of allowed values.

Each enumeration value has an index:

- Values from the list of allowable elements in the column specification are numbered beginning with 1.
- The index value of the empty string error value is 0. This means that you can use the following SELECT statement to find rows into which invalid ENUM values were assigned:

  ```
  mysql> SELECT * FROM tbl_name WHERE enum_col=0;
  ```

- The index of the NULL value is NULL.

For example, a column specified as ENUM('one', 'two', 'three') can have any of the values shown here. The index of each value is also shown:

Value	Index
NULL	NULL
''	0
'one'	1
'two'	2
'three'	3

An enumeration can have a maximum of 65,535 elements.

Starting from MySQL 3.23.51, trailing spaces are automatically deleted from ENUM member values when the table is created.

Values stored into an ENUM column are displayed when retrieved using the lettercase that was used in the column definition. Before MySQL 4.1.1, lettercase is irrelevant when you assign values to an ENUM column. As of 4.1.1, ENUM columns can be assigned a character set and collation. For binary or case-sensitive collations, lettercase does matter when you assign values to to the column.

If you retrieve an ENUM value in a numeric context, the column value's index is returned. For example, you can retrieve numeric values from an ENUM column like this:

```
mysql> SELECT enum_col+0 FROM tbl_name;
```

If you store a number into an ENUM column, the number is treated as an index, and the value stored is the enumeration member with that index. (However, this does not work with LOAD DATA, which treats all input as strings.) It's not advisable to define an ENUM column with enumeration values that look like numbers, because this can easily become confusing. For example, the following column has enumeration members with string values of '0', '1', and '2', but numeric index values of 1, 2, and 3:

```
numbers ENUM('0','1','2')
```

ENUM values are sorted according to the order in which the enumeration members were listed in the column specification. (In other words, ENUM values are sorted according to their index numbers.) For example, 'a' sorts before 'b' for ENUM('a', 'b'), but 'b' sorts before 'a' for ENUM('b', 'a'). The empty string sorts before non-empty strings, and NULL values sort before all other enumeration values. To prevent unexpected results, specify the ENUM list in alphabetical order. You can also use GROUP BY CAST(col AS VARCHAR) or GROUP BY CONCAT(col) to make sure that the column is sorted lexically rather than by index number.

If you want to determine all possible values for an ENUM column, use SHOW COLUMNS FROM *tbl_name* LIKE *enum_col* and parse the ENUM definition in the second column of the output.

11.4.5. The SET Type

A SET is a string object that can have zero or more values, each of which must be chosen from a list of allowed values specified when the table is created. SET column values that consist of multiple set members are specified with members separated by commas (','). A consequence of this is that SET member values cannot themselves contain commas.

For example, a column specified as SET('one', 'two') NOT NULL can have any of these values:

```
''
'one'
'two'
'one,two'
```

A SET can have a maximum of 64 different members.

Starting from MySQL 3.23.51, trailing spaces are automatically deleted from SET member values when the table is created.

Values stored into an SET column are displayed when retrieved using the lettercase that was used in the column definition. Before MySQL 4.1.1, lettercase is irrelevant when you assign values to an SET column. As of 4.1.1, SET columns can be assigned a character set and collation. For binary or case-

sensitive collations, lettercase does matter when you assign values to to the column.

MySQL stores SET values numerically, with the low-order bit of the stored value corresponding to the first set member. If you retrieve a SET value in a numeric context, the value retrieved has bits set corresponding to the set members that make up the column value. For example, you can retrieve numeric values from a SET column like this:

```
mysql> SELECT set_col+0 FROM tbl_name;
```

If a number is stored into a SET column, the bits that are set in the binary representation of the number determine the set members in the column value. For a column specified as SET('a','b','c','d'), the members have the following decimal and binary values:

SET **Member**	**Decimal Value**	**Binary Value**
'a'	1	0001
'b'	2	0010
'c'	4	0100
'd'	8	1000

If you assign a value of 9 to this column, that is 1001 in binary, so the first and fourth SET value members 'a' and 'd' are selected and the resulting value is 'a,d'.

For a value containing more than one SET element, it does not matter what order the elements are listed in when you insert the value. It also does not matter how many times a given element is listed in the value. When the value is retrieved later, each element in the value appears once, with elements listed according to the order in which they were specified at table creation time. If a column is specified as SET('a','b','c','d'), then 'a,d', 'd,a', and 'd,a,a,d,d' all appear as 'a,d' when retrieved.

If you set a SET column to an unsupported value, the value is ignored.

SET values are sorted numerically. NULL values sort before non-NULL SET values.

Normally, you search for SET values using the FIND_IN_SET() function or the LIKE operator:

```
mysql> SELECT * FROM tbl_name WHERE FIND_IN_SET('value',set_col)>0;
mysql> SELECT * FROM tbl_name WHERE set_col LIKE '%value%';
```

The first statement finds rows where *set_col* contains the *value* set member. The second is similar, but not the same: It finds rows where *set_col* contains *value* anywhere, even as a substring of another set member.

The following statements also are legal:

```
mysql> SELECT * FROM tbl_name WHERE set_col & 1;
mysql> SELECT * FROM tbl_name WHERE set_col = 'val1,val2';
```

The first of these statements looks for values containing the first set member. The second looks for an exact match. Be careful with comparisons of the second type. Comparing set values to *'val1,val2'* returns different results than comparing values to *'val2,val1'*. You should specify the values in the same order they are listed in the column definition.

If you want to determine all possible values for a SET column, use SHOW COLUMNS FROM *tbl_name* LIKE *set_col* and parse the SET definition in the second column of the output.

11.5. Column Type Storage Requirements

The storage requirements for each of the column types supported by MySQL are listed by category.

The maximum size of a row in a `MyISAM` table is 65,534 bytes. Each `BLOB` and `TEXT` column accounts for only five to nine bytes toward this size.

If a `MyISAM` or `ISAM` table includes any variable-length column types, the record format is also variable length. When a table is created, MySQL may, under certain conditions, change a column from a variable-length type to a fixed-length type or vice versa. See Section 13.2.6.1, “Silent Column Specification Changes”.

Storage Requirements for Numeric Types

Column Type	Storage Required
`TINYINT`	1 byte
`SMALLINT`	2 bytes
`MEDIUMINT`	3 bytes
`INT`, `INTEGER`	4 bytes
`BIGINT`	8 bytes
`FLOAT(p)`	4 bytes if 0 <= *p* <= 24, 8 bytes if 25 <= *p* <= 53
`FLOAT`	4 bytes
`DOUBLE [PRECISION]`, item `REAL`	8 bytes
`DECIMAL(M,D)`, `NUMERIC(M,D)`	*M*+2 bytes if *D* > 0, *M*+1 bytes if *D* = 0 (*D*+2, if *M* < *D*)
`BIT(M)`	approximately (*M*+7)/8 bytes

Storage Requirements for Date and Time Types

Column Type	Storage Required
`DATE`	3 bytes
`DATETIME`	8 bytes
`TIMESTAMP`	4 bytes
`TIME`	3 bytes
`YEAR`	1 byte

Storage Requirements for String Types

Column Type	Storage Required
`CHAR(M)`	*M* bytes, 0 <= *M* <= 255
`VARCHAR(M)`	*L*+1 bytes, where *L* <= *M* and 0 <= *M* <= 255
`BINARY(M)`	*M* bytes, 0 <= *M* <= 255
`VARBINARY(M)`	*L*+1 bytes, where *L* <= *M* and 0 <= *M* <= 255
`TINYBLOB`, `TINYTEXT`	*L*+1 bytes, where *L* < 2^8
`BLOB`, `TEXT`	*L*+2 bytes, where *L* < 2^16
`MEDIUMBLOB`, `MEDIUMTEXT`	*L*+3 bytes, where *L* < 2^24
`LONGBLOB`, `LONGTEXT`	*L*+4 bytes, where *L* < 2^32

ENUM('*value1*','*value2*',...)	1 or 2 bytes, depending on the number of enumeration values (65,535 values maximum)
SET('*value1*','*value2*',...)	1, 2, 3, 4, or 8 bytes, depending on the number of set members (64 members maximum)

VARCHAR and the BLOB and TEXT types are variable-length types. For each, the storage requirements depend on the actual length of column values (represented by *L* in the preceding table), rather than on the type's maximum possible size. For example, a VARCHAR(10) column can hold a string with a maximum length of 10. The actual storage required is the length of the string (*L*), plus 1 byte to record the length of the string. For the string 'abcd', *L* is 4 and the storage requirement is 5 bytes.

For the CHAR, VARCHAR, and TEXT types, *L* and *M* in the preceding table should be interpreted as number of bytes before MySQL 4.1 and as number of characters thereafter. Lengths for these types in columns specifications indicate number of characters from MySQL 4.1 on. The number of extra bytes for recording lengths for variable-length data types is unchanged. For example, *L*+1 bytes to store a TINYTEXT value before MySQL 4.1 becomes *L* characters + 1 byte to store the length as of MySQL 4.1.

As of MySQL 5.0.3, the NDBCLUSTER engine supports only fixed-width columns. This means that a VARCHAR column from a table in a MySQL CLuster will behave almost as if it were of type CHAR (except that each record still has one extra byte overhead). For example, in a Cluster table, *each* record in a column declared as VARCHAR(100) will take up 101 bytes for storage, regardless of the length of the string actually stored in any given record.

The BLOB and TEXT types require 1, 2, 3, or 4 bytes to record the length of the column value, depending on the maximum possible length of the type. See Section 11.4.3, “The BLOB and TEXT Types”.

The size of an ENUM object is determined by the number of different enumeration values. One byte is used for enumerations with up to 255 possible values. Two bytes are used for enumerations with up to 65,535 values. See Section 11.4.4, “The ENUM Type”.

The size of a SET object is determined by the number of different set members. If the set size is *N*, the object occupies (*N*+7)/8 bytes, rounded up to 1, 2, 3, 4, or 8 bytes. A SET can have a maximum of 64 members. See Section 11.4.5, “The SET Type”.

11.6. Choosing the Right Type for a Column

For the most efficient use of storage, try to use the most precise type in all cases. For example, if an integer column is used for values in the range from 1 to 99999, MEDIUMINT UNSIGNED is the best type. Of the types that represent all the required values, it uses the least amount of storage.

Accurate representation of monetary values is a common problem. In MySQL, you should use the DECIMAL type. This is stored as a string, so no loss of accuracy should occur. (Calculations on DECIMAL values may still be done using double-precision operations, however.) If accuracy is not too important, the DOUBLE type may also be good enough.

For high precision, you can always convert to a fixed-point type stored in a BIGINT. This allows you to do all calculations with integers and convert results back to floating-point values only when necessary.

11.7. Using Column Types from Other Database Engines

To make it easier to use code written for SQL implementations from other vendors, MySQL maps

column types as shown in the following table. These mappings make it easier to import table definitions from other database engines into MySQL:

Other Vendor Type	MySQL Type
`BINARY(M)`	`CHAR(M) BINARY` (before MySQL 4.1.2)
`CHAR VARYING(M)`	`VARCHAR(M)`
`FLOAT4`	`FLOAT`
`FLOAT8`	`DOUBLE`
`INT1`	`TINYINT`
`INT2`	`SMALLINT`
`INT3`	`MEDIUMINT`
`INT4`	`INT`
`INT8`	`BIGINT`
`LONG VARBINARY`	`MEDIUMBLOB`
`LONG VARCHAR`	`MEDIUMTEXT`
`LONG`	`MEDIUMTEXT` (MySQL 4.1.0 on)
`MIDDLEINT`	`MEDIUMINT`
`VARBINARY(M)`	`VARCHAR(M) BINARY` (before MySQL 4.1.2)

As of MySQL 4.1.2, `BINARY` and `VARBINARY` are distinct data types and are not converted to `CHAR BINARY` and `VARCHAR BINARY`.

Column type mapping occurs at table creation time, after which the original type specifications are discarded. If you create a table with types used by other vendors and then issue a `DESCRIBE` *`tbl_name`* statement, MySQL reports the table structure using the equivalent MySQL types.

Chapter 12. Functions and Operators

Expressions can be used at several points in SQL statements, such as in the `ORDER BY` or `HAVING` clauses of `SELECT` statements, in the `WHERE` clause of a `SELECT`, `DELETE`, or `UPDATE` statement, or in `SET` statements. Expressions can be written using literal values, column values, `NULL`, functions, and operators. This chapter describes the functions and operators that are allowed for writing expressions in MySQL.

An expression that contains `NULL` always produces a `NULL` value unless otherwise indicated in the documentation for a particular function or operator.

Note: By default, there must be no whitespace between a function name and the parenthesis following it. This helps the MySQL parser distinguish between function calls and references to tables or columns that happen to have the same name as a function. Spaces around function arguments are permitted, though.

You can tell the MySQL server to accept spaces after function names by starting it with the `--sql-mode=IGNORE_SPACE` option. Individual client programs can request this behavior by using the `CLIENT_IGNORE_SPACE` option for `mysql_real_connect()`. In either case, all function names become reserved words. See Section 5.2.2, “The Server SQL Mode”.

For the sake of brevity, most examples in this chapter display the output from the **mysql** program in abbreviated form. Instead of showing examples in this format:

```
mysql> SELECT MOD(29,9);
+-----------+
| mod(29,9) |
+-----------+
|         2 |
+-----------+
1 rows in set (0.00 sec)
```

This format is used instead:

```
mysql> SELECT MOD(29,9);
        -> 2
```

12.1. Operators

12.1.1. Operator Precedence

Operator precedences are shown in the following list, from lowest precedence to the highest. Operators that are shown together on a line have the same precedence.

```
:=
||, OR, XOR
&&, AND
NOT
BETWEEN, CASE, WHEN, THEN, ELSE
=, <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
|
&
<<, >>
-, +
*, /, DIV, %, MOD
^
```

```
- (unary minus), ~ (unary bit inversion)
!
BINARY, COLLATE
```

The precedence shown for `NOT` is as of MySQL 5.0.2. For earlier versions, or from 5.0.2 on if the `HIGH_NOT_PRECEDENCE` SQL mode is enabled, the precedence of `NOT` is the same as that of the `!` operator. See Section 5.2.2, “The Server SQL Mode”.

12.1.2. Parentheses

- `( ... )`

  Use parentheses to force the order of evaluation in an expression. For example:

  ```
  mysql> SELECT 1+2*3;
          -> 7
  mysql> SELECT (1+2)*3;
          -> 9
  ```

12.1.3. Comparison Functions and Operators

Comparison operations result in a value of `1` (TRUE), `0` (FALSE), or `NULL`. These operations work for both numbers and strings. Strings are automatically converted to numbers and numbers to strings as necessary.

Some of the functions in this section (such as `LEAST()` and `GREATEST()`) return values other than `1` (TRUE), `0` (FALSE), or `NULL`. However, the value they return is based on comparison operations performed as described by the following rules.

MySQL compares values using the following rules:

- If one or both arguments are `NULL`, the result of the comparison is `NULL`, except for the `NULL`-safe `<=>` equality comparison operator.
- If both arguments in a comparison operation are strings, they are compared as strings.
- If both arguments are integers, they are compared as integers.
- Hexadecimal values are treated as binary strings if not compared to a number.
- If one of the arguments is a `TIMESTAMP` or `DATETIME` column and the other argument is a constant, the constant is converted to a timestamp before the comparison is performed. This is done to be more ODBC-friendly. Note that this is not done for arguments in `IN()`! To be safe, always use complete datetime/date/time strings when doing comparisons.
- In all other cases, the arguments are compared as floating-point (real) numbers.

By default, string comparisons are not case sensitive and use the current character set (ISO-8859-1 Latin1 by default, which also works excellently for English).

To convert a value to a specific type for comparison purposes, you can use the `CAST()` function. String

values can be converted to a different character set using `CONVERT()`. See Section 12.7, “Cast Functions and Operators”.

The following examples illustrate conversion of strings to numbers for comparison operations:

```
mysql> SELECT 1 > '6x';
        -> 0
mysql> SELECT 7 > '6x';
        -> 1
mysql> SELECT 0 > 'x6';
        -> 0
mysql> SELECT 0 = 'x6';
        -> 1
```

Note that when you are comparing a string column with a number, MySQL can't use an index on the column to quickly look up the value. If *`str_col`* is an indexed string column, the index cannot be used when performing the lookup in the following statement:

```
SELECT * FROM tbl_name WHERE str_col=1;
```

The reason for this is that there are many different strings that may convert to the value `1`: `'1'`, `' 1'`, `'1a'`, ...

- `=`

  Equal:

  ```
  mysql> SELECT 1 = 0;
          -> 0
  mysql> SELECT '0' = 0;
          -> 1
  mysql> SELECT '0.0' = 0;
          -> 1
  mysql> SELECT '0.01' = 0;
          -> 0
  mysql> SELECT '.01' = 0.01;
          -> 1
  ```

- `<=>`

  `NULL`-safe equal. This operator performs an equality comparison like the `=` operator, but returns `1` rather than `NULL` if both operands are `NULL`, and `0` rather than `NULL` if one operand is `NULL`.

  ```
  mysql> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
          -> 1, 1, 0
  mysql> SELECT 1 = 1, NULL = NULL, 1 = NULL;
          -> 1, NULL, NULL
  ```

  `<=>` was added in MySQL 3.23.0.

- `<>`, `!=`

  Not equal:

```
mysql> SELECT '.01' <> '0.01';
        -> 1
mysql> SELECT .01 <> '0.01';
        -> 0
mysql> SELECT 'zapp' <> 'zappp';
        -> 1
```

- `<=`

  Less than or equal:

```
mysql> SELECT 0.1 <= 2;
        -> 1
```

- `<`

  Less than:

```
mysql> SELECT 2 < 2;
        -> 0
```

- `>=`

  Greater than or equal:

```
mysql> SELECT 2 >= 2;
        -> 1
```

- `>`

  Greater than:

```
mysql> SELECT 2 > 2;
        -> 0
```

- `IS boolean_value`, `IS NOT boolean_value`

  Tests whether a value against a boolean value, where `boolean_value` can be `TRUE`, `FALSE`, or `UNKNOWN`.

```
mysql> SELECT 1 IS TRUE, 0 IS FALSE, NULL IS UNKNOWN;
        -> 1, 1, 1
mysql> SELECT 1 IS NOT UNKNOWN, 0 IS NOT UNKNOWN, NULL IS NOT UNKNOWN;
        -> 1, 1, 0
```

  `IS [NOT] boolean_value` syntax was added in MySQL 5.0.2.

- `IS NULL`, `IS NOT NULL`

  Tests whether a value is or is not `NULL`.

```
mysql> SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL;
        -> 0, 0, 1
mysql> SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;
        -> 1, 1, 0
```

  To be able to work well with ODBC programs, MySQL supports the following extra features when using `IS NULL`:

  - You can find the row that contains the most recent `AUTO_INCREMENT` value by issuing a statement of the following form immediately after generating the value:

    `SELECT * FROM tbl_name WHERE auto_col IS NULL`

    This behavior can be disabled by setting `SQL_AUTO_IS_NULL=0`. See Section 13.5.3, “`SET` Syntax”.

  - For `DATE` and `DATETIME` columns that are declared as `NOT NULL`, you can find the special date `'0000-00-00'` by using a statement like this:

    `SELECT * FROM tbl_name WHERE date_column IS NULL`

    This is needed to get some ODBC applications to work because ODBC doesn't support a `'0000-00-00'` date value.

- `expr BETWEEN min AND max`

  If `expr` is greater than or equal to `min` and `expr` is less than or equal to `max`, `BETWEEN` returns `1`, otherwise it returns `0`. This is equivalent to the expression `(min <= expr AND expr <= max)` if all the arguments are of the same type. Otherwise type conversion takes place according to

the rules described at the beginning of this section, but applied to all the three arguments. **Note**: Before MySQL 4.0.5, arguments were converted to the type of *expr* instead.

```
mysql> SELECT 1 BETWEEN 2 AND 3;
        -> 0
mysql> SELECT 'b' BETWEEN 'a' AND 'c';
        -> 1
mysql> SELECT 2 BETWEEN 2 AND '3';
        -> 1
mysql> SELECT 2 BETWEEN 2 AND 'x-3';
        -> 0
```

- *expr* NOT BETWEEN *min* AND *max*

  This is the same as NOT (*expr* BETWEEN *min* AND *max*).

- COALESCE(*value*,...)

  Returns the first non-NULL value in the list.

```
mysql> SELECT COALESCE(NULL,1);
        -> 1
mysql> SELECT COALESCE(NULL,NULL,NULL);
        -> NULL
```

  COALESCE() was added in MySQL 3.23.3.

- GREATEST(*value1*,*value2*,...)

  With two or more arguments, returns the largest (maximum-valued) argument. The arguments are compared using the same rules as for LEAST().

```
mysql> SELECT GREATEST(2,0);
        -> 2
mysql> SELECT GREATEST(34.0,3.0,5.0,767.0);
        -> 767.0
mysql> SELECT GREATEST('B','A','C');
        -> 'C'
```

  Before MySQL 3.22.5, you can use MAX() instead of GREATEST().

- *expr* IN (*value*,...)

  Returns 1 if *expr* is any of the values in the IN list, else returns 0. If all values are constants, they are evaluated according to the type of *expr* and sorted. The search for the item then is done using a

binary search. This means IN is very quick if the IN value list consists entirely of constants. If *expr* is a case-sensitive string expression, the string comparison is performed in case-sensitive fashion.

```
mysql> SELECT 2 IN (0,3,5,'wefwf');
        -> 0
mysql> SELECT 'wefwf' IN (0,3,5,'wefwf');
        -> 1
```

The number of values in the IN list is only limited by the max_allowed_packet value.

To comply with the SQL standard, from MySQL 4.1 on IN returns NULL not only if the expression on the left hand side is NULL, but also if no match is found in the list and one of the expressions in the list is NULL.

From MySQL 4.1 on, IN() syntax also is used to write certain types of subqueries. See Section 13.1.8.3, "Subqueries with ANY, IN, and SOME".

- *expr* NOT IN (*value*,...)

  This is the same as NOT (*expr* IN (*value*,...)).

- ISNULL(*expr*)

  If *expr* is NULL, ISNULL() returns 1, otherwise it returns 0.

```
mysql> SELECT ISNULL(1+1);
        -> 0
mysql> SELECT ISNULL(1/0);
        -> 1
```

  A comparison of NULL values using = is always false.

- INTERVAL(*N*,*N1*,*N2*,*N3*,...)

  Returns 0 if *N* < *N1*, 1 if *N* < *N2* and so on or -1 if *N* is NULL. All arguments are treated as integers. It is required that *N1* < *N2* < *N3* < ... < *Nn* for this function to work correctly. This is because a binary search is used (very fast).

```
mysql> SELECT INTERVAL(23, 1, 15, 17, 30, 44, 200);
        -> 3
mysql> SELECT INTERVAL(10, 1, 10, 100, 1000);
        -> 2
mysql> SELECT INTERVAL(22, 23, 30, 44, 200);
        -> 0
```

- `LEAST(value1,value2,...)`

  With two or more arguments, returns the smallest (minimum-valued) argument. The arguments are compared using the following rules.

  - If the return value is used in an `INTEGER` context or all arguments are integer-valued, they are compared as integers.
  - If the return value is used in a `REAL` context or all arguments are real-valued, they are compared as reals.
  - If any argument is a case-sensitive string, the arguments are compared as case-sensitive strings.
  - In other cases, the arguments are compared as case-insensitive strings.

```
mysql> SELECT LEAST(2,0);
        -> 0
mysql> SELECT LEAST(34.0,3.0,5.0,767.0);
        -> 3.0
mysql> SELECT LEAST('B','A','C');
        -> 'A'
```

  Before MySQL 3.22.5, you can use `MIN()` instead of `LEAST()`.

  Note that the preceding conversion rules can produce strange results in some borderline cases:

```
mysql> SELECT CAST(LEAST(3600, 9223372036854775808.0) as SIGNED);
        -> -9223372036854775808
```

  This happens because MySQL reads `9223372036854775808.0` in an integer context. The integer representation is not good enough to hold the value, so it wraps to a signed integer.

12.1.4. Logical Operators

In SQL, all logical operators evaluate to TRUE, FALSE, or `NULL` (UNKNOWN). In MySQL, these are implemented as `1` (TRUE), `0` (FALSE), and `NULL`. Most of this is common to different SQL database servers, although some servers may return any non-zero value for TRUE.

- `NOT` , `!`

  Logical NOT. Evaluates to `1` if the operand is `0`, to `0` if the operand is non-zero, and `NOT NULL` returns `NULL`.

```
mysql> SELECT NOT 10;
        -> 0
mysql> SELECT NOT 0;
        -> 1
mysql> SELECT NOT NULL;
        -> NULL
mysql> SELECT ! (1+1);
        -> 0
```

```
mysql> SELECT ! 1+1;
        -> 1
```

The last example produces `1` because the expression evaluates the same way as `(!1)+1`.

- `AND`, `&&`

  Logical AND. Evaluates to `1` if all operands are non-zero and not `NULL`, to `0` if one or more operands are `0`, otherwise `NULL` is returned.

```
mysql> SELECT 1 && 1;
        -> 1
mysql> SELECT 1 && 0;
        -> 0
mysql> SELECT 1 && NULL;
        -> NULL
mysql> SELECT 0 && NULL;
        -> 0
mysql> SELECT NULL && 0;
        -> 0
```

  Please note that MySQL versions prior to 4.0.5 stop evaluation when a `NULL` is encountered, rather than continuing the process to check for possible `0` values. This means that in these versions, `SELECT (NULL AND 0)` returns `NULL` instead of `0`. As of MySQL 4.0.5, the code has been re-engineered so that the result is always as prescribed by the SQL standards while still using the optimization wherever possible.

- `OR`, `||`

  Logical OR. When both operands are non-`NULL`, the result is `1` if any operand is non-zero, and `0` otherwise. With a `NULL` operand, the result is `1` if the other operand is non-zero, and `NULL` otherwise. If both operands are `NULL`, the result is `NULL`.

```
mysql> SELECT 1 || 1;
        -> 1
mysql> SELECT 1 || 0;
        -> 1
mysql> SELECT 0 || 0;
        -> 0
mysql> SELECT 0 || NULL;
        -> NULL
mysql> SELECT 1 || NULL;
        -> 1
```

- `XOR`

  Logical XOR. Returns `NULL` if either operand is `NULL`. For non-`NULL` operands, evaluates to `1` if an odd number of operands is non-zero, otherwise `0` is returned.

```
mysql> SELECT 1 XOR 1;
        -> 0
mysql> SELECT 1 XOR 0;
        -> 1
mysql> SELECT 1 XOR NULL;
        -> NULL
mysql> SELECT 1 XOR 1 XOR 1;
        -> 1
```

`a XOR b` is mathematically equal to `(a AND (NOT b)) OR ((NOT a) and b)`.

`XOR` was added in MySQL 4.0.2.

12.2. Control Flow Functions

- `CASE` *`value`* `WHEN [`*`compare-value`*`] THEN` *`result`* `[WHEN [`*`compare-value`*`] THEN` *`result`* `...] [ELSE` *`result`*`] END`, `CASE WHEN [`*`condition`*`] THEN` *`result`* `[WHEN [`*`condition`*`] THEN` *`result`* `...] [ELSE` *`result`*`] END`

  The first version returns the *`result`* where *`value=compare-value`*. The second version returns the result for the first condition that is true. If there was no matching result value, the result after `ELSE` is returned, or `NULL` if there is no `ELSE` part.

```
mysql> SELECT CASE 1 WHEN 1 THEN 'one'
    ->     WHEN 2 THEN 'two' ELSE 'more' END;
        -> 'one'
mysql> SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
        -> 'true'
mysql> SELECT CASE BINARY 'B'
    ->     WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;
        -> NULL
```

  The type of the return value (`INTEGER`, `DOUBLE`, or `STRING`) is the same as the type of the first returned value (the expression after the first `THEN`).

  `CASE` was added in MySQL 3.23.3.

- `IF(`*`expr1`*`,`*`expr2`*`,`*`expr3`*`)`

  If *`expr1`* is TRUE (*`expr1`* `<> 0` and *`expr1`* `<> NULL`) then `IF()` returns *`expr2`*, else it returns *`expr3`*. `IF()` returns a numeric or string value, depending on the context in which it is used.

```
mysql> SELECT IF(1>2,2,3);
        -> 3
mysql> SELECT IF(1<2,'yes','no');
        -> 'yes'
mysql> SELECT IF(STRCMP('test','test1'),'no','yes');
        -> 'no'
```

If only one of `expr2` or `expr3` is explicitly `NULL`, the result type of the `IF()` function is the type of non-`NULL` expression. (This behavior is new in MySQL 4.0.3.)

`expr1` is evaluated as an integer value, which means that if you are testing floating-point or string values, you should do so using a comparison operation.

```
mysql> SELECT IF(0.1,1,0);
        -> 0
mysql> SELECT IF(0.1<>0,1,0);
        -> 1
```

In the first case shown, `IF(0.1)` returns `0` because `0.1` is converted to an integer value, resulting in a test of `IF(0)`. This may not be what you expect. In the second case, the comparison tests the original floating-point value to see whether it is non-zero. The result of the comparison is used as an integer.

The default return type of `IF()` (which may matter when it is stored into a temporary table) is calculated in MySQL 3.23 as follows:

Expression	Return Value
`expr2` or `expr3` returns a string	string
`expr2` or `expr3` returns a floating-point value	floating-point
`expr2` or `expr3` returns an integer	integer

If `expr2` and `expr3` are strings, the result is case sensitive if either string is case sensitive (starting from MySQL 3.23.51).

- `IFNULL(`*`expr1`*`,`*`expr2`*`)`

  If *`expr1`* is not `NULL`, `IFNULL()` returns *`expr1`*, else it returns *`expr2`*. `IFNULL()` returns a numeric or string value, depending on the context in which it is used.

```
mysql> SELECT IFNULL(1,0);
        -> 1
mysql> SELECT IFNULL(NULL,10);
        -> 10
mysql> SELECT IFNULL(1/0,10);
        -> 10
mysql> SELECT IFNULL(1/0,'yes');
        -> 'yes'
```

In MySQL 4.0.6 and above, the default result value of `IFNULL(`*`expr1`*`,`*`expr2`*`)` is the more ``general'' of the two expressions, in the order `STRING`, `REAL`, or `INTEGER`. The difference from earlier MySQL versions is mostly notable when you create a table based on expressions or MySQL has to internally store a value from `IFNULL()` in a temporary table.

```
CREATE TABLE tmp SELECT IFNULL(1,'test') AS test;
```

As of MySQL 4.0.6, the type for the `test` column is `CHAR(4)`, whereas in earlier versions the type would be `BIGINT`.

- `NULLIF(expr1,expr2)`

  Returns `NULL` if `expr1 = expr2` is true, else returns `expr1`. This is the same as `CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END`.

```
mysql> SELECT NULLIF(1,1);
        -> NULL
mysql> SELECT NULLIF(1,2);
        -> 1
```

  Note that MySQL evaluates `expr1` twice if the arguments are not equal.

  `NULLIF()` was added in MySQL 3.23.15.

12.3. String Functions

String-valued functions return `NULL` if the length of the result would be greater than the value of the `max_allowed_packet` system variable. See Section 7.5.2, “Tuning Server Parameters”.

For functions that operate on string positions, the first position is numbered 1.

- `ASCII(str)`

  Returns the numeric value of the leftmost character of the string `str`. Returns `0` if `str` is the empty string. Returns `NULL` if `str` is `NULL`. `ASCII()` works for characters with numeric values from `0` to `255`.

```
mysql> SELECT ASCII('2');
        -> 50
mysql> SELECT ASCII(2);
        -> 50
mysql> SELECT ASCII('dx');
        -> 100
```

  See also the `ORD()` function.

- `BIN(N)`

  Returns a string representation of the binary value of `N`, where `N` is a longlong (`BIGINT`) number. This is equivalent to `CONV(N,10,2)`. Returns `NULL` if `N` is `NULL`.

```
mysql> SELECT BIN(12);
        -> '1100'
```

- `BIT_LENGTH(str)`

  Returns the length of the string `str` in bits.

```
mysql> SELECT BIT_LENGTH('text');
        -> 32
```

  `BIT_LENGTH()` was added in MySQL 4.0.2.

- `CHAR(N,...)`

  `CHAR()` interprets the arguments as integers and returns a string consisting of the characters given by the code values of those integers. `NULL` values are skipped.

```
mysql> SELECT CHAR(77,121,83,81,'76');
        -> 'MySQL'
mysql> SELECT CHAR(77,77.3,'77.3');
        -> 'MMM'
```

- `CHAR_LENGTH(str)`

  Returns the length of the string `str`, measured in characters. A multi-byte character counts as a single character. This means that for a string containing five two-byte characters, `LENGTH()` returns `10`, whereas `CHAR_LENGTH()` returns `5`.

- `CHARACTER_LENGTH(str)`

  `CHARACTER_LENGTH()` is a synonym for `CHAR_LENGTH()`.

- `COMPRESS(string_to_compress)`

  Compresses a string. This function requires MySQL to have been compiled with a compression library such as `zlib`. Otherwise, the return value is always `NULL`. The compressed string can be uncompressed with `UNCOMPRESS()`.

```
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',1000)));
        -> 21
mysql> SELECT LENGTH(COMPRESS(''));
```

```
        -> 0
mysql> SELECT LENGTH(COMPRESS('a'));
        -> 13
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',16)));
        -> 15
```

The compressed string contents are stored the following way:

- Empty strings are stored as empty strings.
- Non-empty strings are stored as a four-byte length of the uncompressed string (low byte first), followed by the compressed string. If the string ends with space, an extra `'.'` character is added to avoid problems with endspace trimming should the result be stored in a `CHAR` or `VARCHAR` column. (Use of `CHAR` or `VARCHAR` to store compressed strings is not recommended. It is better to use a `BLOB` column instead.)

`COMPRESS()` was added in MySQL 4.1.1.

- `CONCAT(str1,str2,...)`

  Returns the string that results from concatenating the arguments. Returns `NULL` if any argument is `NULL`. May have one or more arguments. If all arguments are non-binary strings, the result is a non-binary string. If the arguments include any binary strings, the result is a binary string. A numeric argument is converted to its equivalent binary string form.

```
mysql> SELECT CONCAT('My', 'S', 'QL');
        -> 'MySQL'
mysql> SELECT CONCAT('My', NULL, 'QL');
        -> NULL
mysql> SELECT CONCAT(14.3);
        -> '14.3'
```

- `CONCAT_WS(separator,str1,str2,...)`

  `CONCAT_WS()` stands for CONCAT With Separator and is a special form of `CONCAT()`. The first argument is the separator for the rest of the arguments. The separator is added between the strings to be concatenated. The separator can be a string as can the rest of the arguments. If the separator is `NULL`, the result is `NULL`. The function skips any `NULL` values after the separator argument.

```
mysql> SELECT CONCAT_WS(',','First name','Second name','Last Name');
        -> 'First name,Second name,Last Name'
mysql> SELECT CONCAT_WS(',','First name',NULL,'Last Name');
        -> 'First name,Last Name'
```

  Before MySQL 4.0.14, `CONCAT_WS()` skips empty strings as well as `NULL` values.

- `CONV(N,from_base,to_base)`

Converts numbers between different number bases. Returns a string representation of the number *N*, converted from base *from_base* to base *to_base*. Returns NULL if any argument is NULL. The argument *N* is interpreted as an integer, but may be specified as an integer or a string. The minimum base is 2 and the maximum base is 36. If *to_base* is a negative number, *N* is regarded as a signed number. Otherwise, *N* is treated as unsigned. CONV() works with 64-bit precision.

```
mysql> SELECT CONV('a',16,2);
        -> '1010'
mysql> SELECT CONV('6E',18,8);
        -> '172'
mysql> SELECT CONV(-17,10,-18);
        -> '-H'
mysql> SELECT CONV(10+'10'+'10'+0xa,10,10);
        -> '40'
```

- ELT(*N*,*str1*,*str2*,*str3*,...)

  Returns *str1* if *N* = 1, *str2* if *N* = 2, and so on. Returns NULL if *N* is less than 1 or greater than the number of arguments. ELT() is the complement of FIELD().

```
mysql> SELECT ELT(1, 'ej', 'Heja', 'hej', 'foo');
        -> 'ej'
mysql> SELECT ELT(4, 'ej', 'Heja', 'hej', 'foo');
        -> 'foo'
```

- EXPORT_SET(*bits*,*on*,*off*[,*separator*[,*number_of_bits*]])

  Returns a string in which for every bit set in the value *bits*, you get an *on* string and for every reset bit you get an *off* string. Bits in *bits* are examined from right to left (from low-order to high-order bits). Strings are added to the result from left to right, separated by the *separator* string (default ','). The number of bits examined is given by *number_of_bits* (default 64).

```
mysql> SELECT EXPORT_SET(5,'Y','N',',',4);
        -> 'Y,N,Y,N'
mysql> SELECT EXPORT_SET(6,'1','0',',',10);
        -> '0,1,1,0,0,0,0,0,0,0'
```

- FIELD(*str*,*str1*,*str2*,*str3*,...)

  Returns the index of *str* in the *str1*, *str2*, *str3*, ... list. Returns 0 if *str* is not found. If *str* is NULL, the return value is 0 because NULL fails equality comparison with any value. FIELD() is the complement of ELT().

```
mysql> SELECT FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
```

```
        -> 2
mysql> SELECT FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo');
        -> 0
```

- `FIND_IN_SET(str,strlist)`

  Returns a value `1` to *`N`* if the string *`str`* is in the string list *`strlist`* consisting of *`N`* substrings. A string list is a string composed of substrings separated by `','` characters. If the first argument is a constant string and the second is a column of type `SET`, the `FIND_IN_SET()` function is optimized to use bit arithmetic. Returns `0` if *`str`* is not in *`strlist`* or if *`strlist`* is the empty string. Returns `NULL` if either argument is `NULL`. This function does not work properly if the first argument contains a comma (`','`) character.

```
mysql> SELECT FIND_IN_SET('b','a,b,c,d');
        -> 2
```

- `HEX(N_or_S)`

  If *`N_OR_S`* is a number, returns a string representation of the hexadecimal value of *`N`*, where *`N`* is a longlong (`BIGINT`) number. This is equivalent to `CONV(N,10,16)`.

  From MySQL 4.0.1 and up, if *`N_OR_S`* is a string, returns a hexadecimal string of *`N_OR_S`* where each character in *`N_OR_S`* is converted to two hexadecimal digits.

```
mysql> SELECT HEX(255);
        -> 'FF'
mysql> SELECT 0x616263;
        -> 'abc'
mysql> SELECT HEX('abc');
        -> 616263
```

- `INSERT(str,pos,len,newstr)`

  Returns the string *`str`*, with the substring beginning at position *`pos`* and *`len`* characters long replaced by the string *`newstr`*. Returns the original string if *`pos`* is not within the length of the string. Replaces the rest of the string from position *`pos`* is *`len`* is not within the length of the rest of the string. Returns `NULL` if any argument is null.

```
mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
        -> 'QuWhattic'
mysql> SELECT INSERT('Quadratic', -1, 4, 'What');
        -> 'Quadratic'
mysql> SELECT INSERT('Quadratic', 3, 100, 'What');
        -> 'QuWhat'
```

This function is multi-byte safe.

- `INSTR(str,substr)`

  Returns the position of the first occurrence of substring *substr* in string *str*. This is the same as the two-argument form of `LOCATE()`, except that the arguments are swapped.

```
mysql> SELECT INSTR('foobarbar', 'bar');
        -> 4
mysql> SELECT INSTR('xbar', 'foobar');
        -> 0
```

  This function is multi-byte safe. In MySQL 3.23, this function is case sensitive. For 4.0 on, it is case sensitive only if either argument is a binary string.

- `LCASE(str)`

  `LCASE()` is a synonym for `LOWER()`.

- `LEFT(str,len)`

  Returns the leftmost *len* characters from the string *str*.

```
mysql> SELECT LEFT('foobarbar', 5);
        -> 'fooba'
```

- `LENGTH(str)`

  Returns the length of the string *str*, measured in bytes. A multi-byte character counts as multiple bytes. This means that for a string containing five two-byte characters, `LENGTH()` returns `10`, whereas `CHAR_LENGTH()` returns `5`.

```
mysql> SELECT LENGTH('text');
        -> 4
```

- `LOAD_FILE(file_name)`

  Reads the file and returns the file contents as a string. The file must be located on the server, you must specify the full pathname to the file, and you must have the `FILE` privilege. The file must be readable by all and be smaller than `max_allowed_packet` bytes.

If the file doesn't exist or cannot be read because one of the preceding conditions is not satisfied, the function returns `NULL`.

```
mysql> UPDATE tbl_name
           SET blob_column=LOAD_FILE('/tmp/picture')
           WHERE id=1;
```

Before MySQL 3.23, you must read the file inside your application and create an `INSERT` statement to update the database with the file contents. If you are using the MySQL++ library, one way to do this can be found in the MySQL++ manual, available at http://dev.mysql.com/doc/.

- `LOCATE(substr,str)`, `LOCATE(substr,str,pos)`

  The first syntax returns the position of the first occurrence of substring *`substr`* in string *`str`*. The second syntax returns the position of the first occurrence of substring *`substr`* in string *`str`*, starting at position *`pos`*. Returns `0` if *`substr`* is not in *`str`*.

```
mysql> SELECT LOCATE('bar', 'foobarbar');
        -> 4
mysql> SELECT LOCATE('xbar', 'foobar');
        -> 0
mysql> SELECT LOCATE('bar', 'foobarbar',5);
        -> 7
```

  This function is multi-byte safe. In MySQL 3.23, this function is case sensitive. For 4.0 on, it is case sensitive only if either argument is a binary string.

- `LOWER(str)`

  Returns the string *`str`* with all characters changed to lowercase according to the current character set mapping (the default is ISO-8859-1 Latin1).

```
mysql> SELECT LOWER('QUADRATICALLY');
        -> 'quadratically'
```

  This function is multi-byte safe.

- `LPAD(str,len,padstr)`

  Returns the string *`str`*, left-padded with the string *`padstr`* to a length of *`len`* characters. If *`str`* is longer than *`len`*, the return value is shortened to *`len`* characters.

```
mysql> SELECT LPAD('hi',4,'??');
```

```
        -> '??hi'
mysql> SELECT LPAD('hi',1,'??');
        -> 'h'
```

- `LTRIM(str)`

  Returns the string *str* with leading space characters removed.

```
mysql> SELECT LTRIM('  barbar');
        -> 'barbar'
```

  This function is multi-byte safe.

- `MAKE_SET(bits,str1,str2,...)`

  Returns a set value (a string containing substrings separated by `','` characters) consisting of the strings that have the corresponding bit in *bits* set. *str1* corresponds to bit 0, *str2* to bit 1, and so on. `NULL` values in *str1*, *str2*, `...` are not appended to the result.

```
mysql> SELECT MAKE_SET(1,'a','b','c');
        -> 'a'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice','world');
        -> 'hello,world'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice',NULL,'world');
        -> 'hello'
mysql> SELECT MAKE_SET(0,'a','b','c');
        -> ''
```

- `MID(str,pos,len)`

  `MID(str,pos,len)` is a synonym for `SUBSTRING(str,pos,len)`.

- `OCT(N)`

  Returns a string representation of the octal value of *N*, where *N* is a longlong (`BIGINT`)number. This is equivalent to `CONV(N,10,8)`. Returns `NULL` if *N* is `NULL`.

```
mysql> SELECT OCT(12);
        -> '14'
```

- `OCTET_LENGTH(str)`

`OCTET_LENGTH()` is a synonym for `LENGTH()`.

- `ORD(str)`

  If the leftmost character of the string *`str`* is a multi-byte character, returns the code for that character, calculated from the numeric values of its constituent bytes using this formula:

```
  (1st byte code)
+ (2nd byte code * 256)
+ (3rd byte code * 256^2) ...
```

  If the leftmost character is not a multi-byte character, `ORD()` returns the same value as the `ASCII()` function.

```
mysql> SELECT ORD('2');
        -> 50
```

- `POSITION(substr IN str)`

  `POSITION(substr IN str)` is a synonym for `LOCATE(substr,str)`.

- `QUOTE(str)`

  Quotes a string to produce a result that can be used as a properly escaped data value in an SQL statement. The string is returned surrounded by single quotes and with each instance of single quote (''), backslash ('\'), ASCII NUL, and Control-Z preceded by a backslash. If the argument is `NULL`, the return value is the word ``NULL'' without surrounding single quotes. The `QUOTE()` function was added in MySQL 4.0.3.

```
mysql> SELECT QUOTE('Don\'t!');
        -> 'Don\'t!'
mysql> SELECT QUOTE(NULL);
        -> NULL
```

- `REPEAT(str,count)`

  Returns a string consisting of the string *`str`* repeated *`count`* times. If *`count`* `<= 0`, returns an empty string. Returns `NULL` if *`str`* or *`count`* are `NULL`.

```
mysql> SELECT REPEAT('MySQL', 3);
        -> 'MySQLMySQLMySQL'
```

- `REPLACE(str,from_str,to_str)`

  Returns the string *str* with all occurrences of the string *from_str* replaced by the string *to_str*.

```
mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
        -> 'WwWwWw.mysql.com'
```

  This function is multi-byte safe.

- `REVERSE(str)`

  Returns the string *str* with the order of the characters reversed.

```
mysql> SELECT REVERSE('abc');
        -> 'cba'
```

  This function is multi-byte safe.

- `RIGHT(str,len)`

  Returns the rightmost *len* characters from the string *str*.

```
mysql> SELECT RIGHT('foobarbar', 4);
        -> 'rbar'
```

  This function is multi-byte safe.

- `RPAD(str,len,padstr)`

  Returns the string *str*, right-padded with the string *padstr* to a length of *len* characters. If *str* is longer than *len*, the return value is shortened to *len* characters.

```
mysql> SELECT RPAD('hi',5,'?');
        -> 'hi???'
mysql> SELECT RPAD('hi',1,'?');
        -> 'h'
```

  This function is multi-byte safe.

- `RTRIM(str)`

  Returns the string *str* with trailing space characters removed.

```
mysql> SELECT RTRIM('barbar   ');
        -> 'barbar'
```

  This function is multi-byte safe.

- `SOUNDEX(str)`

  Returns a soundex string from *str*. Two strings that sound almost the same should have identical soundex strings. A standard soundex string is four characters long, but the `SOUNDEX()` function returns an arbitrarily long string. You can use `SUBSTRING()` on the result to get a standard soundex string. All non-alphabetic characters are ignored in the given string. All international alphabetic characters outside the A-Z range are treated as vowels.

```
mysql> SELECT SOUNDEX('Hello');
        -> 'H400'
mysql> SELECT SOUNDEX('Quadratically');
        -> 'Q36324'
```

  Note: This function implements the original Soundex algorithm, not the more popular enhanced version (also described by D. Knuth). The difference is that original version discards vowels first and then duplicates, whereas the enhanced version discards duplicates first and then vowels.

- *expr1* `SOUNDS LIKE` *expr2*

  This is the same as `SOUNDEX(expr1) = SOUNDEX(expr2)`. It is available only in MySQL 4.1 or later.

- `SPACE(N)`

  Returns a string consisting of *N* space characters.

```
mysql> SELECT SPACE(6);
        -> '      '
```

- `SUBSTRING(str,pos)`, `SUBSTRING(str FROM pos)`, `SUBSTRING(str,pos,len)`, `SUBSTRING(str FROM pos FOR len)`

  The forms without a *len* argument return a substring from string *str* starting at position *pos*. The

forms with a *len* argument return a substring *len* characters long from string *str*, starting at position *pos*. The forms that use FROM are standard SQL syntax.

```
mysql> SELECT SUBSTRING('Quadratically',5);
        -> 'ratically'
mysql> SELECT SUBSTRING('foobarbar' FROM 4);
        -> 'barbar'
mysql> SELECT SUBSTRING('Quadratically',5,6);
        -> 'ratica'
```

This function is multi-byte safe.

- SUBSTRING_INDEX(*str*,*delim*,*count*)

  Returns the substring from string *str* before *count* occurrences of the delimiter *delim*. If *count* is positive, everything to the left of the final delimiter (counting from the left) is returned. If *count* is negative, everything to the right of the final delimiter (counting from the right) is returned.

```
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
        -> 'www.mysql'
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);
        -> 'mysql.com'
```

  This function is multi-byte safe.

- TRIM([{BOTH | LEADING | TRAILING} [*remstr*] FROM] *str*) , TRIM(*remstr* FROM] *str*)

  Returns the string *str* with all *remstr* prefixes and/or suffixes removed. If none of the specifiers BOTH, LEADING, or TRAILING is given, BOTH is assumed. If *remstr* is optional and not specified, spaces are removed.

```
mysql> SELECT TRIM('  bar   ');
        -> 'bar'
mysql> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');
        -> 'barxxx'
mysql> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
        -> 'bar'
mysql> SELECT TRIM(TRAILING 'xyz' FROM 'barxxyz');
        -> 'barx'
```

  This function is multi-byte safe.

- UCASE(*str*)

  UCASE() is a synonym for UPPER().

- `UNCOMPRESS(string_to_uncompress)`

  Uncompresses a string compressed by the `COMPRESS()` function. If the argument is not a compressed value, the result is `NULL`. This function requires MySQL to have been compiled with a compression library such as `zlib`. Otherwise, the return value is always `NULL`.

```
mysql> SELECT UNCOMPRESS(COMPRESS('any string'));
        -> 'any string'
mysql> SELECT UNCOMPRESS('any string');
        -> NULL
```

  `UNCOMPRESS()` was added in MySQL 4.1.1.

- `UNCOMPRESSED_LENGTH(compressed_string)`

  Returns the length of a compressed string before compression.

```
mysql> SELECT UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a',30)));
        -> 30
```

  `UNCOMPRESSED_LENGTH()` was added in MySQL 4.1.1.

- `UNHEX(str)`

  Does the opposite of `HEX(str)`. That is, it interprets each pair of hexadecimal digits in the argument as a number and converts it to the character represented by the number. The resulting characters are returned as a binary string.

```
mysql> SELECT UNHEX('4D7953514C');
        -> 'MySQL'
mysql> SELECT 0x4D7953514C;
        -> 'MySQL'
mysql> SELECT UNHEX(HEX('string'));
        -> 'string'
mysql> SELECT HEX(UNHEX('1267'));
        -> '1267'
```

  `UNHEX()` was added in MySQL 4.1.2.

- `UPPER(str)`

  Returns the string `str` with all characters changed to uppercase according to the current character set mapping (the default is ISO-8859-1 Latin1).

```
mysql> SELECT UPPER('Hej');
        -> 'HEJ'
```

This function is multi-byte safe.

12.3.1. String Comparison Functions

MySQL automatically converts numbers to strings as necessary, and vice versa.

```
mysql> SELECT 1+'1';
        -> 2
mysql> SELECT CONCAT(2,' test');
        -> '2 test'
```

If you want to convert a number to a string explicitly, use the `CAST()` or `CONCAT()` function:

```
mysql> SELECT 38.8, CAST(38.8 AS CHAR);
        -> 38.8, '38.8'
mysql> SELECT 38.8, CONCAT(38.8);
        -> 38.8, '38.8'
```

`CAST()` is preferable, but it is unavailable before MySQL 4.0.2.

If a string function is given a binary string as an argument, the resulting string is also a binary string. A number converted to a string is treated as a binary string. This affects only comparisons.

Normally, if any expression in a string comparison is case sensitive, the comparison is performed in case-sensitive fashion.

- *expr* `LIKE` *pat* `[ESCAPE '`*escape-char*`']`

  Pattern matching using SQL simple regular expression comparison. Returns `1` (TRUE) or `0` (FALSE). If either *expr* or *pat* is `NULL`, the result is `NULL`.

  The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

  With `LIKE` you can use the following two wildcard characters in the pattern:

Character	Description
`%`	Matches any number of characters, even zero characters
`_`	Matches exactly one character

```
mysql> SELECT 'David!' LIKE 'David_';
        -> 1
mysql> SELECT 'David!' LIKE '%D%v%';
        -> 1
```

To test for literal instances of a wildcard character, precede the character with the escape character. If you don't specify the `ESCAPE` character, '\' is assumed.

String	Description
`\%`	Matches one '%' character
`\_`	Matches one '_' character

```
mysql> SELECT 'David!' LIKE 'David\_';
        -> 0
mysql> SELECT 'David_' LIKE 'David\_';
        -> 1
```

To specify a different escape character, use the `ESCAPE` clause:

```
mysql> SELECT 'David_' LIKE 'David|_' ESCAPE '|';
        -> 1
```

The following two statements illustrate that string comparisons are not case sensitive unless one of the operands is a binary string:

```
mysql> SELECT 'abc' LIKE 'ABC';
        -> 1
mysql> SELECT 'abc' LIKE BINARY 'ABC';
        -> 0
```

In MySQL, `LIKE` is allowed on numeric expressions. (This is an extension to the standard SQL `LIKE`.)

```
mysql> SELECT 10 LIKE '1%';
        -> 1
```

Note: Because MySQL uses the C escape syntax in strings (for example, '\n' to represent newline), you must double any '\' that you use in your `LIKE` strings. For example, to search for '\n', specify it as '\\n'. To search for '\', specify it as '\\\\' (the backslashes are stripped once by the parser and another time when the pattern match is done, leaving a single backslash to be matched).

- *expr* `NOT LIKE` *pat* `[ESCAPE '`*escape-char*`']`

  This is the same as `NOT (`*expr* `LIKE` *pat* `[ESCAPE '`*escape-char*`']`).

- *expr* `NOT REGEXP` *pat*, *expr* `NOT RLIKE` *pat*

  This is the same as `NOT (`*expr* `REGEXP` *pat*`)`.

- *expr* REGEXP *pat*, *expr* RLIKE *pat*

  Performs a pattern match of a string expression *expr* against a pattern *pat*. The pattern can be an extended regular expression. The syntax for regular expressions is discussed in Appendix G, *MySQL Regular Expressions*. Returns 1 if *expr* matches *pat*, otherwise returns 0. If either *expr* or *pat* is NULL, the result is NULL. RLIKE is a synonym for REGEXP, provided for mSQL compatibility.

  The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

  Note: Because MySQL uses the C escape syntax in strings (for example, '\n' to represent newline), you must double any '\' that you use in your REGEXP strings.

  As of MySQL 3.23.4, REGEXP is not case sensitive for normal (not binary) strings.

```
mysql> SELECT 'Monty!' REGEXP 'm%y%%';
        -> 0
mysql> SELECT 'Monty!' REGEXP '.*';
        -> 1
mysql> SELECT 'new*\n*line' REGEXP 'new\\*.\\*line';
        -> 1
mysql> SELECT 'a' REGEXP 'A', 'a' REGEXP BINARY 'A';
        -> 1  0
mysql> SELECT 'a' REGEXP '^[a-d]';
        -> 1
```

  REGEXP and RLIKE use the current character set (ISO-8859-1 Latin1 by default) when deciding the type of a character. However, these operators are not multi-byte safe.

- STRCMP(*expr1*,*expr2*)

  STRCMP() returns 0 if the strings are the same, -1 if the first argument is smaller than the second according to the current sort order, and 1 otherwise.

```
mysql> SELECT STRCMP('text', 'text2');
        -> -1
mysql> SELECT STRCMP('text2', 'text');
        -> 1
mysql> SELECT STRCMP('text', 'text');
        -> 0
```

  As of MySQL 4.0, STRCMP() uses the current character set when performing comparisons. This makes the default comparison behavior case insensitive unless one or both of the operands are binary strings. Before MySQL 4.0, STRCMP() is case sensitive.

12.4. Numeric Functions

12.4.1. Arithmetic Operators

The usual arithmetic operators are available. Note that in the case of `-`, `+`, and `*`, the result is calculated with `BIGINT` (64-bit) precision if both arguments are integers. If one of the arguments is an unsigned integer, and the other argument is also an integer, the result is an unsigned integer. See Section 12.7, “Cast Functions and Operators”.

- `+`

  Addition:

  ```
  mysql> SELECT 3+5;
          -> 8
  ```

- `-`

  Subtraction:

  ```
  mysql> SELECT 3-5;
          -> -2
  ```

- `-`

  Unary minus. Changes the sign of the argument.

  ```
  mysql> SELECT - 2;
          -> -2
  ```

  Note that if this operator is used with a `BIGINT`, the return value is a `BIGINT`! This means that you should avoid using `-` on integers that may have the value of `-2^63`!

- `*`

  Multiplication:

  ```
  mysql> SELECT 3*5;
          -> 15
  mysql> SELECT 18014398509481984*18014398509481984.0;
          -> 324518553658426726783156020576256.0
  mysql> SELECT 18014398509481984*18014398509481984;
          -> 0
  ```

  The result of the last expression is incorrect because the result of the integer multiplication exceeds the 64-bit range of `BIGINT` calculations.

- `/`

  Division:

  ```
  mysql> SELECT 3/5;
          -> 0.60
  ```

  Division by zero produces a `NULL` result:

  ```
  mysql> SELECT 102/(1-1);
          -> NULL
  ```

  A division is calculated with `BIGINT` arithmetic only if performed in a context where its result are converted to an integer.

- `DIV`

  Integer division. Similar to `FLOOR()` but safe with `BIGINT` values.

  ```
  mysql> SELECT 5 DIV 2;
          -> 2
  ```

  `DIV` is new in MySQL 4.1.0.

12.4.2. Mathematical Functions

All mathematical functions return `NULL` in case of an error.

- `ABS(X)`

  Returns the absolute value of *X*.

  ```
  mysql> SELECT ABS(2);
          -> 2
  mysql> SELECT ABS(-32);
          -> 32
  ```

  This function is safe to use with `BIGINT` values.

- `ACOS(X)`

  Returns the arc cosine of *X*, that is, the value whose cosine is *X*. Returns `NULL` if *X* is not in the

range -1 to 1.

```
mysql> SELECT ACOS(1);
        -> 0.000000
mysql> SELECT ACOS(1.0001);
        -> NULL
mysql> SELECT ACOS(0);
        -> 1.570796
```

- ASIN(*X*)

  Returns the arc sine of *X*, that is, the value whose sine is *X*. Returns NULL if *X* is not in the range -1 to 1.

```
mysql> SELECT ASIN(0.2);
        -> 0.201358
mysql> SELECT ASIN('foo');
        -> 0.000000
```

- ATAN(*X*)

  Returns the arc tangent of *X*, that is, the value whose tangent is *X*.

```
mysql> SELECT ATAN(2);
        -> 1.107149
mysql> SELECT ATAN(-2);
        -> -1.107149
```

- ATAN(*Y*,*X*), ATAN2(*Y*,*X*)

  Returns the arc tangent of the two variables *X* and *Y*. It is similar to calculating the arc tangent of *Y* / *X*, except that the signs of both arguments are used to determine the quadrant of the result.

```
mysql> SELECT ATAN(-2,2);
        -> -0.785398
mysql> SELECT ATAN2(PI(),0);
        -> 1.570796
```

- CEILING(*X*), CEIL(*X*)

  Returns the smallest integer value not less than *X*.

```
mysql> SELECT CEILING(1.23);
        -> 2
mysql> SELECT CEIL(-1.23);
        -> -1
```

Note that the return value is converted to a `BIGINT`!

The `CEIL()` alias was added in MySQL 4.0.6.

- `COS(X)`

  Returns the cosine of *X*, where *X* is given in radians.

```
mysql> SELECT COS(PI());
        -> -1.000000
```

- `COT(X)`

  Returns the cotangent of *X*.

```
mysql> SELECT COT(12);
        -> -1.57267341
mysql> SELECT COT(0);
        -> NULL
```

- `CRC32(expr)`

  Computes a cyclic redundancy check value and returns a 32-bit unsigned value. The result is `NULL` if the argument is `NULL`. The argument is expected be a string and is treated as one if it is not.

```
mysql> SELECT CRC32('MySQL');
        -> 3259397556
```

`CRC32()` is available as of MySQL 4.1.0.

- `DEGREES(X)`

  Returns the argument *X*, converted from radians to degrees.

```
mysql> SELECT DEGREES(PI());
        -> 180.000000
```

- `EXP(X)`

  Returns the value of `e` (the base of natural logarithms) raised to the power of *X*.

```
mysql> SELECT EXP(2);
        -> 7.389056
mysql> SELECT EXP(-2);
        -> 0.135335
```

- `FLOOR(X)`

  Returns the largest integer value not greater than *X*.

```
mysql> SELECT FLOOR(1.23);
        -> 1
mysql> SELECT FLOOR(-1.23);
        -> -2
```

  Note that the return value is converted to a `BIGINT`!

- `LN(X)`

  Returns the natural logarithm of *X*.

```
mysql> SELECT LN(2);
        -> 0.693147
mysql> SELECT LN(-2);
        -> NULL
```

  This function was added in MySQL 4.0.3. It is synonymous with `LOG(X)` in MySQL.

- `LOG(X)` , `LOG(B,X)`

  If called with one parameter, this function returns the natural logarithm of *X*.

```
mysql> SELECT LOG(2);
        -> 0.693147
mysql> SELECT LOG(-2);
```

```
        -> NULL
```

If called with two parameters, this function returns the logarithm of *X* for an arbitrary base *B*.

```
mysql> SELECT LOG(2,65536);
        -> 16.000000
mysql> SELECT LOG(1,100);
        -> NULL
```

The arbitrary base option was added in MySQL 4.0.3. `LOG(B,X)` is equivalent to `LOG(X)/LOG(B)`.

- `LOG2(X)`

  Returns the base-2 logarithm of *X*.

```
mysql> SELECT LOG2(65536);
        -> 16.000000
mysql> SELECT LOG2(-100);
        -> NULL
```

`LOG2()` is useful for finding out how many bits a number would require for storage. This function was added in MySQL 4.0.3. In earlier versions, you can use `LOG(X)/LOG(2)` instead.

- `LOG10(X)`

  Returns the base-10 logarithm of *X*.

```
mysql> SELECT LOG10(2);
        -> 0.301030
mysql> SELECT LOG10(100);
        -> 2.000000
mysql> SELECT LOG10(-100);
        -> NULL
```

- `MOD(N,M)`, `N % M`, `N MOD M`

  Modulo operation. Returns the remainder of *N* divided by *M*.

```
mysql> SELECT MOD(234, 10);
        -> 4
mysql> SELECT 253 % 7;
        -> 1
mysql> SELECT MOD(29,9);
        -> 2
```

```
mysql> SELECT 29 MOD 9;
        -> 2
```

This function is safe to use with `BIGINT` values. The *N* `MOD` *M* syntax works only as of MySQL 4.1.

As of MySQL 4.1.7, `MOD()` works on values with a fractional part and returns the exact remainder after division:

```
mysql> SELECT MOD(34.5,3);
        -> 1.5
```

Before MySQL 4.1.7, `MOD()` rounds arguments with a fractional value to integers and returns an integer result:

```
mysql> SELECT MOD(34.5,3);
        -> 2
```

- `PI()`

  Returns the value of PI. The default number of decimals displayed is five, but MySQL internally uses the full double-precision value for PI.

```
mysql> SELECT PI();
        -> 3.141593
mysql> SELECT PI()+0.000000000000000000;
        -> 3.141592653589793116
```

- `POW(`*X*`,`*Y*`)`, `POWER(`*X*`,`*Y*`)`

  Returns the value of *X* raised to the power of *Y*.

```
mysql> SELECT POW(2,2);
        -> 4.000000
mysql> SELECT POW(2,-2);
        -> 0.250000
```

- `RADIANS(`*X*`)`

  Returns the argument *X*, converted from degrees to radians.

```
mysql> SELECT RADIANS(90);
```

```
        -> 1.570796
```

- `RAND()`, `RAND(N)`

  Returns a random floating-point value in the range from `0` to `1.0`. If an integer argument *N* is specified, it is used as the seed value (producing a repeatable sequence).

```
mysql> SELECT RAND();
        -> 0.9233482386203
mysql> SELECT RAND(20);
        -> 0.15888261251047
mysql> SELECT RAND(20);
        -> 0.15888261251047
mysql> SELECT RAND();
        -> 0.63553050033332
mysql> SELECT RAND();
        -> 0.70100469486881
```

  You can't use a column with `RAND()` values in an `ORDER BY` clause, because `ORDER BY` would evaluate the column multiple times. As of MySQL 3.23, you can retrieve rows in random order like this:

```
mysql> SELECT * FROM tbl_name ORDER BY RAND();
```

  `ORDER BY RAND()` combined with `LIMIT` is useful for selecting a random sample of a set of rows:

```
mysql> SELECT * FROM table1, table2 WHERE a=b AND c<d
    -> ORDER BY RAND() LIMIT 1000;
```

  Note that `RAND()` in a `WHERE` clause is re-evaluated every time the `WHERE` is executed.

  `RAND()` is not meant to be a perfect random generator, but instead a fast way to generate ad hoc random numbers that is portable between platforms for the same MySQL version.

- `ROUND(X)`, `ROUND(X,D)`

  Returns the argument *X*, rounded to the nearest integer. With two arguments, returns *X* rounded to *D* decimals. *D* can be negative to round *D* digits left of the decimal point of the value *X*.

```
mysql> SELECT ROUND(-1.23);
        -> -1
mysql> SELECT ROUND(-1.58);
        -> -2
mysql> SELECT ROUND(1.58);
        -> 2
mysql> SELECT ROUND(1.298, 1);
```

```
        -> 1.3
mysql> SELECT ROUND(1.298, 0);
        -> 1
mysql> SELECT ROUND(23.298, -1);
        -> 20
```

Note that the behavior of `ROUND()` when the argument is halfway between two integers depends on the C library implementation. Different implementations round to the nearest even number, always up, always down, or always toward zero. If you need one kind of rounding, you should use a well-defined function such as `TRUNCATE()` or `FLOOR()` instead.

- `SIGN(X)`

  Returns the sign of the argument as `-1`, `0`, or `1`, depending on whether *X* is negative, zero, or positive.

```
mysql> SELECT SIGN(-32);
        -> -1
mysql> SELECT SIGN(0);
        -> 0
mysql> SELECT SIGN(234);
        -> 1
```

- `SIN(X)`

  Returns the sine of *X*, where *X* is given in radians.

```
mysql> SELECT SIN(PI());
        -> 0.000000
```

- `SQRT(X)`

  Returns the non-negative square root of *X*.

```
mysql> SELECT SQRT(4);
        -> 2.000000
mysql> SELECT SQRT(20);
        -> 4.472136
```

- `TAN(X)`

  Returns the tangent of *X*, where *X* is given in radians.

```
mysql> SELECT TAN(PI()+1);
        -> 1.557408
```

- `TRUNCATE(X,D)`

  Returns the number *X*, truncated to *D* decimals. If *D* is `0`, the result has no decimal point or fractional part. *D* can be negative to truncate (make zero) *D* digits left of the decimal point of the value *X*.

```
mysql> SELECT TRUNCATE(1.223,1);
        -> 1.2
mysql> SELECT TRUNCATE(1.999,1);
        -> 1.9
mysql> SELECT TRUNCATE(1.999,0);
        -> 1
mysql> SELECT TRUNCATE(-1.999,1);
        -> -1.9
mysql> SELECT TRUNCATE(122,-2);
       -> 100
```

  Starting from MySQL 3.23.51, all numbers are rounded toward zero.

  Note that decimal numbers are normally not stored as exact numbers in computers, but as double-precision values, so you may be surprised by the following result:

```
mysql> SELECT TRUNCATE(10.28*100,0);
       -> 1027
```

  This happens because `10.28` is actually stored as something like `10.2799999999999999`.

12.5. Date and Time Functions

This section describes the functions that can be used to manipulate temporal values. See Section 11.3, “Date and Time Types” for a description of the range of values each date and time type has and the valid formats in which values may be specified.

Here is an example that uses date functions. The following query selects all records with a `date_col` value from within the last 30 days:

```
mysql> SELECT something FROM tbl_name
    -> WHERE DATE_SUB(CURDATE(),INTERVAL 30 DAY) <= date_col;
```

Note that the query also selects records with dates that lie in the future.

Functions that expect date values usually accept datetime values and ignore the time part. Functions that expect time values usually accept datetime values and ignore the date part.

Functions that return the current date or time each are evaluated only once per query at the start of query

execution. This means that multiple references to a function such as NOW() within a single query always produce the same result. This principle also applies to CURDATE(), CURTIME(), UTC_DATE(), UTC_TIME(), UTC_TIMESTAMP(), and to any of their synonyms.

Beginning with MySQL 4.1.3, the CURRENT_TIMESTAMP(), CURRENT_TIME(), CURRENT_DATE(), and FROM_UNIXTIME() functions return values in the connection's current time zone, which is available as the value of the time_zone system variable. Also, UNIX_TIMESTAMP() assumes that its argument is a datetime value in the current time zone. See Section 5.8.8, “MySQL Server Time Zone Support”.

The return value ranges in the following function descriptions apply for complete dates. If a date is a ``zero'' value or an incomplete date such as '2001-11-00', functions that extract a part of a date may return 0. For example, DAYOFMONTH('2001-11-00') returns 0.

- ADDDATE(*date*,INTERVAL *expr type*),ADDDATE(*expr*,*days*)

  When invoked with the INTERVAL form of the second argument, ADDDATE() is a synonym for DATE_ADD(). The related function SUBDATE() is a synonym for DATE_SUB(). For information on the INTERVAL argument, see the discussion for DATE_ADD().

```
mysql> SELECT DATE_ADD('1998-01-02', INTERVAL 31 DAY);
        -> '1998-02-02'
mysql> SELECT ADDDATE('1998-01-02', INTERVAL 31 DAY);
        -> '1998-02-02'
```

  As of MySQL 4.1.1, the second syntax is allowed, where *expr* is a date or datetime expression and *days* is the number of days to be added to *expr*.

```
mysql> SELECT ADDDATE('1998-01-02', 31);
        -> '1998-02-02'
```

- ADDTIME(*expr*,*expr2*)

  ADDTIME() adds *expr2* to *expr* and returns the result. *expr* is a time or datetime expression, and *expr2* is a time expression.

```
mysql> SELECT ADDTIME('1997-12-31 23:59:59.999999',
    ->                '1 1:1:1.000002');
        -> '1998-01-02 01:01:01.000001'
mysql> SELECT ADDTIME('01:00:00.999999', '02:00:00.999998');
        -> '03:00:01.999997'
```

  ADDTIME() was added in MySQL 4.1.1.

- CONVERT_TZ(*dt*,*from_tz*,*to_tz*)

  CONVERT_TZ() converts a datetime value *dt* from time zone given by *from_tz* to the time zone given by *to_tz* and returns the resulting value. Time zones may be specified as described in Sec-

tion 5.8.8, “MySQL Server Time Zone Support”. This function returns `NULL` if the arguments are invalid.

If the value falls out of the supported range of the `TIMESTAMP` type when converted fom *`from_tz`* to UTC, no conversion occurs. The `TIMESTAMP` range is described at Section 11.1.2, “Overview of Date and Time Types”.

```
mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','GMT','MET');
        -> '2004-01-01 13:00:00'
mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','+00:00','-07:00');
        -> '2004-01-01 05:00:00'
```

To use named time zones such as `'MET'` or `'Europe/Moscow'`, the time zone tables must be properly set up. See Section 5.8.8, “MySQL Server Time Zone Support” for instructions.

`CONVERT_TZ()` was added in MySQL 4.1.3.

- `CURDATE()`

  Returns the current date as a value in `'YYYY-MM-DD'` or `YYYYMMDD` format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT CURDATE();
        -> '1997-12-15'
mysql> SELECT CURDATE() + 0;
        -> 19971215
```

- `CURRENT_DATE` , `CURRENT_DATE()`

  `CURRENT_DATE` and `CURRENT_DATE()` are synonyms for `CURDATE()`.

- `CURTIME()`

  Returns the current time as a value in `'HH:MM:SS'` or `HHMMSS` format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT CURTIME();
        -> '23:50:26'
mysql> SELECT CURTIME() + 0;
        -> 235026
```

- `CURRENT_TIME` , `CURRENT_TIME()`

  `CURRENT_TIME` and `CURRENT_TIME()` are synonyms for `CURTIME()`.

- `CURRENT_TIMESTAMP`, `CURRENT_TIMESTAMP()`

  `CURRENT_TIMESTAMP` and `CURRENT_TIMESTAMP()` are synonyms for `NOW()`.

- `DATE(expr)`

  Extracts the date part of the date or datetime expression *expr*.

```
mysql> SELECT DATE('2003-12-31 01:02:03');
        -> '2003-12-31'
```

  `DATE()` is available as of MySQL 4.1.1.

- `DATEDIFF(expr,expr2)`

  `DATEDIFF()` returns the number of days between the start date *expr* and the end date *expr2*. *expr* and *expr2* are date or date-and-time expressions. Only the date parts of the values are used in the calculation.

```
mysql> SELECT DATEDIFF('1997-12-31 23:59:59','1997-12-30');
        -> 1
mysql> SELECT DATEDIFF('1997-11-30 23:59:59','1997-12-31');
        -> -31
```

  `DATEDIFF()` was added in MySQL 4.1.1.

- `DATE_ADD(date,INTERVAL expr type)`, `DATE_SUB(date,INTERVAL expr type)`

  These functions perform date arithmetic. *date* is a `DATETIME` or `DATE` value specifying the starting date. *expr* is an expression specifying the interval value to be added or subtracted from the starting date. *expr* is a string; it may start with a `'-'` for negative intervals. *type* is a keyword indicating how the expression should be interpreted.

  The `INTERVAL` keyword and the `type` specifier are not case sensitive.

  The following table shows how the *type* and *expr* arguments are related:

type **Value**	**Expected** *expr* **Format**
MICROSECOND	MICROSECONDS
SECOND	SECONDS
MINUTE	MINUTES
HOUR	HOURS
DAY	DAYS

WEEK	WEEKS
MONTH	MONTHS
QUARTER	QUARTERS
YEAR	YEARS
SECOND_MICROSECOND	'SECONDS.MICROSECONDS'
MINUTE_MICROSECOND	'MINUTES.MICROSECONDS'
MINUTE_SECOND	'MINUTES:SECONDS'
HOUR_MICROSECOND	'HOURS.MICROSECONDS'
HOUR_SECOND	'HOURS:MINUTES:SECONDS'
HOUR_MINUTE	'HOURS:MINUTES'
DAY_MICROSECOND	'DAYS.MICROSECONDS'
DAY_SECOND	'DAYS HOURS:MINUTES:SECONDS'
DAY_MINUTE	'DAYS HOURS:MINUTES'
DAY_HOUR	'DAYS HOURS'
YEAR_MONTH	'YEARS-MONTHS'

The *type* values DAY_MICROSECOND, HOUR_MICROSECOND, MINUTE_MICROSECOND, SECOND_MICROSECOND, and MICROSECOND are allowed as of MySQL 4.1.1. The values QUARTER and WEEK are allowed as of MySQL 5.0.0.

MySQL allows any punctuation delimiter in the *expr* format. Those shown in the table are the suggested delimiters. If the *date* argument is a DATE value and your calculations involve only YEAR, MONTH, and DAY parts (that is, no time parts), the result is a DATE value. Otherwise, the result is a DATETIME value.

As of MySQL 3.23, INTERVAL *expr type* is allowed on either side of the + operator if the expression on the other side is a date or datetime value. For the - operator, INTERVAL *expr type* is allowed only on the right side, because it makes no sense to subtract a date or datetime value from an interval. (See examples below.)

```
mysql> SELECT '1997-12-31 23:59:59' + INTERVAL 1 SECOND;
        -> '1998-01-01 00:00:00'
mysql> SELECT INTERVAL 1 DAY + '1997-12-31';
        -> '1998-01-01'
mysql> SELECT '1998-01-01' - INTERVAL 1 SECOND;
        -> '1997-12-31 23:59:59'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
    ->                 INTERVAL 1 SECOND);
        -> '1998-01-01 00:00:00'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
    ->                 INTERVAL 1 DAY);
        -> '1998-01-01 23:59:59'
mysql> SELECT DATE_ADD('1997-12-31 23:59:59',
    ->                 INTERVAL '1:1' MINUTE_SECOND);
        -> '1998-01-01 00:01:00'
mysql> SELECT DATE_SUB('1998-01-01 00:00:00',
    ->                 INTERVAL '1 1:1:1' DAY_SECOND);
        -> '1997-12-30 22:58:59'
mysql> SELECT DATE_ADD('1998-01-01 00:00:00',
    ->                 INTERVAL '-1 10' DAY_HOUR);
```

```
        -> '1997-12-30 14:00:00'
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
        -> '1997-12-02'
mysql> SELECT DATE_ADD('1992-12-31 23:59:59.000002',
    ->            INTERVAL '1.999999' SECOND_MICROSECOND);
        -> '1993-01-01 00:00:01.000001'
```

If you specify an interval value that is too short (does not include all the interval parts that would be expected from the *type* keyword), MySQL assumes that you have left out the leftmost parts of the interval value. For example, if you specify a `type` of `DAY_SECOND`, the value of *expr* is expected to have days, hours, minutes, and seconds parts. If you specify a value like `'1:10'`, MySQL assumes that the days and hours parts are missing and the value represents minutes and seconds. In other words, `'1:10' DAY_SECOND` is interpreted in such a way that it is equivalent to `'1:10' MINUTE_SECOND`. This is analogous to the way that MySQL interprets `TIME` values as representing elapsed time rather than as time of day.

If you add to or subtract from a date value something that contains a time part, the result is automatically converted to a datetime value:

```
mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 DAY);
        -> '1999-01-02'
mysql> SELECT DATE_ADD('1999-01-01', INTERVAL 1 HOUR);
        -> '1999-01-01 01:00:00'
```

If you use really malformed dates, the result is `NULL`. If you add `MONTH`, `YEAR_MONTH`, or `YEAR` and the resulting date has a day that is larger than the maximum day for the new month, the day is adjusted to the maximum days in the new month:

```
mysql> SELECT DATE_ADD('1998-01-30', INTERVAL 1 MONTH);
        -> '1998-02-28'
```

- `DATE_FORMAT(`*date*`,`*format*`)`

  Formats the *date* value according to the *format* string. The following specifiers may be used in the *format* string:

Specifier	Description
`%a`	Abbreviated weekday name (`Sun`..`Sat`)
`%b`	Abbreviated month name (`Jan`..`Dec`)
`%c`	Month, numeric (`0`..`12`)
`%D`	Day of the month with English suffix (`0th`, `1st`, `2nd`, `3rd`, ...)
`%d`	Day of the month, numeric (`00`..`31`)
`%e`	Day of the month, numeric (`0`..`31`)
`%f`	Microseconds (`000000`..`999999`)
`%H`	Hour (`00`..`23`)
`%h`	Hour (`01`..`12`)
`%I`	Hour (`01`..`12`)
`%i`	Minutes, numeric (`00`..`59`)

%j	Day of year (001..366)
%k	Hour (0..23)
%l	Hour (1..12)
%M	Month name (January..December)
%m	Month, numeric (00..12)
%p	AM or PM
%r	Time, 12-hour (hh:mm:ss followed by AM or PM)
%S	Seconds (00..59)
%s	Seconds (00..59)
%T	Time, 24-hour (hh:mm:ss)
%U	Week (00..53), where Sunday is the first day of the week
%u	Week (00..53), where Monday is the first day of the week
%V	Week (01..53), where Sunday is the first day of the week; used with %X
%v	Week (01..53), where Monday is the first day of the week; used with %x
%W	Weekday name (Sunday..Saturday)
%w	Day of the week (0=Sunday..6=Saturday)
%X	Year for the week where Sunday is the first day of the week, numeric, four digits; used with %V
%x	Year for the week, where Monday is the first day of the week, numeric, four digits; used with %v
%Y	Year, numeric, four digits
%y	Year, numeric, two digits
%%	A literal '%'.

All other characters are copied to the result without interpretation.

The %v, %V, %x, and %X format specifiers are available as of MySQL 3.23.8. %f is available as of MySQL 4.1.1.

As of MySQL 3.23, the '%' character is required before format specifier characters. In earlier versions of MySQL, '%' was optional.

The reason the ranges for the month and day specifiers begin with zero is that MySQL allows incomplete dates such as '2004-00-00' to be stored as of MySQL 3.23.

```
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%W %M %Y');
        -> 'Saturday October 1997'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00', '%H:%i:%s');
        -> '22:23:00'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
                          '%D %y %a %d %m %b %j');
        -> '4th 97 Sat 04 10 Oct 277'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
                          '%H %k %I %r %T %S %w');
        -> '22 22 10 10:23:00 PM 22:23:00 00 6'
mysql> SELECT DATE_FORMAT('1999-01-01', '%X %V');
        -> '1998 52'
```

- `DAY(date)`

  `DAY()` is a synonym for `DAYOFMONTH()`. It is available as of MySQL 4.1.1.

- `DAYNAME(date)`

  Returns the name of the weekday for *`date`*.

```
mysql> SELECT DAYNAME('1998-02-05');
        -> 'Thursday'
```

- `DAYOFMONTH(date)`

  Returns the day of the month for *`date`*, in the range `1` to `31`.

```
mysql> SELECT DAYOFMONTH('1998-02-03');
        -> 3
```

- `DAYOFWEEK(date)`

  Returns the weekday index for *`date`* (`1` = Sunday, `2` = Monday, ..., `7` = Saturday). These index values correspond to the ODBC standard.

```
mysql> SELECT DAYOFWEEK('1998-02-03');
        -> 3
```

- `DAYOFYEAR(date)`

  Returns the day of the year for *`date`*, in the range `1` to `366`.

```
mysql> SELECT DAYOFYEAR('1998-02-03');
        -> 34
```

- `EXTRACT(type FROM date)`

The EXTRACT() function uses the same kinds of interval type specifiers as DATE_ADD() or DATE_SUB(), but extracts parts from the date rather than performing date arithmetic.

```
mysql> SELECT EXTRACT(YEAR FROM '1999-07-02');
       -> 1999
mysql> SELECT EXTRACT(YEAR_MONTH FROM '1999-07-02 01:02:03');
       -> 199907
mysql> SELECT EXTRACT(DAY_MINUTE FROM '1999-07-02 01:02:03');
       -> 20102
mysql> SELECT EXTRACT(MICROSECOND
    ->                FROM '2003-01-02 10:30:00.00123');
        -> 123
```

EXTRACT() was added in MySQL 3.23.0.

- FROM_DAYS(*N*)

  Given a daynumber *N*, returns a DATE value.

```
mysql> SELECT FROM_DAYS(729669);
        -> '1997-10-07'
```

  FROM_DAYS() is not intended for use with values that precede the advent of the Gregorian calendar (1582), because it does not take into account the days that were lost when the calendar was changed.

- FROM_UNIXTIME(*unix_timestamp*), FROM_UNIXTIME(*unix_timestamp*,*format*)

  Returns a representation of the *unix_timestamp* argument as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT FROM_UNIXTIME(875996580);
        -> '1997-10-04 22:23:00'
mysql> SELECT FROM_UNIXTIME(875996580) + 0;
        -> 19971004222300
```

  If *format* is given, the result is formatted according to the *format* string. *format* may contain the same specifiers as those listed in the entry for the DATE_FORMAT() function.

```
mysql> SELECT FROM_UNIXTIME(UNIX_TIMESTAMP(),
    ->                      '%Y %D %M %h:%i:%s %x');
        -> '2003 6th August 06:22:58 2003'
```

- `GET_FORMAT(DATE|TIME|DATETIME, 'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL')`

  Returns a format string. This function is useful in combination with the `DATE_FORMAT()` and the `STR_TO_DATE()` functions.

  The three possible values for the first argument and the five possible values for the second argument result in 15 possible format strings (for the specifiers used, see the table in the `DATE_FORMAT()` function description).

Function Call	Result
`GET_FORMAT(DATE,'USA')`	`'%m.%d.%Y'`
`GET_FORMAT(DATE,'JIS')`	`'%Y-%m-%d'`
`GET_FORMAT(DATE,'ISO')`	`'%Y-%m-%d'`
`GET_FORMAT(DATE,'EUR')`	`'%d.%m.%Y'`
`GET_FORMAT(DATE,'INTERNAL')`	`'%Y%m%d'`
`GET_FORMAT(DATETIME,'USA')`	`'%Y-%m-%d-%H.%i.%s'`
`GET_FORMAT(DATETIME,'JIS')`	`'%Y-%m-%d %H:%i:%s'`
`GET_FORMAT(DATETIME,'ISO')`	`'%Y-%m-%d %H:%i:%s'`
`GET_FORMAT(DATETIME,'EUR')`	`'%Y-%m-%d-%H.%i.%s'`
`GET_FORMAT(DATETIME,'INTERNAL')`	`'%Y%m%d%H%i%s'`
`GET_FORMAT(TIME,'USA')`	`'%h:%i:%s %p'`
`GET_FORMAT(TIME,'JIS')`	`'%H:%i:%s'`
`GET_FORMAT(TIME,'ISO')`	`'%H:%i:%s'`
`GET_FORMAT(TIME,'EUR')`	`'%H.%i.%S'`
`GET_FORMAT(TIME,'INTERNAL')`	`'%H%i%s'`

  ISO format is ISO 9075, not ISO 8601.

  As of MySQL 4.1.4, `TIMESTAMP` can also be used; `GET_FORMAT()` returns the same values as for `DATETIME`.

```
mysql> SELECT DATE_FORMAT('2003-10-03',GET_FORMAT(DATE,'EUR'));
         -> '03.10.2003'
mysql> SELECT STR_TO_DATE('10.31.2003',GET_FORMAT(DATE,'USA'));
         -> 2003-10-31
```

  `GET_FORMAT()` is available as of MySQL 4.1.1. See Section 13.5.3, “`SET` Syntax”.

- `HOUR(time)`

  Returns the hour for *time*. The range of the return value is `0` to `23` for time-of-day values.

  ```
  mysql> SELECT HOUR('10:05:03');
          -> 10
  ```

  However, the range of `TIME` values actually is much larger, so `HOUR` can return values greater than `23`.

  ```
  mysql> SELECT HOUR('272:59:59');
          -> 272
  ```

- `LAST_DAY(date)`

  Takes a date or datetime value and returns the corresponding value for the last day of the month. Returns `NULL` if the argument is invalid.

  ```
  mysql> SELECT LAST_DAY('2003-02-05');
          -> '2003-02-28'
  mysql> SELECT LAST_DAY('2004-02-05');
          -> '2004-02-29'
  mysql> SELECT LAST_DAY('2004-01-01 01:01:01');
          -> '2004-01-31'
  mysql> SELECT LAST_DAY('2003-03-32');
          -> NULL
  ```

  `LAST_DAY()` is available as of MySQL 4.1.1.

- `LOCALTIME`, `LOCALTIME()`

  `LOCALTIME` and `LOCALTIME()` are synonyms for `NOW()`.

  They were added in MySQL 4.0.6.

- `LOCALTIMESTAMP`, `LOCALTIMESTAMP()`

  `LOCALTIMESTAMP` and `LOCALTIMESTAMP()` are synonyms for `NOW()`.

  They were added in MySQL 4.0.6.

- `MAKEDATE(year,dayofyear)`

Returns a date, given year and day-of-year values. *dayofyear* must be greater than 0 or the result is NULL.

```
mysql> SELECT MAKEDATE(2001,31), MAKEDATE(2001,32);
        -> '2001-01-31', '2001-02-01'
mysql> SELECT MAKEDATE(2001,365), MAKEDATE(2004,365);
        -> '2001-12-31', '2004-12-30'
mysql> SELECT MAKEDATE(2001,0);
        -> NULL
```

MAKEDATE() is available as of MySQL 4.1.1.

- MAKETIME(*hour*,*minute*,*second*)

  Returns a time value calculated from the *hour*, *minute*, and *second* arguments.

```
mysql> SELECT MAKETIME(12,15,30);
        -> '12:15:30'
```

MAKETIME() is available as of MySQL 4.1.1.

- MICROSECOND(*expr*)

  Returns the microseconds from the time or datetime expression *expr* as a number in the range from 0 to 999999.

```
mysql> SELECT MICROSECOND('12:00:00.123456');
        -> 123456
mysql> SELECT MICROSECOND('1997-12-31 23:59:59.000010');
        -> 10
```

MICROSECOND() is available as of MySQL 4.1.1.

- MINUTE(*time*)

  Returns the minute for *time*, in the range 0 to 59.

```
mysql> SELECT MINUTE('98-02-03 10:05:03');
        -> 5
```

- MONTH(*date*)

Returns the month for *date*, in the range 1 to 12.

```
mysql> SELECT MONTH('1998-02-03');
        -> 2
```

- MONTHNAME(*date*)

  Returns the full name of the month for *date*.

```
mysql> SELECT MONTHNAME('1998-02-05');
        -> 'February'
```

- NOW()

  Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDH-HMMSS format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT NOW();
        -> '1997-12-15 23:50:26'
mysql> SELECT NOW() + 0;
        -> 19971215235026
```

- PERIOD_ADD(*P*,*N*)

  Adds *N* months to period *P* (in the format YYMM or YYYYMM). Returns a value in the format YYYYMM. Note that the period argument *P* is *not* a date value.

```
mysql> SELECT PERIOD_ADD(9801,2);
        -> 199803
```

- PERIOD_DIFF(*P1*,*P2*)

  Returns the number of months between periods *P1* and *P2*. *P1* and *P2* should be in the format YYMM or YYYYMM. Note that the period arguments *P1* and *P2* are *not* date values.

```
mysql> SELECT PERIOD_DIFF(9802,199703);
        -> 11
```

- `QUARTER(date)`

  Returns the quarter of the year for *date*, in the range `1` to `4`.

```
mysql> SELECT QUARTER('98-04-01');
        -> 2
```

- `SECOND(time)`

  Returns the second for *time*, in the range `0` to `59`.

```
mysql> SELECT SECOND('10:05:03');
        -> 3
```

- `SEC_TO_TIME(seconds)`

  Returns the *seconds* argument, converted to hours, minutes, and seconds, as a value in `'HH:MM:SS'` or `HHMMSS` format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT SEC_TO_TIME(2378);
        -> '00:39:38'
mysql> SELECT SEC_TO_TIME(2378) + 0;
        -> 3938
```

- `STR_TO_DATE(str,format)`

  This is the reverse function of the `DATE_FORMAT()` function. It takes a string *str* and a format string *format*. `STR_TO_DATE()` returns a `DATETIME` value if the format string contains both date and time parts, or a `DATE` or `TIME` value if the string contains only date or time parts.

  The date, time, or datetime values contained in *str* should be given in the format indicated by

format. For the specifiers that can be used in *format*, see the table in the DATE_FORMAT() function description. All other characters are just taken verbatim, thus not being interpreted. If *str* contains an illegal date, time, or datetime value, STR_TO_DATE() returns NULL.

```
mysql> SELECT STR_TO_DATE('03.10.2003 09.20',
    ->                    '%d.%m.%Y %H.%i');
        -> '2003-10-03 09:20:00'
mysql> SELECT STR_TO_DATE('10arp', '%carp');
        -> '0000-10-00 00:00:00'
mysql> SELECT STR_TO_DATE('2003-15-10 00:00:00',
    ->                    '%Y-%m-%d %H:%i:%s');
        -> NULL
```

Range checking on the parts of date values is as described in Section 11.3.1, “The DATETIME, DATE, and TIMESTAMP Types”. This means, for example, that a date with a day part larger than the number of days in a month is allowable as long as the day part is in the range from 1 to 31. Also, ``zero'' dates or dates with part values of 0 are allowed.

```
mysql> SELECT STR_TO_DATE('00/00/0000', '%m/%d/%Y');
        -> '0000-00-00'
mysql> SELECT STR_TO_DATE('04/31/2004', '%m/%d/%Y');
        -> '2004-04-31'
```

STR_TO_DATE() is available as of MySQL 4.1.1.

- SUBDATE(*date*,INTERVAL *expr type*),SUBDATE(*expr*,*days*)

  When invoked with the INTERVAL form of the second argument, SUBDATE() is a synonym for DATE_SUB(). For information on the INTERVAL argument, see the discussion for DATE_ADD().

```
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
        -> '1997-12-02'
mysql> SELECT SUBDATE('1998-01-02', INTERVAL 31 DAY);
        -> '1997-12-02'
```

  As of MySQL 4.1.1, the second syntax is allowed, where *expr* is a date or datetime expression and *days* is the number of days to be subtracted from *expr*.

```
mysql> SELECT SUBDATE('1998-01-02 12:00:00', 31);
        -> '1997-12-02 12:00:00'
```

- SUBTIME(*expr*,*expr2*)

  SUBTIME() subtracts *expr2* from *expr* and returns the result. *expr* is a time or datetime expression, and *expr2* is a time expression.

```
mysql> SELECT SUBTIME('1997-12-31 23:59:59.999999',
    ->                '1 1:1:1.000002');
        -> '1997-12-30 22:58:58.999997'
mysql> SELECT SUBTIME('01:00:00.999999', '02:00:00.999998');
        -> '-00:59:59.999999'
```

`SUBTIME()` was added in MySQL 4.1.1.

- `SYSDATE()`

  `SYSDATE()` is a synonym for `NOW()`.

- `TIME(expr)`

  Extracts the time part of the time or datetime expression *`expr`*.

```
mysql> SELECT TIME('2003-12-31 01:02:03');
        -> '01:02:03'
mysql> SELECT TIME('2003-12-31 01:02:03.000123');
        -> '01:02:03.000123'
```

`TIME()` is available as of MySQL 4.1.1.

- `TIMEDIFF(expr,expr2)`

  `TIMEDIFF()` returns the time between the start time *`expr`* and the end time *`expr2`*. *`expr`* and *`expr2`* are time or date-and-time expressions, but both must be of the same type.

```
mysql> SELECT TIMEDIFF('2000:01:01 00:00:00',
    ->                 '2000:01:01 00:00:00.000001');
        -> '-00:00:00.000001'
mysql> SELECT TIMEDIFF('1997-12-31 23:59:59.000001',
    ->                 '1997-12-30 01:01:01.000002');
        -> '46:58:57.999999'
```

`TIMEDIFF()` was added in MySQL 4.1.1.

- `TIMESTAMP(expr)`, `TIMESTAMP(expr,expr2)`

  With one argument, returns the date or datetime expression *`expr`* as a datetime value. With two arguments, adds the time expression *`expr2`* to the date or datetime expression *`expr`* and returns a datetime value.

```
mysql> SELECT TIMESTAMP('2003-12-31');
        -> '2003-12-31 00:00:00'
mysql> SELECT TIMESTAMP('2003-12-31 12:00:00','12:00:00');
        -> '2004-01-01 00:00:00'
```

`TIMESTAMP()` is available as of MySQL 4.1.1.

- `TIMESTAMPADD(interval,int_expr,datetime_expr)`

  Adds the integer expression *`int_expr`* to the date or datetime expression *`datetime_expr`*. The unit for *`int_expr`* is given by the `interval` argument, which should be one of the following values: `FRAC_SECOND`, `SECOND`, `MINUTE`, `HOUR`, `DAY`, `WEEK`, `MONTH`, `QUARTER`, or `YEAR`.

  The *`interval`* value may be specified using one of keywords as shown, or with a prefix of `SQL_TSI_`. For example, `DAY` or `SQL_TSI_DAY` both are legal.

```
mysql> SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');
        -> '2003-01-02 00:01:00'
mysql> SELECT TIMESTAMPADD(WEEK,1,'2003-01-02');
        -> '2003-01-09'
```

  `TIMESTAMPADD()` is available as of MySQL 5.0.0.

- `TIMESTAMPDIFF(interval,datetime_expr1,datetime_expr2)`

  Returns the integer difference between the date or datetime expressions *`datetime_expr1`* and *`datetime_expr2`*. The unit for the result is given by the *`interval`* argument. The legal values for *`interval`* are the same as those listed in the description of the `TIMESTAMPADD()` function.

```
mysql> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
        -> 3
mysql> SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');
        -> -1
```

  `TIMESTAMPDIFF()` is available as of MySQL 5.0.0.

- `TIME_FORMAT(time,format)`

  This is used like the `DATE_FORMAT()` function, but the *`format`* string may contain only those format specifiers that handle hours, minutes, and seconds. Other specifiers produce a `NULL` value or `0`.

  If the *`time`* value contains an hour part that is greater than `23`, the `%H` and `%k` hour format specifiers produce a value larger than the usual range of `0..23`. The other hour format specifiers produce the hour value modulo 12.

```
mysql> SELECT TIME_FORMAT('100:00:00', '%H %k %h %I %l');
        -> '100 100 04 04 4'
```

- `TIME_TO_SEC(time)`

  Returns the *time* argument, converted to seconds.

```
mysql> SELECT TIME_TO_SEC('22:23:00');
        -> 80580
mysql> SELECT TIME_TO_SEC('00:39:38');
        -> 2378
```

- `TO_DAYS(date)`

  Given a date *date*, returns a daynumber (the number of days since year 0).

```
mysql> SELECT TO_DAYS(950501);
        -> 728779
mysql> SELECT TO_DAYS('1997-10-07');
        -> 729669
```

  `TO_DAYS()` is not intended for use with values that precede the advent of the Gregorian calendar (1582), because it does not take into account the days that were lost when the calendar was changed.

  Remember that MySQL converts two-digit year values in dates to four-digit form using the rules in Section 11.3, “Date and Time Types”. For example, `'1997-10-07'` and `'97-10-07'` are seen as identical dates:

```
mysql> SELECT TO_DAYS('1997-10-07'), TO_DAYS('97-10-07');
        -> 729669, 729669
```

  For other dates before 1582, results from this function are undefined.

- `UNIX_TIMESTAMP()`, `UNIX_TIMESTAMP(date)`

  If called with no argument, returns a Unix timestamp (seconds since `'1970-01-01 00:00:00'` GMT) as an unsigned integer. If `UNIX_TIMESTAMP()` is called with a *date* argument, it returns the value of the argument as seconds since `'1970-01-01 00:00:00'` GMT. *date* may be a `DATE` string, a `DATETIME` string, a `TIMESTAMP`, or a number in the format `YYMMDD` or `YYYYMMDD` in local time.

```
mysql> SELECT UNIX_TIMESTAMP();
        -> 882226357
```

```
mysql> SELECT UNIX_TIMESTAMP('1997-10-04 22:23:00');
        -> 875996580
```

When `UNIX_TIMESTAMP` is used on a `TIMESTAMP` column, the function returns the internal timestamp value directly, with no implicit ``string-to-Unix-timestamp'' conversion. If you pass an out-of-range date to `UNIX_TIMESTAMP()`, it returns `0`, but please note that only basic range checking is performed (year from `1970` to `2037`, month from `01` to `12`, day from `01` from `31`).

If you want to subtract `UNIX_TIMESTAMP()` columns, you might want to cast the result to signed integers. See Section 12.7, “Cast Functions and Operators”.

- `UTC_DATE`, `UTC_DATE()`

  Returns the current UTC date as a value in `'YYYY-MM-DD'` or `YYYYMMDD` format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT UTC_DATE(), UTC_DATE() + 0;
        -> '2003-08-14', 20030814
```

  `UTC_DATE()` is available as of MySQL 4.1.1.

- `UTC_TIME`, `UTC_TIME()`

  Returns the current UTC time as a value in `'HH:MM:SS'` or `HHMMSS` format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT UTC_TIME(), UTC_TIME() + 0;
        -> '18:07:53', 180753
```

  `UTC_TIME()` is available as of MySQL 4.1.1.

- `UTC_TIMESTAMP`, `UTC_TIMESTAMP()`

  Returns the current UTC date and time as a value in `'YYYY-MM-DD HH:MM:SS'` or `YYYYMMDDHHMMSS` format, depending on whether the function is used in a string or numeric context.

```
mysql> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;
        -> '2003-08-14 18:08:04', 20030814180804
```

  `UTC_TIMESTAMP()` is available as of MySQL 4.1.1.

- `WEEK(`*`date`*`[,`*`mode`*`])`

The function returns the week number for *date*. The two-argument form of WEEK() allows you to specify whether the week starts on Sunday or Monday and whether the return value should be in the range from 0 to 53 or from 1 to 53. If the *mode* argument is omitted, the value of the default_week_format system variable is used (or 0 before MySQL 4.0.14). See Section 5.2.3, “Server System Variables”.

The following table describes how the *mode* argument works:

	First day		
Mode	**of week**	**Range**	**Week 1 is the first week...**
0	Sunday	0-53	with a Sunday in this year
1	Monday	0-53	with more than 3 days this year
2	Sunday	1-53	with a Sunday in this year
3	Monday	1-53	with more than 3 days this year
4	Sunday	0-53	with more than 3 days this year
5	Monday	0-53	with a Monday in this year
6	Sunday	1-53	with more than 3 days this year
7	Monday	1-53	with a Monday in this year

A *mode* value of 3 can be used as of MySQL 4.0.5. Values of 4 and above can be used as of MySQL 4.0.17.

```
mysql> SELECT WEEK('1998-02-20');
         -> 7
mysql> SELECT WEEK('1998-02-20',0);
         -> 7
mysql> SELECT WEEK('1998-02-20',1);
         -> 8
mysql> SELECT WEEK('1998-12-31',1);
         -> 53
```

Note: In MySQL 4.0, WEEK(*date*,0) was changed to match the calendar in the USA. Before that, WEEK() was calculated incorrectly for dates in the USA. (In effect, WEEK(*date*) and WEEK(*date*,0) were incorrect for all cases.)

Note that if a date falls in the last week of the previous year, MySQL returns 0 if you don't use 2, 3, 6, or 7 as the optional mode argument:

```
mysql> SELECT YEAR('2000-01-01'), WEEK('2000-01-01',0);
         -> 2000, 0
```

One might argue that MySQL should return 52 for the WEEK() function, because the given date actually occurs in the 52nd week of 1999. We decided to return 0 instead because we want the function to return ``the week number in the given year.'' This makes use of the WEEK() function reliable when combined with other functions that extract a date part from a date.

If you would prefer the result to be evaluated with respect to the year that contains the first day of the week for the given date, you should use 0, 2, 5, or 7 as the optional mode argument.

```
mysql> SELECT WEEK('2000-01-01',2);
        -> 52
```

Alternatively, use the `YEARWEEK()` function:

```
mysql> SELECT YEARWEEK('2000-01-01');
        -> 199952
mysql> SELECT MID(YEARWEEK('2000-01-01'),5,2);
        -> '52'
```

- `WEEKDAY(date)`

  Returns the weekday index for *`date`* (`0` = Monday, `1` = Tuesday, ... `6` = Sunday).

```
mysql> SELECT WEEKDAY('1998-02-03 22:23:00');
        -> 1
mysql> SELECT WEEKDAY('1997-11-05');
        -> 2
```

- `WEEKOFYEAR(date)`

  Returns the calendar week of the date as a number in the range from `1` to `53`. It is a compatibility function that is equivalent to `WEEK(date,3)`.

```
mysql> SELECT WEEKOFYEAR('1998-02-20');
        -> 8
```

  `WEEKOFYEAR()` is available as of MySQL 4.1.1.

- `YEAR(date)`

  Returns the year for *`date`*, in the range `1000` to `9999`.

```
mysql> SELECT YEAR('98-02-03');
        -> 1998
```

- `YEARWEEK(date)`, `YEARWEEK(date,start)`

  Returns year and week for a date. The *`start`* argument works exactly like the *`start`* argument to `WEEK()`. The year in the result may be different from the year in the date argument for the first and

the last week of the year.

```
mysql> SELECT YEARWEEK('1987-01-01');
        -> 198653
```

Note that the week number is different from what the `WEEK()` function would return (`0`) for optional arguments `0` or `1`, as `WEEK()` then returns the week in the context of the given year.

`YEARWEEK()` was added in MySQL 3.23.8.

12.6. Full-Text Search Functions

- `MATCH (col1,col2,...) AGAINST (expr [IN BOOLEAN MODE | WITH QUERY EXPANSION])`

  As of MySQL 3.23.23, MySQL has support for full-text indexing and searching. A full-text index in MySQL is an index of type `FULLTEXT`. `FULLTEXT` indexes are used with `MyISAM` tables only and can be created from `CHAR`, `VARCHAR`, or `TEXT` columns at `CREATE TABLE` time or added later with `ALTER TABLE` or `CREATE INDEX`. For large datasets, it is much faster to load your data into a table that has no `FULLTEXT` index, then create the index with `ALTER TABLE` (or `CREATE INDEX`). Loading data into a table that has an existing `FULLTEXT` index could be significantly slower.

  Constraints on full-text searching are listed in Section 12.6.3, “Full-Text Restrictions”.

Full-text searching is performed with the `MATCH()` function.

```
mysql> CREATE TABLE articles (
    ->   id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
    ->   title VARCHAR(200),
    ->   body TEXT,
    ->   FULLTEXT (title,body)
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO articles (title,body) VALUES
    -> ('MySQL Tutorial','DBMS stands for DataBase ...'),
    -> ('How To Use MySQL Well','After you went through a ...'),
    -> ('Optimizing MySQL','In this tutorial we will show ...'),
    -> ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
    -> ('MySQL vs. YourSQL','In the following database comparison ...'),
    -> ('MySQL Security','When configured properly, MySQL ...');
Query OK, 6 rows affected (0.00 sec)
Records: 6  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM articles
    -> WHERE MATCH (title,body) AGAINST ('database');
+----+-------------------+-----------------------------------------+
| id | title             | body                                    |
+----+-------------------+-----------------------------------------+
|  5 | MySQL vs. YourSQL | In the following database comparison ... |
|  1 | MySQL Tutorial    | DBMS stands for DataBase ...            |
+----+-------------------+-----------------------------------------+
2 rows in set (0.00 sec)
```

The MATCH() function performs a natural language search for a string against a text collection. A collection is a set of one or more columns included in a FULLTEXT index. The search string is given as the argument to AGAINST(). For every row in the table, MATCH() returns a relevance value, that is, a similarity measure between the search string and the text in that row in the columns named in the MATCH() list.

By default, the search is performed in case-insensitive fashion. In MySQL 4.1 and up, you can make a full-text search by using a binary collation for the indexed columns. For example, a column that has a character set of latin1 can be assigned a collation of latin1_bin to make it case sensitive for full-text searches.

When MATCH() is used in a WHERE clause, as in the preceding example, the rows returned are automatically sorted with the highest relevance first. Relevance values are non-negative floating-point numbers. Zero relevance means no similarity. Relevance is computed based on the number of words in the row, the number of unique words in that row, the total number of words in the collection, and the number of documents (rows) that contain a particular word.

For natural-language full-text searches, it is a requirement that the columns named in the MATCH() function be the same columns included in some FULLTEXT index in your table. For the preceding query, note that the columns named in the MATCH() function (title and body) are the same as those named in the definition of the article table's FULLTEXT index. If you wanted to search the title or body separately, you would need to create FULLTEXT indexes for each column.

It is also possible to perform a boolean search or a search with query expansion. These search types are described in Section 12.6.1, “Boolean Full-Text Searches” and Section 12.6.2, “Full-Text Searches with Query Expansion”.

The preceding example is a basic illustration showing how to use the MATCH() function where rows are returned in order of decreasing relevance. The next example shows how to retrieve the relevance values explicitly. Returned rows are not ordered because the SELECT statement includes neither WHERE nor ORDER BY clauses:

```
mysql> SELECT id, MATCH (title,body) AGAINST ('Tutorial')
    -> FROM articles;
+----+----------------------------------------+
| id | MATCH (title,body) AGAINST ('Tutorial') |
+----+----------------------------------------+
|  1 |                       0.65545833110809 |
|  2 |                                      0 |
|  3 |                       0.66266459226608 |
|  4 |                                      0 |
|  5 |                                      0 |
|  6 |                                      0 |
+----+----------------------------------------+
6 rows in set (0.00 sec)
```

The following example is more complex. The query returns the relevance values and it also sorts the rows in order of decreasing relevance. To achieve this result, you should specify MATCH() twice: once in the SELECT list and once in the WHERE clause. This causes no additional overhead, because the MySQL optimizer notices that the two MATCH() calls are identical and invokes the full-text search code only once.

```
mysql> SELECT id, body, MATCH (title,body) AGAINST
    -> ('Security implications of running MySQL as root') AS score
    -> FROM articles WHERE MATCH (title,body) AGAINST
    -> ('Security implications of running MySQL as root');
+----+-------------------------------------+-----------------+
| id | body                                | score           |
+----+-------------------------------------+-----------------+
```

```
|  4 | 1. Never run mysqld as root. 2. ... | 1.5219271183014 |
|  6 | When configured properly, MySQL ... | 1.3114095926285 |
+----+-------------------------------------+-----------------+
2 rows in set (0.00 sec)
```

MySQL uses a very simple parser to split text into words. A ``word'' is any sequence of true word characters (letters, digits, and underscores), optionally separated by no more than one sequential `''` character. For example, `wasn't` is parsed as a single word, but `wasn''t` is parsed as two words `wasn` and `t`. (And then `t` would be ignored as too short according to the rules following.) Also, single quotes at the ends of words are stripped; only embedded single quotes are retained.

Some words are ignored in full-text searches:

- Any word that is too short is ignored. The default minimum length of words that are found by full-text searches is four characters.
- Words in the stopword list are ignored. A stopword is a word such as ``the'' or ``some'' that is so common that it is considered to have zero semantic value. There is a built-in stopword list.

The default minimum word length and stopword list can be changed as described in Section 12.6.4, “Fine-Tuning MySQL Full-Text Search”.

Every correct word in the collection and in the query is weighted according to its significance in the collection or query. This way, a word that is present in many documents has a lower weight (and may even have a zero weight), because it has lower semantic value in this particular collection. Conversely, if the word is rare, it receives a higher weight. The weights of the words are then combined to compute the relevance of the row.

Such a technique works best with large collections (in fact, it was carefully tuned this way). For very small tables, word distribution does not adequately reflect their semantic value, and this model may sometimes produce bizarre results. For example, although the word ``MySQL'' is present in every row of the `articles` table, a search for the word produces no results:

```
mysql> SELECT * FROM articles
    -> WHERE MATCH (title,body) AGAINST ('MySQL');
Empty set (0.00 sec)
```

The search result is empty because the word ``MySQL'' is present in at least 50% of the rows. As such, it is effectively treated as a stopword. For large datasets, this is the most desirable behavior---a natural language query should not return every second row from a 1GB table. For small datasets, it may be less desirable.

A word that matches half of rows in a table is less likely to locate relevant documents. In fact, it most likely finds plenty of irrelevant documents. We all know this happens far too often when we are trying to find something on the Internet with a search engine. It is with this reasoning that rows containing the word are assigned a low semantic value for *the particular dataset in which they occur*. A given word may exceed the 50% threshold in one dataset but not another.

The 50% threshold has a significant implication when you first try full-text searching to see how it works: If you create a table and insert only one or two rows of text into it, every word in the text occurs in at least 50% of the rows. As a result, no search returns any results. Be sure to insert at least three rows, and preferably many more.

12.6.1. Boolean Full-Text Searches

As of Version 4.0.1, MySQL can also perform boolean full-text searches using the `IN BOOLEAN MODE` modifier.

```
mysql> SELECT * FROM articles WHERE MATCH (title,body)
    -> AGAINST ('+MySQL -YourSQL' IN BOOLEAN MODE);
+----+-----------------------+-------------------------------------+
| id | title                 | body                                |
+----+-----------------------+-------------------------------------+
|  1 | MySQL Tutorial        | DBMS stands for DataBase ...        |
|  2 | How To Use MySQL Well | After you went through a ...        |
|  3 | Optimizing MySQL      | In this tutorial we will show ...   |
|  4 | 1001 MySQL Tricks     | 1. Never run mysqld as root. 2. ... |
|  6 | MySQL Security        | When configured properly, MySQL ... |
+----+-----------------------+-------------------------------------+
```

This query retrieves all the rows that contain the word ``MySQL'' but that do *not* contain the word ``YourSQL''.

Boolean full-text searches have these characteristics:

- They do not use the 50% threshold.
- They do not automatically sort rows in order of decreasing relevance. You can see this from the preceding query result: The row with the highest relevance is the one that contains ``MySQL'' twice, but it is listed last, not first.
- They can work even without a `FULLTEXT` index, although this would be *slow*.
- The minimum and maximum word length full-text parameters apply.
- The stopword list applies.

The boolean full-text search capability supports the following operators:

- `+`

  A leading plus sign indicates that this word *must be* present in every row returned.

- `-`

  A leading minus sign indicates that this word *must not be* present in any row returned.

- `(no operator)`

  By default (when neither `+` nor `-` is specified) the word is optional, but the rows that contain it are rated higher. This mimics the behavior of `MATCH() ... AGAINST()` without the `IN BOOLEAN MODE` modifier.

- `> <`

  These two operators are used to change a word's contribution to the relevance value that is assigned to a row. The `>` operator increases the contribution and the `<` operator decreases it. See the example below.

- `( )`

  Parentheses are used to group words into subexpressions. Parenthesized groups can be nested.

- `~`

  A leading tilde acts as a negation operator, causing the word's contribution to the row relevance to be negative. It's useful for marking noise words. A row that contains such a word is rated lower than others, but is not excluded altogether, as it would be with the `-` operator.

- `*`

  An asterisk is the truncation operator. Unlike the other operators, it should be *appended* to the word.

- `"`

  A phrase that is enclosed within double quote (`'"'`) characters matches only rows that contain the phrase *literally, as it was typed*. The full-text engine splits the phrase into words, performs a search in the `FULLTEXT` index for the words. Before MySQL 5.0.3, the engine then performs a substring search for the phrase in the records that are found, so the match must include non-word characters in the phrase. As of MySQL 5.0.3, non-word characters need not be matched exactly: Phrase searching requires only that matches contain exactly the same words as the phrase and in the same order. For example, `"test phrase"` matches `"test, phrase"` as of MySQL 5.0.3, but not before.

  If the phrase contains no words that are in the index, the result is empty. For example, if all words are either stopwords or shorter than the minimum length of indexed words, the result is empty.

The following examples demonstrate some search strings that use boolean full-text operators:

- `'apple banana'`

  Find rows that contain at least one of the two words.

- `'+apple +juice'`

  Find rows that contain both words.

- `'+apple macintosh'`

  Find rows that contain the word ``apple'', but rank rows higher if they also contain ``macintosh''.

- `'+apple -macintosh'`

  Find rows that contain the word ``apple'' but not ``macintosh''.

- `'+apple +(>turnover <strudel)'`

  Find rows that contain the words ``apple'' and ``turnover'', or ``apple'' and ``strudel'' (in any order), but rank ``apple turnover'' higher than ``apple strudel''.

- `'apple*'`

  Find rows that contain words such as ``apple'', ``apples'', ``applesauce'', or ``applet''.

- `'"some words"'`

  Find rows that contain the exact phrase ``some words'' (for example, rows that contain ``some words of wisdom'' but not ``some noise words''). Note that the `'"'` characters that surround the phrase are operator characters that delimit the phrase. They are not the quotes that surround the search string itself.

12.6.2. Full-Text Searches with Query Expansion

As of MySQL 4.1.1, full-text search supports query expansion (in particular, its variant ``blind query expansion''). This is generally useful when a search phrase is too short, which often means that the user is relying on implied knowledge that the full-text search engine usually lacks. For example, a user searching for ``database'' may really mean that ``MySQL'', ``Oracle'', ``DB2'', and ``RDBMS'' all are phrases that should match ``databases'' and should be returned, too. This is implied knowledge.

Blind query expansion (also known as automatic relevance feedback) is enabled by adding `WITH QUERY EXPANSION` following the search phrase. It works by performing the search twice, where the search phrase for the second search is the original search phrase concatenated with the few top found documents from the first search. Thus, if one of these documents contains the word ``databases'' and the word ``MySQL'', the second search finds the documents that contain the word ``MySQL'' even if they do not contain the word ``database''. The following example shows this difference:

```
mysql> SELECT * FROM articles
    -> WHERE MATCH (title,body) AGAINST ('database');
+----+-------------------+----------------------------------------+
| id | title             | body                                   |
+----+-------------------+----------------------------------------+
|  5 | MySQL vs. YourSQL | In the following database comparison ... |
|  1 | MySQL Tutorial    | DBMS stands for DataBase ...           |
+----+-------------------+----------------------------------------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM articles
    -> WHERE MATCH (title,body)
    -> AGAINST ('database' WITH QUERY EXPANSION);
+----+-------------------+----------------------------------------+
| id | title             | body                                   |
+----+-------------------+----------------------------------------+
|  1 | MySQL Tutorial    | DBMS stands for DataBase ...           |
|  5 | MySQL vs. YourSQL | In the following database comparison ... |
|  3 | Optimizing MySQL  | In this tutorial we will show ...      |
+----+-------------------+----------------------------------------+
3 rows in set (0.00 sec)
```

Another example could be searching for books by Georges Simenon about Maigret, when a user is not sure how to spell ``Maigret''. A search for ``Megre and the reluctant witnesses'' finds only ``Maigret and the Reluctant Witnesses'' without query expansion. A search with query expansion finds all books with the word ``Maigret'' on the second pass.

Note: Because blind query expansion tends to increase noise significantly by returning non-relevant documents, it's only meaningful to use when a search phrase is rather short.

12.6.3. Full-Text Restrictions

- Full-text searches are supported for `MyISAM` tables only.
- As of MySQL 4.1.1, full-text searches can be used with most multi-byte character sets. The exception is that for Unicode, the `utf8` character set can be used, but not the `ucs2` character set.
- As of MySQL 4.1, the use of multiple character sets within a single table is supported. However, all columns in a `FULLTEXT` index must have the same character set and collation.
- The `MATCH()` column list must exactly match the column list in some `FULLTEXT` index definition for the table, unless this `MATCH()` is `IN BOOLEAN MODE`.

- The argument to `AGAINST()` must be a constant string.

12.6.4. Fine-Tuning MySQL Full-Text Search

The MySQL full-text search capability has few user-tunable parameters yet, although adding more is very high on the TODO. You can exert more control over full-text searching behavior if you have a MySQL source distribution because some changes require source code modifications. See Section 2.8, “MySQL Installation Using a Source Distribution”.

Note that full-text search was carefully tuned for the best searching effectiveness. Modifying the default behavior, in most cases, makes the search results worse. Do not alter the MySQL sources unless you know what you are doing!

Most full-text variables described in the following items must be set at server startup time. For these variables, a server restart is required to change them and you cannot modify them dynamically while the server is running.

Some variable changes require that you rebuild the `FULLTEXT` indexes in your tables. Instructions for doing this are given at the end of this section.

- The minimum and maximum length of words to be indexed is defined by the `ft_min_word_len` and `ft_max_word_len` system variables (available as of MySQL 4.0.0). See Section 5.2.3, “Server System Variables”. The default minimum value is four characters. The default maximum depends on your version of MySQL. If you change either value, you must rebuild your `FULLTEXT` indexes. For example, if you want three-character words to be searchable, you can set the `ft_min_word_len` variable by putting the following lines in an option file:

  ```
  [mysqld]
  ft_min_word_len=3
  ```

  Then restart the server and rebuild your `FULLTEXT` indexes. Also note particularly the remarks regarding **myisamchk** in the instructions following this list.

- To override the default stopword list, set the `ft_stopword_file` system variable (available as of MySQL 4.0.10). See Section 5.2.3, “Server System Variables”. The variable value should be the pathname of the file containing the stopword list, or the empty string to disable stopword filtering. After changing the value, rebuild your `FULLTEXT` indexes.

- The 50% threshold for natural language searches is determined by the particular weighting scheme chosen. To disable it, look for the following line in `myisam/ftdefs.h`:

  ```
  #define GWS_IN_USE GWS_PROB
  ```

  Change the line to this:

  ```
  #define GWS_IN_USE GWS_FREQ
  ```

  Then recompile MySQL. There is no need to rebuild the indexes in this case. **Note**: By doing this you *severely* decrease MySQL's ability to provide adequate relevance values for the `MATCH()` function. If you really need to search for such common words, it would be better to search using `IN BOOLEAN MODE` instead, which does not observe the 50% threshold.

- To change the operators used for boolean full-text searches, set the `ft_boolean_syntax` system variable (available as of MySQL 4.0.1). The variable also can be changed while the server is run-

ning, but you must have the SUPER privilege to do so. No index rebuilding is necessary. Section 5.2.3, “Server System Variables” describes the rules that define how to set this variable.

If you modify full-text variables that affect indexing (ft_min_word_len, ft_max_word_len, or ft_stopword_file), you must rebuild your FULLTEXT indexes after making the changes and restarting the server. To rebuild the indexes in this case, it's sufficient to do a QUICK repair operation:

```
mysql> REPAIR TABLE tbl_name QUICK;
```

With regard specifically to using the IN BOOLEAN MODE capability, if you upgrade from MySQL 3.23 to 4.0 or later, it's necessary to replace the index header as well. To do this, do a USE_FRM repair operation:

```
mysql> REPAIR TABLE tbl_name USE_FRM;
```

This is necessary because boolean full-text searches require a flag in the index header that was not present in MySQL 3.23, and that is not added if you do only a QUICK repair. If you attempt a boolean full-text search without rebuilding the indexes this way, the search returns incorrect results.

Note that if you use **myisamchk** to perform an operation that modifies table indexes (such as repair or analyze), the FULLTEXT indexes are rebuilt using the default full-text parameter values for minimum and maximum word length and the stopword file unless you specify otherwise. This can result in queries failing.

The problem occurs because these parameters are known only by the server. They are not stored in MyISAM index files. To avoid the problem if you have modified the minimum or maximum word length or the stopword file in the server, specify the same ft_min_word_len, ft_max_word_len, and ft_stopword_file values to **myisamchk** that you use for **mysqld**. For example, if you have set the minimum word length to 3, you can repair a table with **myisamchk** like this:

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

To ensure that **myisamchk** and the server use the same values for full-text parameters, you can place each one in both the [mysqld] and [myisamchk] sections of an option file:

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

An alternative to using **myisamchk** is to use the REPAIR TABLE, ANALYZE TABLE, OPTIMIZE TABLE, or ALTER TABLE. These statements are performed by the server, which knows the proper full-text parameter values to use.

12.6.5. Full-Text Search TODO

- Improved performance for all FULLTEXT operations.
- Proximity operators.
- Support for ``always-index words." These could be any strings the user wants to treat as words, such as ``C++", ``AS/400", or ``TCP/IP".

- Support for full-text search in MERGE tables.
- Support for the ucs2 character set.
- Make the stopword list dependent on the language of the dataset.
- Stemming
- Generic user-suppliable UDF preparser.
- Make the model more flexible (by adding some adjustable parameters to FULLTEXT in CREATE TABLE and ALTER TABLE statements).

12.7. Cast Functions and Operators

- BINARY

  The BINARY operator casts the string following it to a binary string. This is an easy way to force a column comparison to be done byte by byte rather than character by character. This causes the comparison to be case sensitive even if the column isn't defined as BINARY or BLOB. BINARY also causes trailing spaces to be significant.

```
mysql> SELECT 'a' = 'A';
        -> 1
mysql> SELECT BINARY 'a' = 'A';
        -> 0
mysql> SELECT 'a' = 'a ';
        -> 1
mysql> SELECT BINARY 'a' = 'a ';
        -> 0
```

  BINARY affects the entire comparison; it can be given before either operand with the same result.

  BINARY was added in MySQL 3.23.0. As of MySQL 4.0.2, BINARY str is a shorthand for CAST(str AS BINARY).

  Note that in some contexts, if you cast an indexed column to BINARY, MySQL is not able to use the index efficiently.

  If you want to compare a BLOB value or other binary string in case-insensitive fashion, you can do so as follows:

  - Before MySQL 4.1.1, use the UPPER() function to convert the binary string to uppercase before performing the comparison:

    SELECT 'A' LIKE UPPER(*blob_col*) FROM *tbl_name*;

    If the comparison value is lowercase, convert the string value using LOWER() instead.

  - For MySQL 4.1.1 and up, binary strings have no character set, and thus no concept of lettercase. To perform a case-insensitive comparison, use the CONVERT() function to convert the string value to a character set that is not case sensitive. The result is a non-binary string, so the LIKE

operation is not case sensitive:

```
SELECT 'A' LIKE CONVERT(blob_col USING latin1) FROM tbl_name;
```

To use a different character set, substitute its name for `latin1` in the preceding statement.

`CONVERT()` can be used more generally for comparing strings that are represented in different character sets.

- `CAST(expr AS type)`, `CONVERT(expr,type)`, `CONVERT(expr USING transcoding_name)`

  The `CAST()` and `CONVERT()` functions can be used to take a value of one type and produce a value of another type.

  The *`type`* can be one of the following values:

  - `BINARY`
  - `CHAR`
  - `DATE`
  - `DATETIME`
  - `SIGNED [INTEGER]`
  - `TIME`
  - `UNSIGNED [INTEGER]`

  `BINARY` produces a binary string. See the entry for the `BINARY` operator in this section for a description of how this affects comparisons.

  `CAST()` and `CONVERT()` are available as of MySQL 4.0.2. The `CHAR` conversion type is available as of 4.0.6. The `USING` form of `CONVERT()` is available as of 4.1.0.

  `CAST()` and `CONVERT(... USING ...)` are standard SQL syntax. The non-`USING` form of `CONVERT()` is ODBC syntax.

  `CONVERT()` with `USING` is used to convert data between different character sets. In MySQL, transcoding names are the same as the corresponding character set names. For example, this statement converts the string `'abc'` in the server's default character set to the corresponding string in the `utf8` character set:

  ```
  SELECT CONVERT('abc' USING utf8);
  ```

The cast functions are useful when you want to create a column with a specific type in a `CREATE ... SELECT` statement:

```
CREATE TABLE new_table SELECT CAST('2000-01-01' AS DATE);
```

The functions also can be useful for sorting ENUM columns in lexical order. Normally sorting of ENUM columns occurs using the internal numeric values. Casting the values to CHAR results in a lexical sort:

```
SELECT enum_col FROM tbl_name ORDER BY CAST(enum_col AS CHAR);
```

CAST(*str* AS BINARY) is the same thing as BINARY *str*. CAST(*expr* AS CHAR) treats the expression as a string with the default character set.

Note: In MySQL 4.0, a CAST() to DATE, DATETIME, or TIME only marks the column to be a specific type but doesn't change the value of the column.

As of MySQL 4.1.0, the value is converted to the correct column type when it's sent to the user (this is a feature of how the new protocol in 4.1 sends date information to the client):

```
mysql> SELECT CAST(NOW() AS DATE);
       -> 2003-05-26
```

As of MySQL 4.1.1, CAST() also changes the result if you use it as part of a more complex expression such as CONCAT('Date: ',CAST(NOW() AS DATE)).

You should not use CAST() to extract data in different formats but instead use string functions like LEFT() or EXTRACT(). See Section 12.5, “Date and Time Functions”.

To cast a string to a numeric value in numeric context, you don't normally have to do anything. Just use the string value as though it were a number:

```
mysql> SELECT 1+'1';
       -> 2
```

If you use a number in string context, the number automatically is converted to a BINARY string.

```
mysql> SELECT CONCAT('hello you ',2);
        -> 'hello you 2'
```

MySQL supports arithmetic with both signed and unsigned 64-bit values. If you are using numeric operators (such as +) and one of the operands is an unsigned integer, the result is unsigned. You can override this by using the SIGNED and UNSIGNED cast operators to cast the operation to a signed or unsigned 64-bit integer, respectively.

```
mysql> SELECT CAST(1-2 AS UNSIGNED)
        -> 18446744073709551615
mysql> SELECT CAST(CAST(1-2 AS UNSIGNED) AS SIGNED);
        -> -1
```

Note that if either operand is a floating-point value, the result is a floating-point value and is not affected by the preceding rule. (In this context, DECIMAL column values are regarded as floating-point values.)

```
mysql> SELECT CAST(1 AS UNSIGNED) - 2.0;
        -> -1.0
```

If you are using a string in an arithmetic operation, this is converted to a floating-point number.

The handing of unsigned values was changed in MySQL 4.0 to be able to support BIGINT values properly. If you have some code that you want to run in both MySQL 4.0 and 3.23, you probably can't use the CAST() function. You can use the following technique to get a signed result when subtracting two

unsigned integer columns `ucol1` and `ucol2`:

```
mysql> SELECT (ucol1+0.0)-(ucol2+0.0) FROM ...;
```

The idea is that the columns are converted to floating-point values before the subtraction occurs.

If you have a problem with `UNSIGNED` columns in old MySQL applications when porting them to MySQL 4.0, you can use the `--sql-mode=NO_UNSIGNED_SUBTRACTION` option when starting **mysqld**. However, as long as you use this option, you are not able to make efficient use of the `BIGINT UNSIGNED` column type.

12.8. Other Functions

12.8.1. Bit Functions

MySQL uses `BIGINT` (64-bit) arithmetic for bit operations, so these operators have a maximum range of 64 bits.

- `|`

  Bitwise OR:

  ```
  mysql> SELECT 29 | 15;
          -> 31
  ```

  The result is an unsigned 64-bit integer.

- `&`

  Bitwise AND:

  ```
  mysql> SELECT 29 & 15;
          -> 13
  ```

  The result is an unsigned 64-bit integer.

- `^`

  Bitwise XOR:

  ```
  mysql> SELECT 1 ^ 1;
          -> 0
  mysql> SELECT 1 ^ 0;
          -> 1
  mysql> SELECT 11 ^ 3;
          -> 8
  ```

The result is an unsigned 64-bit integer.

Bitwise XOR was added in MySQL 4.0.2.

Shifts a longlong (`BIGINT`) number to the left.

- `<<`

```
mysql> SELECT 1 << 2;
        -> 4
```

The result is an unsigned 64-bit integer.

- `>>`

Shifts a longlong (`BIGINT`) number to the right.

```
mysql> SELECT 4 >> 2;
        -> 1
```

The result is an unsigned 64-bit integer.

- `~`

Invert all bits.

```
mysql> SELECT 5 & ~1;
        -> 4
```

The result is an unsigned 64-bit integer.

- `BIT_COUNT(N)`

Returns the number of bits that are set in the argument *N*.

```
mysql> SELECT BIT_COUNT(29);
        -> 4
```

12.8.2. Encryption Functions

The functions in this section encrypt and decrypt data values. If you want to store results from an encryption function that might contain arbitrary byte values, use a `BLOB` column rather than a `CHAR` or `VARCHAR` column to avoid potential problems with trailing space removal that would change data values.

- `AES_ENCRYPT(str,key_str)`, `AES_DECRYPT(crypt_str,key_str)`

  These functions allow encryption and decryption of data using the official AES (Advanced Encryption Standard) algorithm, previously known as "Rijndael." Encoding with a 128-bit key length is used, but you can extend it up to 256 bits by modifying the source. We chose 128 bits because it is much faster and it is usually secure enough.

  The input arguments may be any length. If either argument is `NULL`, the result of this function is also `NULL`.

  Because AES is a block-level algorithm, padding is used to encode uneven length strings and so the result string length may be calculated as `16*(trunc(string_length/16)+1)`.

  If `AES_DECRYPT()` detects invalid data or incorrect padding, it returns `NULL`. However, it is possible for `AES_DECRYPT()` to return a non-`NULL` value (possibly garbage) if the input data or the key is invalid.

  You can use the AES functions to store data in an encrypted form by modifying your queries:

  ```
  INSERT INTO t VALUES (1,AES_ENCRYPT('text','password'));
  ```

  You can get even more security by not transferring the key over the connection for each query, which can be accomplished by storing it in a server-side variable at connection time. For example:

  ```
  SELECT @password:='my password';
  INSERT INTO t VALUES (1,AES_ENCRYPT('text',@password));
  ```

  `AES_ENCRYPT()` and `AES_DECRYPT()` were added in MySQL 4.0.2, and can be considered the most cryptographically secure encryption functions currently available in MySQL.

- `DECODE(crypt_str,pass_str)`

  Decrypts the encrypted string *`crypt_str`* using *`pass_str`* as the password. *`crypt_str`* should be a string returned from `ENCODE()`.

- `ENCODE(str,pass_str)`

  Encrypt *`str`* using *`pass_str`* as the password. To decrypt the result, use `DECODE()`.

  The result is a binary string of the same length as *`str`*. If you want to save it in a column, use a `BLOB` column type.

- `DES_DECRYPT(crypt_str[,key_str])`

  Decrypts a string encrypted with `DES_ENCRYPT()`. On error, this function returns `NULL`.

  Note that this function works only if MySQL has been configured with SSL support. See Section 5.6.7, “Using Secure Connections”.

  If no *`key_str`* argument is given, `DES_DECRYPT()` examines the first byte of the encrypted string to determine the DES key number that was used to encrypt the original string, and then reads the key from the DES key file to decrypt the message. For this to work, the user must have the `SUPER` privilege. The key file can be specified with the `--des-key-file` server option.

  If you pass this function a *`key_str`* argument, that string is used as the key for decrypting the message.

  If the *`crypt_str`* argument doesn't look like an encrypted string, MySQL returns the given *`crypt_str`*.

  `DES_DECRYPT()` was added in MySQL 4.0.1.

- `DES_ENCRYPT(str[,(key_num|key_str)])`

  Encrypts the string with the given key using the Triple-DES algorithm. On error, this function returns `NULL`.

  Note that this function works only if MySQL has been configured with SSL support. See Section 5.6.7, “Using Secure Connections”.

  The encryption key to use is chosen based on the second argument to `DES_ENCRYPT()`, if one was given:

Argument	Description
No argument	The first key from the DES key file is used.
`key_num`	The given key number (0-9) from the DES key file is used.
`key_str`	The given key string is used to encrypt *`str`*.

  The key file can be specified with the `--des-key-file` server option.

  The return string is a binary string where the first character is `CHAR(128 | key_num)`.

  The 128 is added to make it easier to recognize an encrypted key. If you use a string key, *`key_num`* is 127.

  The string length for the result is *`new_len = orig_len + (8-(orig_len % 8))+1`*.

  Each line in the DES key file has the following format:

  `key_num des_key_str`

  Each *`key_num`* must be a number in the range from `0` to `9`. Lines in the file may be in any order. *`des_key_str`* is the string that is used to encrypt the message. Between the number and the key there should be at least one space. The first key is the default key that is used if you don't specify any

key argument to DES_ENCRYPT()

You can tell MySQL to read new key values from the key file with the FLUSH DES_KEY_FILE statement. This requires the RELOAD privilege.

One benefit of having a set of default keys is that it gives applications a way to check for the existence of encrypted column values, without giving the end user the right to decrypt those values.

```
mysql> SELECT customer_address FROM customer_table WHERE
       crypted_credit_card = DES_ENCRYPT('credit_card_number');
```

DES_ENCRYPT() was added in MySQL 4.0.1.

- ENCRYPT(*str*[,*salt*])

  Encrypt *str* using the Unix crypt() system call. The *salt* argument should be a string with two characters. (As of MySQL 3.22.16, *salt* may be longer than two characters.) If no *salt* argument is given, a random value is used.

```
mysql> SELECT ENCRYPT('hello');
        -> 'VxuFAJXVARROc'
```

  ENCRYPT() ignores all but the first eight characters of *str*, at least on some systems. This behavior is determined by the implementation of the underlying crypt() system call.

  If crypt() is not available on your system, ENCRYPT() always returns NULL. Because of this, we recommend that you use MD5() or SHA1() instead, because those two functions exist on all platforms.

- MD5(*str*)

  Calculates an MD5 128-bit checksum for the string. The value is returned as a binary string of 32 hex digits, or NULL if the argument was NULL. The return value can, for example, be used as a hash key.

```
mysql> SELECT MD5('testing');
        -> 'ae2b1fca515949e5d54fb22b8ed95575'
```

  This is the "RSA Data Security, Inc. MD5 Message-Digest Algorithm."

  If you want to convert the value to uppercase, see the description of binary string conversion given in the entry for the BINARY operator in Section 12.7, “Cast Functions and Operators”.

  MD5() was added in MySQL 3.23.2.

- OLD_PASSWORD(*str*)

OLD_PASSWORD() is available as of MySQL 4.1, when the implementation of PASSWORD() was changed to improve security. OLD_PASSWORD() returns the value of the pre-4.1 implementation of PASSWORD(). Section 5.5.9, “Password Hashing in MySQL 4.1”.

- PASSWORD(*str*)

  Calculates and returns a password string from the plaintext password *str*, or NULL if the argument was NULL. This is the function that is used for encrypting MySQL passwords for storage in the Password column of the user grant table.

```
mysql> SELECT PASSWORD('badpwd');
        -> '7f84554057dd964b'
```

  PASSWORD() encryption is one-way (not reversible).

  PASSWORD() does not perform password encryption in the same way that Unix passwords are encrypted. See ENCRYPT().

  Note: The PASSWORD() function is used by the authentication system in MySQL Server, you should *not* use it in your own applications. For that purpose, use MD5() or SHA1() instead. Also see RFC 2195 for more information about handling passwords and authentication securely in your application.

- SHA1(*str*), SHA(*str*)

  Calculates an SHA1 160-bit checksum for the string, as described in RFC 3174 (Secure Hash Algorithm). The value is returned as a string of 40 hex digits, or NULL if the argument was NULL. One of the possible uses for this function is as a hash key. You can also use it as a cryptographically safe function for storing passwords.

```
mysql> SELECT SHA1('abc');
        -> 'a9993e364706816aba3e25717850c26c9cd0d89d'
```

  SHA1() was added in MySQL 4.0.2, and can be considered a cryptographically more secure equivalent of MD5(). SHA() is synonym for SHA1().

12.8.3. Information Functions

- BENCHMARK(*count*,*expr*)

  The BENCHMARK() function executes the expression *expr* repeatedly *count* times. It may be used to time how fast MySQL processes the expression. The result value is always 0. The intended use is from within the **mysql** client, which reports query execution times:

```
mysql> SELECT BENCHMARK(1000000,ENCODE('hello','goodbye'));
+----------------------------------------------+
| BENCHMARK(1000000,ENCODE('hello','goodbye')) |
+----------------------------------------------+
|                                            0 |
+----------------------------------------------+
1 row in set (4.74 sec)
```

The time reported is elapsed time on the client end, not CPU time on the server end. It is advisable to execute `BENCHMARK()` several times, and to interpret the result with regard to how heavily loaded the server machine is.

- `CHARSET(str)`

  Returns the character set of the string argument.

```
mysql> SELECT CHARSET('abc');
        -> 'latin1'
mysql> SELECT CHARSET(CONVERT('abc' USING utf8));
        -> 'utf8'
mysql> SELECT CHARSET(USER());
        -> 'utf8'
```

  `CHARSET()` was added in MySQL 4.1.0.

- `COERCIBILITY(str)`

  Returns the collation coercibility value of the string argument.

```
mysql> SELECT COERCIBILITY('abc' COLLATE latin1_swedish_ci);
        -> 0
mysql> SELECT COERCIBILITY(USER());
        -> 3
mysql> SELECT COERCIBILITY('abc');
        -> 4
```

  The return values have the following meanings:

Coercibility	Meaning	Example
0	Explicit collation	Value with `COLLATE` clause
1	No collation	Concatenation of strings with different collations
2	Implicit collation	Column value
3	System constant	`USER()` return value

4	Coercible	Literal string
5	Ignorable	NULL or an expression derived from NULL

Before MySQL 4.1.11 and 5.0.3, the return values are shown as follows, and functions such as USER() have a coercibility of 2:

Coercibility	Meaning	Example
0	Explicit collation	Value with COLLATE clause
1	No collation	Concatenation of strings with different collations
2	Implicit collation	Column value
3	Coercible	Literal string

Lower values have higher precedence.

COERCIBILITY() was added in MySQL 4.1.1.

- COLLATION(*str*)

  Returns the collation for the character set of the string argument.

```
mysql> SELECT COLLATION('abc');
        -> 'latin1_swedish_ci'
mysql> SELECT COLLATION(_utf8'abc');
        -> 'utf8_general_ci'
```

  COLLATION() was added in MySQL 4.1.0.

- CONNECTION_ID()

  Returns the connection ID (thread ID) for the connection. Every connection has its own unique ID.

```
mysql> SELECT CONNECTION_ID();
        -> 23786
```

  CONNECTION_ID() was added in MySQL 3.23.14.

- CURRENT_USER()

  Returns the username and hostname combination that the current session was authenticated as. This

value corresponds to the MySQL account that determines your access privileges. It can be different from the value of USER().

```
mysql> SELECT USER();
        -> 'davida@localhost'
mysql> SELECT * FROM mysql.user;
ERROR 1044: Access denied for user ''@'localhost' to
database 'mysql'
mysql> SELECT CURRENT_USER();
        -> '@localhost'
```

The example illustrates that although the client specified a username of davida (as indicated by the value of the USER() function), the server authenticated the client using an anonymous user account (as seen by the empty username part of the CURRENT_USER() value). One way this might occur is that there is no account listed in the grant tables for davida.

CURRENT_USER() was added in MySQL 4.0.6. As of MySQL 4.1, the string has the utf8 character set.

- DATABASE()

  Returns the default (current) database name. As of MySQL 4.1, the string has the utf8 character set.

```
mysql> SELECT DATABASE();
        -> 'test'
```

  If there is no default database, DATABASE() returns NULL as of MySQL 4.1.1, and the empty string before that.

- FOUND_ROWS()

  A SELECT statement may include a LIMIT clause to restrict the number of rows the server returns to the client. In some cases, it is desirable to know how many rows the statement would have returned without the LIMIT, but without running the statement again. To get this row count, include a SQL_CALC_FOUND_ROWS option in the SELECT statement, then invoke FOUND_ROWS() afterward:

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name
    -> WHERE id > 100 LIMIT 10;
mysql> SELECT FOUND_ROWS();
```

  The second SELECT returns a number indicating how many rows the first SELECT would have returned had it been written without the LIMIT clause. (If the preceding SELECT statement does not include the SQL_CALC_FOUND_ROWS option, then FOUND_ROWS() may return a different result when LIMIT is used than when it is not.)

  Note that if you are using SELECT SQL_CALC_FOUND_ROWS, MySQL must calculate how many

rows are in the full result set. However, this is faster than running the query again without LIMIT, because the result set need not be sent to the client.

SQL_CALC_FOUND_ROWS and FOUND_ROWS() can be useful in situations when you want to restrict the number of rows that a query returns, but also determine the number of rows in the full result set without running the query again. An example is a Web script that presents a paged display containing links to the pages that show other sections of a search result. Using FOUND_ROWS() allows you to determine how many other pages are needed for the rest of the result.

The use of SQL_CALC_FOUND_ROWS and FOUND_ROWS() is more complex for UNION queries than for simple SELECT statements, because LIMIT may occur at multiple places in a UNION. It may be applied to individual SELECT statements in the UNION, or global to the UNION result as a whole.

The intent of SQL_CALC_FOUND_ROWS for UNION is that it should return the row count that would be returned without a global LIMIT. The conditions for use of SQL_CALC_FOUND_ROWS with UNION are:

- The SQL_CALC_FOUND_ROWS keyword must appear in the first SELECT of the UNION.
- The value of FOUND_ROWS() is exact only if UNION ALL is used. If UNION without ALL is used, duplicate removal occurs and the value of FOUND_ROWS() is only approximate.
- If no LIMIT is present in the UNION, SQL_CALC_FOUND_ROWS is ignored and returns the number of rows in the temporary table that is created to process the UNION.

SQL_CALC_FOUND_ROWS and FOUND_ROWS() are available starting at MySQL 4.0.0.

- LAST_INSERT_ID(), LAST_INSERT_ID(*expr*)

  Returns the last automatically generated value that was inserted into an AUTO_INCREMENT column.

```
mysql> SELECT LAST_INSERT_ID();
        -> 195
```

The last ID that was generated is maintained in the server on a per-connection basis. This means the value the function returns to a given client is the most recent AUTO_INCREMENT value generated by that client. The value cannot be affected by other clients, even if they generate AUTO_INCREMENT values of their own. This behavior ensures that you can retrieve your own ID without concern for the activity of other clients, and without the need for locks or transactions.

The value of LAST_INSERT_ID() is not changed if you update the AUTO_INCREMENT column of a row with a non-magic value (that is, a value that is not NULL and not 0).

If you insert many rows at the same time with an insert statement, LAST_INSERT_ID() returns the value for the first inserted row. The reason for this is to make it possible to easily reproduce the same INSERT statement against some other server.

If you use INSERT IGNORE and the record is ignored, the AUTO_INCREMENT counter is not incremented and LAST_INSERT_ID() returns 0, which reflects that no record was inserted. (Before MySQL 4.1, AUTO_INCREMENT counter still is incremented and LAST_INSERT_ID() returns the new value.)

If *expr* is given as an argument to LAST_INSERT_ID(), the value of the argument is returned by the function and is remembered as the next value to be returned by LAST_INSERT_ID(). This can be used to simulate sequences:

- Create a table to hold the sequence counter and initialize it:

  ```
  mysql> CREATE TABLE sequence (id INT NOT NULL);
  mysql> INSERT INTO sequence VALUES (0);
  ```

- Use the table to generate sequence numbers like this:

  ```
  mysql> UPDATE sequence SET id=LAST_INSERT_ID(id+1);
  mysql> SELECT LAST_INSERT_ID();
  ```

  The UPDATE statement increments the sequence counter and causes the next call to LAST_INSERT_ID() to return the updated value. The SELECT statement retrieves that value. The mysql_insert_id() C API function can also be used to get the value. See Section 22.2.3.33, "mysql_insert_id()".

You can generate sequences without calling LAST_INSERT_ID(), but the utility of using the function this way is that the ID value is maintained in the server as the last automatically generated value. It is multi-user safe because multiple clients can issue the UPDATE statement and get their own sequence value with the SELECT statement (or mysql_insert_id()), without affecting or being affected by other clients that generate their own sequence values.

Note that mysql_insert_id() is only updated after INSERT and UPDATE statements, so you cannot use the C API function to retrieve the value for LAST_INSERT_ID(*expr*) after executing other SQL statements like SELECT or SET.

- ROW_COUNT()

  ROW_COUNT() returns the number of rows updated, inserted, or deleted by the preceding statement. This is the same as the row count that the **mysql** client displays and the value from the mysql_affected_rows() C API function.

  ```
  mysql> INSERT INTO t VALUES(1),(2),(3);
  Query OK, 3 rows affected (0.00 sec)
  Records: 3  Duplicates: 0  Warnings: 0

  mysql> SELECT ROW_COUNT();
  +-------------+
  | ROW_COUNT() |
  +-------------+
  |           3 |
  +-------------+
  1 row in set (0.00 sec)

  mysql> DELETE FROM t WHERE i IN(1,2);
  Query OK, 2 rows affected (0.00 sec)

  mysql> SELECT ROW_COUNT();
  +-------------+
  | ROW_COUNT() |
  ```

```
+-------------+
|           2 |
+-------------+
1 row in set (0.00 sec)
```

`ROW_COUNT()` was added in MySQL 5.0.1.

- `SESSION_USER()`

  `SESSION_USER()` is a synonym for `USER()`.

- `SYSTEM_USER()`

  `SYSTEM_USER()` is a synonym for `USER()`.

- `USER()`

  Returns the current MySQL username and hostname.

```
mysql> SELECT USER();
        -> 'davida@localhost'
```

  The value indicates the username you specified when connecting to the server, and the client host from which you connected. The value can be different than that of `CURRENT_USER()`.

  Prior to MySQL 3.22.11, the function value does not include the client hostname. You can extract just the username part, regardless of whether the value includes a hostname part, like this:

```
mysql> SELECT SUBSTRING_INDEX(USER(),'@',1);
        -> 'davida'
```

  As of MySQL 4.1, `USER()` returns a value in the `utf8` character set, so you should also make sure that the `'@'` string literal is interpreted in that character set:

```
mysql> SELECT SUBSTRING_INDEX(USER(),_utf8'@',1);
        -> 'davida'
```

- `VERSION()`

  Returns a string that indicates the MySQL server version. As of MySQL 4.1, the string has the `utf8` character set.

```
mysql> SELECT VERSION();
```

```
        -> '4.1.3-beta-log'
```

Note that if your version string ends with `-log` this means that logging is enabled.

12.8.4. Miscellaneous Functions

- `DEFAULT(col_name)`

  Returns the default value for a table column. Starting from MySQL 5.0.2, you get an error if the column doesn't have a default value.

  ```
  mysql> UPDATE t SET i = DEFAULT(i)+1 WHERE id < 100;
  ```

  `DEFAULT()` was added in MySQL 4.1.0.

- `FORMAT(X,D)`

  Formats the number *X* to a format like `'#,###,###.##'`, rounded to *D* decimals, and returns the result as a string. If *D* is `0`, the result has no decimal point or fractional part.

  ```
  mysql> SELECT FORMAT(12332.123456, 4);
          -> '12,332.1235'
  mysql> SELECT FORMAT(12332.1,4);
          -> '12,332.1000'
  mysql> SELECT FORMAT(12332.2,0);
          -> '12,332'
  ```

- `GET_LOCK(str,timeout)`

  Tries to obtain a lock with a name given by the string *str*, with a timeout of *timeout* seconds. Returns `1` if the lock was obtained successfully, `0` if the attempt timed out (for example, because another client has previously locked the name), or `NULL` if an error occurred (such as running out of memory or the thread was killed with **mysqladmin kill**). If you have a lock obtained with `GET_LOCK()`, it is released when you execute `RELEASE_LOCK()`, execute a new `GET_LOCK()`, or your connection terminates (either normally or abnormally).

  This function can be used to implement application locks or to simulate record locks. Names are locked on a server-wide basis. If a name has been locked by one client, `GET_LOCK()` blocks any request by another client for a lock with the same name. This allows clients that agree on a given lock name to use the name to perform cooperative advisory locking.

  ```
  mysql> SELECT GET_LOCK('lock1',10);
          -> 1
  mysql> SELECT IS_FREE_LOCK('lock2');
  ```

```
        -> 1
mysql> SELECT GET_LOCK('lock2',10);
        -> 1
mysql> SELECT RELEASE_LOCK('lock2');
        -> 1
mysql> SELECT RELEASE_LOCK('lock1');
        -> NULL
```

Note that the second `RELEASE_LOCK()` call returns `NULL` because the lock `'lock1'` was automatically released by the second `GET_LOCK()` call.

- `INET_ATON(expr)`

  Given the dotted-quad representation of a network address as a string, returns an integer that represents the numeric value of the address. Addresses may be 4- or 8-byte addresses.

```
mysql> SELECT INET_ATON('209.207.224.40');
        -> 3520061480
```

  The generated number is always in network byte order. For the example just shown, the number is calculated as `209*256^3 + 207*256^2 + 224*256 + 40`.

  As of MySQL 4.1.2, `INET_ATON()` also understands short-form IP addresses:

```
mysql> SELECT INET_ATON('127.0.0.1'), INET_ATON('127.1');
        -> 2130706433, 2130706433
```

  `INET_ATON()` was added in MySQL 3.23.15.

- `INET_NTOA(expr)`

  Given a numeric network address (4 or 8 byte), returns the dotted-quad representation of the address as a string.

```
mysql> SELECT INET_NTOA(3520061480);
        -> '209.207.224.40'
```

  `INET_NTOA()` was added in MySQL 3.23.15.

- `IS_FREE_LOCK(str)`

  Checks whether the lock named *`str`* is free to use (that is, not locked). Returns `1` if the lock is free (no one is using the lock), `0` if the lock is in use, and `NULL` on errors (such as incorrect arguments).

  `IS_FREE_LOCK()` was added in MySQL 4.0.2.

- `IS_USED_LOCK(str)`

  Checks whether the lock named *`str`* is in use (that is, locked). If so, it returns the connection identifier of the client that holds the lock. Otherwise, it returns `NULL`.

  `IS_USED_LOCK()` was added in MySQL 4.1.0.

- `MASTER_POS_WAIT(log_name,log_pos[,timeout])`

  This function is useful for control of master/slave synchronization. It blocks until the slave has read and applied all updates up to the specified position in the master log. The return value is the number of log events it had to wait for to get to the specified position. The function returns `NULL` if the slave SQL thread is not started, the slave's master information is not initialized, the arguments are incorrect, or an error occurs. It returns `-1` if the timeout has been exceeded. If the slave SQL thread stops while `MASTER_POS_WAIT()` is waiting, the function returns `NULL`. If the slave is past the specified position, the function returns immediately.

  If a *`timeout`* value is specified, `MASTER_POS_WAIT()` stops waiting when *`timeout`* seconds have elapsed. *`timeout`* must be greater than 0; a zero or negative *`timeout`* means no timeout.

  `MASTER_POS_WAIT()` was added in MySQL 3.23.32. The *`timeout`* argument was added in 4.0.10.

- `RELEASE_LOCK(str)`

  Releases the lock named by the string *`str`* that was obtained with `GET_LOCK()`. Returns `1` if the lock was released, `0` if the lock wasn't locked by this thread (in which case the lock is not released), and `NULL` if the named lock didn't exist. The lock does not exist if it was never obtained by a call to `GET_LOCK()` or if it has previously been released.

  The `DO` statement is convenient to use with `RELEASE_LOCK()`. See Section 13.1.2, “`DO` Syntax”.

- `UUID()`

  Returns a Universal Unique Identifier (UUID) generated according to ``DCE 1.1: Remote Procedure Call'' (Appendix A) CAE (Common Applications Environment) Specifications published by The Open Group in October 1997 (Document Number C706).

  A UUID is designed as a number that is globally unique in space and time. Two calls to `UUID()` are expected to generate two different values, even if these calls are performed on two separate computers that are not connected to each other.

  A UUID is a 128-bit number represented by a string of five hexadecimal numbers in `aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee` format:

  - The first three numbers are generated from a timestamp.
  - The fourth number preserves temporal uniqueness in case the timestamp value loses monotonicity (for example, due to daylight saving time).

- The fifth number is an IEEE 802 node number that provides spatial uniqueness. A random number is substituted if the latter is not available (for example, because the host computer has no Ethernet card, or we do not know how to find the hardware address of an interface on your operating system). In this case, spatial uniqueness cannot be guaranteed. Nevertheless, a collision should have *very* low probability.

  Currently, the MAC address of an interface is taken into account only on FreeBSD and Linux. On other operating systems, MySQL uses a randomly generated 48-bit number.

```
mysql> SELECT UUID();
        -> '6ccd780c-baba-1026-9564-0040f4311e29'
```

Note that `UUID()` does not yet work with replication.

`UUID()` was added in MySQL 4.1.2.

12.9. Functions and Modifiers for Use with `GROUP BY` Clauses

12.9.1. `GROUP BY` (Aggregate) Functions

If you use a group function in a statement containing no `GROUP BY` clause, it is equivalent to grouping on all rows.

- `AVG([DISTINCT] expr)`

  Returns the average value of *`expr`*. The `DISTINCT` option can be used as of MySQL 5.0.3 to return the averge of the distinct values of *`expr`*.

```
mysql> SELECT student_name, AVG(test_score)
    ->        FROM student
    ->        GROUP BY student_name;
```

- `BIT_AND(expr)`

  Returns the bitwise `AND` of all bits in *`expr`*. The calculation is performed with 64-bit (`BIGINT`) precision.

  As of MySQL 4.0.17, this function returns `18446744073709551615` if there were no matching rows. (This is an unsigned `BIGINT` value with all bits set to 1.) Before 4.0.17, the function returns `-1` if there were no matching rows.

- `BIT_OR(expr)`

Returns the bitwise OR of all bits in *expr*. The calculation is performed with 64-bit (BIGINT) precision.

This function returns 0 if there were no matching rows.

- BIT_XOR(*expr*)

  Returns the bitwise XOR of all bits in *expr*. The calculation is performed with 64-bit (BIGINT) precision.

  This function returns 0 if there were no matching rows.

  This function is available as of MySQL 4.1.1.

- COUNT(*expr*)

  Returns a count of the number of non-NULL values in the rows retrieved by a SELECT statement.

```
mysql> SELECT student.student_name,COUNT(*)
    ->        FROM student,course
    ->        WHERE student.student_id=course.student_id
    ->        GROUP BY student_name;
```

  COUNT(*) is somewhat different in that it returns a count of the number of rows retrieved, whether or not they contain NULL values.

  COUNT(*) is optimized to return very quickly if the SELECT retrieves from one table, no other columns are retrieved, and there is no WHERE clause. For example:

```
mysql> SELECT COUNT(*) FROM student;
```

  This optimization applies only to MyISAM and ISAM tables only, because an exact record count is stored for these table types and can be accessed very quickly. For transactional storage engines (InnoDB, BDB), storing an exact row count is more problematic because multiple transactions may be occurring, each of which may affect the count.

- COUNT(DISTINCT *expr*,[*expr*...])

  Returns a count of the number of different non-NULL values.

```
mysql> SELECT COUNT(DISTINCT results) FROM student;
```

  In MySQL, you can get the number of distinct expression combinations that don't contain NULL by giving a list of expressions. In standard SQL, you would have to do a concatenation of all expressions inside COUNT(DISTINCT ...).

`COUNT(DISTINCT ...)` was added in MySQL 3.23.2.

- `GROUP_CONCAT(expr)`

  This function returns a string result with the concatenated non-`NULL` values from a group. It returns `NULL` if there are no non-`NULL` values. The full syntax is as follows:

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
             [ORDER BY {unsigned_integer | col_name | expr}
                 [ASC | DESC] [,col_name ...]]
             [SEPARATOR str_val])
```

```
mysql> SELECT student_name,
    ->     GROUP_CONCAT(test_score)
    ->     FROM student
    ->     GROUP BY student_name;
```

Or:

```
mysql> SELECT student_name,
    ->     GROUP_CONCAT(DISTINCT test_score
    ->                ORDER BY test_score DESC SEPARATOR ' ')
    ->     FROM student
    ->     GROUP BY student_name;
```

In MySQL, you can get the concatenated values of expression combinations. You can eliminate duplicate values by using `DISTINCT`. If you want to sort values in the result, you should use `ORDER BY` clause. To sort in reverse order, add the `DESC` (descending) keyword to the name of the column you are sorting by in the `ORDER BY` clause. The default is ascending order; this may be specified explicitly using the `ASC` keyword. `SEPARATOR` is followed by the string value that should be inserted between values of result. The default is a comma (`','`). You can remove the separator altogether by specifying `SEPARATOR ''`.

You can set a maximum allowed length with the `group_concat_max_len` system variable. The syntax to do this at runtime is as follows, where `val` is an unsigned integer:

```
SET [SESSION | GLOBAL] group_concat_max_len = val;
```

If a maximum length has been set, the result is truncated to this maximum length.

`GROUP_CONCAT()` was added in MySQL 4.1.

Note: Before MySQL 4.1.6, there are some small limitations with `GROUP_CONCAT()` for `BLOB`

and TEXT values when it comes to using DISTINCT together with ORDER BY. To work around this limitation, use MID(*expr*,1,255) instead.

- MIN([DISTINCT] *expr*),MAX([DISTINCT] *expr*)

  Returns the minimum or maximum value of *expr*. MIN() and MAX() may take a string argument; in such cases they return the minimum or maximum string value. See Section 7.4.5, “How MySQL Uses Indexes”. The DISTINCT keyword can be used as of MySQL 5.0.0 to find the minimum or maximum of the distinct values of *expr*; this is supported, but produces the same result as omitting DISTINCT.

```
mysql> SELECT student_name, MIN(test_score), MAX(test_score)
    ->        FROM student
    ->        GROUP BY student_name;
```

  For MIN(), MAX(), and other aggregate functions, MySQL currently compares ENUM and SET columns by their string value rather than by the string's relative position in the set. This differs from how ORDER BY compares them. This will be rectified.

- STD(*expr*),STDDEV(*expr*)

  Returns the population standard deviation of *expr*. This is an extension to standard SQL. The STDDEV() form of this function is provided for Oracle compatibility. As of MySQL 5.0.3, the standard SQL function STDDEV_POP() can be used instead.

- STDDEV_POP(*expr*)

  Returns the population standard deviation of *expr* (the square root of VAR_POP()). This function was added in MySQL 5.0.3. Before 5.0.3, you can use STD() or STDDEV(), which are equivalent but not standard SQL.

- STDDEV_SAMP(*expr*)

  Returns the sample standard deviation of *expr* (the square root of VAR_SAMP(). This function was added in MySQL 5.0.3.

- SUM([DISTINCT] *expr*)

  Returns the sum of *expr*. If the return set has no rows, SUM() returns NULL. The DISTINCT keyword can be used as of MySQL 5.0.0 to sum only the distinct values of *expr*.

- VAR_POP(*expr*)

  Returns the population standard variance of *expr*. It considers rows as the whole population, not as a sample, so it has the number of rows as the denominator. This function was added in MySQL

5.0.3. Before 5.0.3, you can use `VARIANCE()`, which is equivalent but not standard SQL.

- `VAR_SAMP(expr)`

  Returns the sample variance of *expr*. That is, the denominator is the number of rows minus one. This function was added in MySQL 5.0.3.

- `VARIANCE(expr)`

  Returns the population standard variance of *expr*. This is an extension to standard SQL, available in MySQL 4.1 or later. As of MySQL 5.0.3, the standard SQL function `VAR_POP()` can be used instead.

12.9.2. GROUP BY Modifiers

As of MySQL 4.1.1, the `GROUP BY` clause allows a `WITH ROLLUP` modifier that causes extra rows to be added to the summary output. These rows represent higher-level (or super-aggregate) summary operations. `ROLLUP` thus allows you to answer questions at multiple levels of analysis with a single query. It can be used, for example, to provide support for OLAP (Online Analytical Processing) operations.

As an illustration, suppose that a table named `sales` has `year`, `country`, `product`, and `profit` columns for recording sales profitability:

```
CREATE TABLE sales
(
    year    INT NOT NULL,
    country VARCHAR(20) NOT NULL,
    product VARCHAR(32) NOT NULL,
    profit  INT
);
```

The table's contents can be summarized per year with a simple `GROUP BY` like this:

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year;
+------+-------------+
| year | SUM(profit) |
+------+-------------+
| 2000 |        4525 |
| 2001 |        3010 |
+------+-------------+
```

This output shows the total profit for each year, but if you also want to determine the total profit summed over all years, you must add up the individual values yourself or run an additional query.

Or you can use `ROLLUP`, which provides both levels of analysis with a single query. Adding a `WITH ROLLUP` modifier to the `GROUP BY` clause causes the query to produce another row that shows the grand total over all year values:

```
mysql> SELECT year, SUM(profit) FROM sales GROUP BY year WITH ROLLUP;
+------+-------------+
| year | SUM(profit) |
+------+-------------+
| 2000 |        4525 |
```

```
| 2001 |        3010 |
| NULL |        7535 |
+------+-------------+
```

The grand total super-aggregate line is identified by the value `NULL` in the `year` column.

`ROLLUP` has a more complex effect when there are multiple `GROUP BY` columns. In this case, each time there is a ``break'' (change in value) in any but the last grouping column, the query produces an extra super-aggregate summary row.

For example, without `ROLLUP`, a summary on the `sales` table based on `year`, `country`, and `product` might look like this:

```
mysql> SELECT year, country, product, SUM(profit)
    -> FROM sales
    -> GROUP BY year, country, product;
+------+---------+------------+-------------+
| year | country | product    | SUM(profit) |
+------+---------+------------+-------------+
| 2000 | Finland | Computer   |        1500 |
| 2000 | Finland | Phone      |         100 |
| 2000 | India   | Calculator |         150 |
| 2000 | India   | Computer   |        1200 |
| 2000 | USA     | Calculator |          75 |
| 2000 | USA     | Computer   |        1500 |
| 2001 | Finland | Phone      |          10 |
| 2001 | USA     | Calculator |          50 |
| 2001 | USA     | Computer   |        2700 |
| 2001 | USA     | TV         |         250 |
+------+---------+------------+-------------+
```

The output indicates summary values only at the year/country/product level of analysis. When `ROLLUP` is added, the query produces several extra rows:

```
mysql> SELECT year, country, product, SUM(profit)
    -> FROM sales
    -> GROUP BY year, country, product WITH ROLLUP;
+------+---------+------------+-------------+
| year | country | product    | SUM(profit) |
+------+---------+------------+-------------+
| 2000 | Finland | Computer   |        1500 |
| 2000 | Finland | Phone      |         100 |
| 2000 | Finland | NULL       |        1600 |
| 2000 | India   | Calculator |         150 |
| 2000 | India   | Computer   |        1200 |
| 2000 | India   | NULL       |        1350 |
| 2000 | USA     | Calculator |          75 |
| 2000 | USA     | Computer   |        1500 |
| 2000 | USA     | NULL       |        1575 |
| 2000 | NULL    | NULL       |        4525 |
| 2001 | Finland | Phone      |          10 |
| 2001 | Finland | NULL       |          10 |
| 2001 | USA     | Calculator |          50 |
| 2001 | USA     | Computer   |        2700 |
| 2001 | USA     | TV         |         250 |
| 2001 | USA     | NULL       |        3000 |
| 2001 | NULL    | NULL       |        3010 |
| NULL | NULL    | NULL       |        7535 |
+------+---------+------------+-------------+
```

For this query, adding ROLLUP causes the output to include summary information at four levels of analysis, not just one. Here's how to interpret the ROLLUP output:

- Following each set of product rows for a given year and country, an extra summary row is produced showing the total for all products. These rows have the product column set to NULL.
- Following each set of rows for a given year, an extra summary row is produced showing the total for all countries and products. These rows have the country and products columns set to NULL.
- Finally, following all other rows, an extra summary row is produced showing the grand total for all years, countries, and products. This row has the year, country, and products columns set to NULL.

Other Considerations When using ROLLUP

The following items list some behaviors specific to the MySQL implementation of ROLLUP:

When you use ROLLUP, you cannot also use an ORDER BY clause to sort the results. In other words, ROLLUP and ORDER BY are mutually exclusive. However, you still have some control over sort order. GROUP BY in MySQL sorts results, and you can use explicit ASC and DESC keywords with columns named in the GROUP BY list to specify sort order for individual columns. (The higher-level summary rows added by ROLLUP still appear after the rows from which they are calculated, regardless of the sort order.)

LIMIT can be used to restrict the number of rows returned to the client. LIMIT is applied after ROLLUP, so the limit applies against the extra rows added by ROLLUP. For example:

```
mysql> SELECT year, country, product, SUM(profit)
    -> FROM sales
    -> GROUP BY year, country, product WITH ROLLUP
    -> LIMIT 5;
+------+---------+------------+-------------+
| year | country | product    | SUM(profit) |
+------+---------+------------+-------------+
| 2000 | Finland | Computer   |        1500 |
| 2000 | Finland | Phone      |         100 |
| 2000 | Finland | NULL       |        1600 |
| 2000 | India   | Calculator |         150 |
| 2000 | India   | Computer   |        1200 |
+------+---------+------------+-------------+
```

Using LIMIT with ROLLUP may produce results that are more difficult to interpret, because you have less context for understanding the super-aggregate rows.

The NULL indicators in each super-aggregate row are produced when the row is sent to the client. The server looks at the columns named in the GROUP BY clause following the leftmost one that has changed value. For any column in the result set with a name that is a lexical match to any of those names, its value is set to NULL. (If you specify grouping columns by column number, the server identifies which columns to set to NULL by number.)

Because the NULL values in the super-aggregate rows are placed into the result set at such a late stage in query processing, you cannot test them as NULL values within the query itself. For example, you cannot add HAVING product IS NULL to the query to eliminate from the output all but the super-aggregate rows.

On the other hand, the NULL values do appear as NULL on the client side and can be tested as such using any MySQL client programming interface.

12.9.3. GROUP BY with Hidden Fields

MySQL extends the use of GROUP BY so that you can use columns or calculations in the SELECT list that don't appear in the GROUP BY clause. This stands for *any possible value for this group*. You can use this to get better performance by avoiding sorting and grouping on unnecessary items. For example, you don't need to group on customer.name in the following query:

```
mysql> SELECT order.custid, customer.name, MAX(payments)
    ->        FROM order,customer
    ->        WHERE order.custid = customer.custid
    ->        GROUP BY order.custid;
```

In standard SQL, you would have to add customer.name to the GROUP BY clause. In MySQL, the name is redundant if you don't run in ANSI mode.

Do *not* use this feature if the columns you omit from the GROUP BY part are not unique in the group! You get unpredictable results.

In some cases, you can use MIN() and MAX() to obtain a specific column value even if it isn't unique. The following gives the value of column from the row containing the smallest value in the sort column:

```
SUBSTR(MIN(CONCAT(RPAD(sort,6,' '),column)),7)
```

See Section 3.6.4, “The Rows Holding the Group-wise Maximum of a Certain Field”.

Note that if you are using MySQL 3.22 (or earlier) or if you are trying to follow standard SQL, you can't use expressions in GROUP BY or ORDER BY clauses. You can work around this limitation by using an alias for the expression:

```
mysql> SELECT id,FLOOR(value/100) AS val FROM tbl_name
    ->     GROUP BY id, val ORDER BY val;
```

In MySQL 3.23 and up, aliases are unnecessary. You can use expressions in GROUP BY and ORDER BY clauses. For example:

```
mysql> SELECT id, FLOOR(value/100) FROM tbl_name ORDER BY RAND();
```

Chapter 13. SQL Statement Syntax

This chapter describes the syntax for the SQL statements supported in MySQL.

13.1. Data Manipulation Statements

13.1.1. DELETE Syntax

Single-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
       [WHERE where_definition]
       [ORDER BY ...]
       [LIMIT row_count]
```

Multiple-table syntax:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
       tbl_name[.*] [, tbl_name[.*] ...]
       FROM table_references
       [WHERE where_definition]
```

Or:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
       FROM tbl_name[.*] [, tbl_name[.*] ...]
       USING table_references
       [WHERE where_definition]
```

`DELETE` deletes rows from *`tbl_name`* that satisfy the condition given by *`where_definition`*, and returns the number of records deleted.

If you issue a `DELETE` statement with no `WHERE` clause, all rows are deleted. A faster way to do this, when you don't want to know the number of deleted rows, is to use `TRUNCATE TABLE`. See Section 13.1.9, “`TRUNCATE` Syntax”.

In MySQL 3.23, `DELETE` without a `WHERE` clause returns zero as the number of affected records.

In MySQL 3.23, if you really want to know how many records are deleted when you are deleting all rows, and are willing to suffer a speed penalty, you can use a `DELETE` statement that includes a `WHERE` clause with an expression that is true for every row. For example:

```
mysql> DELETE FROM tbl_name WHERE 1>0;
```

This is much slower than `TRUNCATE` *`tbl_name`*, because it deletes rows one at a time.

If you delete the row containing the maximum value for an `AUTO_INCREMENT` column, the value is reused for an `ISAM` or `BDB` table, but not for a `MyISAM` or `InnoDB` table. If you delete all rows in the table with `DELETE FROM` *`tbl_name`* (without a `WHERE`) in `AUTOCOMMIT` mode, the sequence starts over for all table types except for `InnoDB` and (as of MySQL 4.0) `MyISAM`. There are some exceptions to this behavior for `InnoDB` tables, discussed in Section 15.7.3, “How an `AUTO_INCREMENT` Column Works in `InnoDB`”.

For MyISAM and BDB tables, you can specify an AUTO_INCREMENT secondary column in a multiple-column key. In this case, reuse of values deleted from the top of the sequence occurs even for MyISAM tables. See Section 3.6.9, “Using AUTO_INCREMENT”.

The DELETE statement supports the following modifiers:

- If you specify the LOW_PRIORITY keyword, execution of the DELETE is delayed until no other clients are reading from the table.
- For MyISAM tables, if you specify the QUICK keyword, the storage engine does not merge index leaves during delete, which may speed up certain kind of deletes.
- The IGNORE keyword causes MySQL to ignore all errors during the process of deleting rows. (Errors encountered during the parsing stage are processed in the usual manner.) Errors that are ignored due to the use of this option are returned as warnings. This option first appeared in MySQL 4.1.1.

The speed of delete operations may also be affected by factors discussed in Section 7.2.16, “Speed of DELETE Statements”.

In MyISAM tables, deleted records are maintained in a linked list and subsequent INSERT operations reuse old record positions. To reclaim unused space and reduce file sizes, use the OPTIMIZE TABLE statement or the **myisamchk** utility to reorganize tables. OPTIMIZE TABLE is easier, but **myisamchk** is faster. See Section 13.5.2.5, “OPTIMIZE TABLE Syntax” and Section 5.7.3.10, “Table Optimization”.

The QUICK modifier affects whether index leaves are merged for delete operations. DELETE QUICK is most useful for applications where index values for deleted rows are replaced by similar index values from rows inserted later. In this case, the holes left by deleted values are reused.

DELETE QUICK is not useful when deleted values lead to underfilled index blocks spanning a range of index values for which new inserts occur again. In this case, use of QUICK can lead to wasted space in the index that remains unreclaimed. Here is an example of such a scenario:

1. Create a table that contains an indexed AUTO_INCREMENT column.
2. Insert many records into the table. Each insert results in an index values that is added to the high end of the index.
3. Delete a block of records at the low end of the column range using DELETE QUICK.

In this scenario, the index blocks associated with the deleted index values become underfilled but are not merged with other index blocks due to the use of QUICK. They remain underfilled when new inserts occur, because new records does not have index values in the deleted range. Furthermore, they remain underfilled even if you later use DELETE without QUICK, unless some of the deleted index values happen to lie in index blocks within or adjacent to the underfilled blocks. To reclaim unused index space under these circumstances, you can use OPTIMIZE TABLE.

If you are going to delete many rows from a table, it might be faster to use DELETE QUICK followed by OPTIMIZE TABLE. This rebuilds the index rather than performing many index block merge operations.

The MySQL-specific LIMIT *row_count* option to DELETE tells the server the maximum number of rows to be deleted before control is returned to the client. This can be used to ensure that a specific DELETE statement doesn't take too much time. You can simply repeat the DELETE statement until the

number of affected rows is less than the LIMIT value.

If the DELETE statement includes an ORDER BY clause, the rows are deleted in the order specified by the clause. This is really useful only in conjunction with LIMIT. For example, the following statement finds rows matching the WHERE clause, sorts them in timestamp order, and deletes the first (oldest) one:

```
DELETE FROM somelog
WHERE user = 'jcole'
ORDER BY timestamp
LIMIT 1
```

ORDER BY can be used with DELETE beginning with MySQL 4.0.0.

From MySQL 4.0, you can specify multiple tables in the DELETE statement to delete rows from one or more tables depending on a particular condition in multiple tables. However, you cannot use ORDER BY or LIMIT in a multiple-table DELETE.

The first multiple-table DELETE syntax is supported starting from MySQL 4.0.0. The second is supported starting from MySQL 4.0.2. The table_references part lists the tables involved in the join. Its syntax is described in Section 13.1.7.1, "JOIN Syntax".

For the first syntax, only matching rows from the tables listed before the FROM clause are deleted. For the second syntax, only matching rows from the tables listed in the FROM clause (before the USING clause) are deleted. The effect is that you can delete rows from many tables at the same time and also have additional tables that are used for searching:

```
DELETE t1, t2 FROM t1, t2, t3 WHERE t1.id=t2.id AND t2.id=t3.id;
```

Or:

```
DELETE FROM t1, t2 USING t1, t2, t3 WHERE t1.id=t2.id AND t2.id=t3.id;
```

These statements use all three files when searching for rows to delete, but delete matching rows only from tables t1 and t2.

The examples show inner joins using the comma operator, but multiple-table DELETE statements can use any type of join allowed in SELECT statements, such as LEFT JOIN.

The syntax allows .* after the table names for compatibility with Access.

If you use a multiple-table DELETE statement involving InnoDB tables for which there are foreign key constraints, the MySQL optimizer might process tables in an order that differs from that of their parent/child relationship. In this case, the statement fails and rolls back. Instead, delete from a single table and rely on the ON DELETE capabilities that InnoDB provides to cause the other tables to be modified accordingly.

Note: In MySQL 4.0, you should refer to the table names to be deleted with the true table name. In MySQL 4.1, you must use the alias (if one was given) when referring to a table name:

In MySQL 4.0:

```
DELETE test FROM test AS t1, test2 WHERE ...
```

In MySQL 4.1:

```
DELETE t1 FROM test AS t1, test2 WHERE ...
```

The reason we didn't make this change in 4.0 is that we didn't want to break any old 4.0 applications that were using the old syntax.

Currently, you cannot delete from a table and select from the same table in a subquery.

Cross-database deletes are supported for multiple-table deletes, but in this case, you must refer to the tables without using aliases. For example:

```
DELETE test1.tmp1, test2.tmp2 FROM test1.tmp1, test2.tmp2 WHERE ...
```

13.1.2. DO Syntax

```
DO expr [, expr] ...
```

DO executes the expressions but doesn't return any results. This is shorthand for SELECT *expr*, ..., but has the advantage that it's slightly faster when you don't care about the result.

DO is useful mainly with functions that have side effects, such as RELEASE_LOCK().

DO was added in MySQL 3.23.47.

13.1.3. HANDLER Syntax

```
HANDLER tbl_name OPEN [ AS alias ]
HANDLER tbl_name READ index_name { = | >= | <= | < } (value1,value2,...)
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ index_name { FIRST | NEXT | PREV | LAST }
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ { FIRST | NEXT }
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name CLOSE
```

The HANDLER statement provides direct access to table storage engine interfaces. It is available for MyISAM tables as MySQL 4.0.0 and InnoDB tables as of MySQL 4.0.3.

The HANDLER ... OPEN statement opens a table, making it accessible via subsequent HANDLER ... READ statements. This table object is not shared by other threads and is not closed until the thread calls HANDLER ... CLOSE or the thread terminates. If you open the table using an alias, further references to the table with other HANDLER statements must use the alias rather than the table name.

The first HANDLER ... READ syntax fetches a row where the index specified satisfies the given values and the WHERE condition is met. If you have a multiple-column index, specify the index column values as a comma-separated list. Either specify values for all the columns in the index, or specify values for a leftmost prefix of the index columns. Suppose that an index includes three columns named col_a, col_b, and col_c, in that order. The HANDLER statement can specify values for all three columns in the index, or for the columns in a leftmost prefix. For example:

```
HANDLER ... index_name = (col_a_val,col_b_val,col_c_val) ...
HANDLER ... index_name = (col_a_val,col_b_val) ...
HANDLER ... index_name = (col_a_val) ...
```

The second HANDLER ... READ syntax fetches a row from the table in index order that matches WHERE condition.

The third `HANDLER ... READ` syntax fetches a row from the table in natural row order that matches the `WHERE` condition. It is faster than `HANDLER tbl_name READ index_name` when a full table scan is desired. Natural row order is the order in which rows are stored in a `MyISAM` table data file. This statement works for `InnoDB` tables as well, but there is no such concept because there is no separate data file.

Without a `LIMIT` clause, all forms of `HANDLER ... READ` fetch a single row if one is available. To return a specific number of rows, include a `LIMIT` clause. It has the same syntax as for the `SELECT` statement. See Section 13.1.7, "`SELECT` Syntax".

`HANDLER ... CLOSE` closes a table that was opened with `HANDLER ... OPEN`.

Note: To use the `HANDLER` interface to refer to a table's `PRIMARY KEY`, use the quoted identifier `` `PRIMARY` ``:

```
HANDLER tbl_name READ `PRIMARY` > (...);
```

`HANDLER` is a somewhat low-level statement. For example, it does not provide consistency. That is, `HANDLER ... OPEN` does *not* take a snapshot of the table, and does *not* lock the table. This means that after a `HANDLER ... OPEN` statement is issued, table data can be modified (by this or any other thread) and these modifications might appear only partially in `HANDLER ... NEXT` or `HANDLER ... PREV` scans.

There are several reasons to use the `HANDLER` interface instead of normal `SELECT` statements:

- `HANDLER` is faster than `SELECT`:
  - A designated storage engine handler object is allocated for the `HANDLER ... OPEN`. The object is reused for the following `HANDLER` statements for the table; it need not be reinitialized for each one.
  - There is less parsing involved.
  - There is no optimizer or query-checking overhead.
  - The table doesn't have to be locked between two handler requests.
  - The handler interface doesn't have to provide a consistent look of the data (for example, dirty reads are allowed), so the storage engine can use optimizations that `SELECT` doesn't normally allow.
- `HANDLER` makes it much easier to port applications that use an `ISAM`-like interface to MySQL.
- `HANDLER` allows you to traverse a database in a manner that is not easy (or perhaps even impossible) to do with `SELECT`. The `HANDLER` interface is a more natural way to look at data when working with applications that provide an interactive user interface to the database.

13.1.4. INSERT Syntax

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    VALUES ({expr | DEFAULT},...),(...),...
    [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

Or:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    SET col_name={expr | DEFAULT}, ...
    [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

Or:

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    SELECT ...
    [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

`INSERT` inserts new rows into an existing table. The `INSERT ... VALUES` and `INSERT ... SET` forms of the statement insert rows based on explicitly specified values. The `INSERT ... SELECT` form inserts rows selected from another table or tables. The `INSERT ... VALUES` form with multiple value lists is supported in MySQL 3.22.5 or later. The `INSERT ... SET` syntax is supported in MySQL 3.22.10 or later. `INSERT ... SELECT` is discussed further in See Section 13.1.4.1, "`INSERT ... SELECT` Syntax".

`tbl_name` is the table into which rows should be inserted. The columns for which the statement provides values can be specified as follows:

- The column name list or the `SET` clause indicates the columns explicitly.
- If you do not specify the column list for `INSERT ... VALUES` or `INSERT ... SELECT`, values for every column in the table must be provided in the `VALUES()` list or by the `SELECT`. If you don't know the order of the columns in the table, use `DESCRIBE tbl_name` to find out.

Column values can be given in several ways:

- If you are not running in strict mode, any column not explicitly given a value is set to its default (explicit or implicit) value. For example, if you specify a column list that doesn't name all the columns in the table, unnamed columns are set to their default values. Default value assignment is described in Section 13.2.6, "`CREATE TABLE` Syntax". See Section 1.5.6.2, "Constraints on Invalid Data".

  If you want `INSERT` statements to generate an error unless you explicitly specify values for all columns that don't have a default value, you should use STRICT mode. See Section 5.2.2, "The Server SQL Mode".
- You can use the keyword `DEFAULT` to explicitly set a column to its default value. (New in MySQL 4.0.3.) This makes it easier to write `INSERT` statements that assign values to all but a few columns, because it allows you to avoid writing an incomplete `VALUES` list that does not include a value for each column in the table. Otherwise, you would have to write out the list of column names corresponding to each value in the `VALUES` list.

  As of MySQL 4.1.0, you can use `DEFAULT(col_name)` as a more general form that can be used in expressions to produce a column's default value.
- If both the column list and the `VALUES` list are empty, `INSERT` creates a row with each column set to its default value:

  ```
  mysql> INSERT INTO tbl_name () VALUES();
  ```

- You can specify an expression *expr* to provide a column value. This might involve type conversion if the type of the expression does not match the type of the column, and conversion of a given value can result in different inserted values depending on the column type. For example, inserting the string `'1999.0e-2'` into an `INT`, `FLOAT`, `DECIMAL(10,6)`, or `YEAR` column results in the values `1999`, `19.9921`, `19.992100`, and `1999`. The reason the value stored in the `INT` and `YEAR` columns is `1999` is that the string-to-integer conversion looks only at as much of the initial part of the string as may be considered a valid integer or year. For the floating-point and fixed-point columns, the string-to-floating-point conversion considers the entire string as a valid floating-point value.

  An expression *expr* can refer to any column that was set earlier in a value list. For example, you can do this because the value for `col2` refers to `col1`, which has previously been assigned:

  ```
  mysql> INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
  ```

  But you cannot do this because the value for `col1` refers to `col2`, which is assigned after `col1`:

  ```
  mysql> INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
  ```

  One exception involves columns that contain `AUTO_INCREMENT` values. Because the AUTO_INCREMENT value is generated after other value assignments, any reference to an AUTO_INCREMENT column in the assignment returns a 0.

The `INSERT` statement supports the following modifiers:

- If you specify the `DELAYED` keyword, the server puts the row or rows to be inserted into a buffer, and the client issuing the `INSERT DELAYED` statement then can continue on. If the table is busy, the server holds the rows. When the table becomes free, it begins inserting rows, checking periodically to see whether there are new read requests for the table. If there are, the delayed row queue is suspended until the table becomes free again. See Section 13.1.4.2, "`INSERT DELAYED` Syntax". `DELAYED` was added in MySQL 3.22.5.

- If you specify the `LOW_PRIORITY` keyword, execution of the `INSERT` is delayed until no other clients are reading from the table. This includes other clients that began reading while existing clients are reading, and while the `INSERT LOW_PRIORITY` statement is waiting. It is possible, therefore, for a client that issues an `INSERT LOW_PRIORITY` statement to wait for a very long time (or even forever) in a read-heavy environment. (This is in contrast to `INSERT DELAYED`, which lets the client continue at once.) See Section 13.1.4.2, "`INSERT DELAYED` Syntax". Note that `LOW_PRIORITY` should normally not be used with `MyISAM` tables because doing so disables concurrent inserts. See Section 14.1, "The `MyISAM` Storage Engine". `LOW_PRIORITY` was added in MySQL 3.22.5.

- If you specify the `HIGH_PRIORITY` keyword, it overrides the effect of the `--low-priority-updates` option if the server was started with that option. It also causes concurrent inserts not to be used. `HIGH_PRIORITY` was added in MySQL 3.23.11.

- The rows-affected value for an `INSERT` can be obtained using the `mysql_affected_rows()` C API function. See Section 22.2.3.1, "`mysql_affected_rows()`".

- If you specify the `IGNORE` keyword in an `INSERT` statement, errors that occur while executing the statement are treated as warnings instead. For example, without `IGNORE`, a row that duplicates an existing `UNIQUE` index or `PRIMARY KEY` value in the table causes a duplicate-key error and the statement is aborted. With `IGNORE`, the error is ignored and the row is not inserted. Data conversions that would trigger errors abort the statement if `IGNORE` is not specified. With `IGNORE`, invalid values are adjusted to the closest value values and inserted; warnings are produced but the state-

ment does not abort. You can determine with the `mysql_info()` C API function how many rows were inserted into the table.

If you specify the `ON DUPLICATE KEY UPDATE` clause (new in MySQL 4.1.0), and a row is inserted that would cause a duplicate value in a `UNIQUE` index or `PRIMARY KEY`, an `UPDATE` of the old row is performed. For example, if column `a` is declared as `UNIQUE` and contains the value `1`, the following two statements have identical effect:

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3)
    -> ON DUPLICATE KEY UPDATE c=c+1;

mysql> UPDATE table SET c=c+1 WHERE a=1;
```

The rows-affected value is 1 if the row is inserted as a new record and 2 if an existing record is updated.

Note: If column `b` is unique too, the `INSERT` would be equivalent to this `UPDATE` statement instead:

```
mysql> UPDATE table SET c=c+1 WHERE a=1 OR b=2 LIMIT 1;
```

If `a=1 OR b=2` matches several rows, only *one* row is updated! In general, you should try to avoid using the `ON DUPLICATE KEY` clause on tables with multiple `UNIQUE` keys.

As of MySQL 4.1.1, you can use the `VALUES(col_name)` function in the `UPDATE` clause to refer to column values from the `INSERT` part of the `INSERT ... UPDATE` statement. In other words, `VALUES(col_name)` in the `UPDATE` clause refers to the value of *`col_name`* that would be inserted if no duplicate-key conflict occurred. This function is especially useful in multiple-row inserts. The `VALUES()` function is meaningful only in `INSERT ... UPDATE` statements and returns `NULL` otherwise.

Example:

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
    -> ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

That statement is identical to the following two statements:

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3)
    -> ON DUPLICATE KEY UPDATE c=3;
mysql> INSERT INTO table (a,b,c) VALUES (4,5,6)
    -> ON DUPLICATE KEY UPDATE c=9;
```

When you use `ON DUPLICATE KEY UPDATE`, the `DELAYED` option is ignored.

You can find the value used for an `AUTO_INCREMENT` column by using the `LAST_INSERT_ID()` function. From within the C API, use the `mysql_insert_id()` function. However, note that the two functions do not behave quite identically under all circumstances. The behavior of `INSERT` statements with respect to `AUTO_INCREMENT` columns is discussed further in Section 12.8.3, “Information Functions” and Section 22.2.3.33, “`mysql_insert_id()`”.

If you use an `INSERT ... VALUES` statement with multiple value lists or `INSERT ... SELECT`, the statement returns an information string in this format:

```
Records: 100 Duplicates: 0 Warnings: 0
```

`Records` indicates the number of rows processed by the statement. (This is not necessarily the number

of rows actually inserted. `Duplicates` can be non-zero.) `Duplicates` indicates the number of rows that couldn't be inserted because they would duplicate some existing unique index value. `Warnings` indicates the number of attempts to insert column values that were problematic in some way. Warnings can occur under any of the following conditions:

- Inserting `NULL` into a column that has been declared `NOT NULL`. For multiple-row `INSERT` statements or `INSERT ... SELECT` statements, the column is set to the default value appropriate for the column type. This is `0` for numeric types, the empty string (`''`) for string types, and the ``zero'' value for date and time types.
- Setting a numeric column to a value that lies outside the column's range. The value is clipped to the closest endpoint of the range.
- Assigning a value such as `'10.34 a'` to a numeric column. The trailing non-numeric text is stripped off and the remaining numeric part is inserted. If the string value has no leading numeric part, the column is set to `0`.
- Inserting a string into a string column (`CHAR`, `VARCHAR`, `TEXT`, or `BLOB`) that exceeds the column's maximum length. The value is truncated to the column's maximum length.
- Inserting a value into a date or time column that is illegal for the column type. The column is set to the appropriate zero value for the type.

If you are using the C API, the information string can be obtained by invoking the `mysql_info()` function. See Section 22.2.3.31, “`mysql_info()`”.

13.1.4.1. `INSERT ... SELECT` Syntax

```
INSERT [LOW_PRIORITY] [IGNORE] [INTO] tbl_name [(column_list)]
    SELECT ...
```

With `INSERT ... SELECT`, you can quickly insert many rows into a table from one or many tables.

For example:

```
INSERT INTO tbl_temp2 (fld_id)
    SELECT tbl_temp1.fld_order_id
    FROM tbl_temp1 WHERE tbl_temp1.fld_order_id > 100;
```

The following conditions hold for an `INSERT ... SELECT` statement:

- Prior to MySQL 4.0.1, `INSERT ... SELECT` implicitly operates in `IGNORE` mode. As of MySQL 4.0.1, specify `IGNORE` explicitly to ignore records that would cause duplicate-key violations.
- Do not use `DELAYED` with `INSERT ... SELECT`.
- Prior to MySQL 4.0.14, the target table of the `INSERT` statement cannot appear in the `FROM` clause of the `SELECT` part of the query. This limitation is lifted in 4.0.14.
- `AUTO_INCREMENT` columns work as usual.
- To ensure that the binary log can be used to re-create the original tables, MySQL does not allow concurrent inserts during `INSERT ... SELECT`.

- Currently, you cannot insert into a table and select from the same table in a subquery.

You can use REPLACE instead of INSERT to overwrite old rows. REPLACE is the counterpart to INSERT IGNORE in the treatment of new rows that contain unique key values that duplicate old rows: The new rows are used to replace the old rows rather than being discarded.

13.1.4.2. INSERT DELAYED Syntax

```
INSERT DELAYED ...
```

The DELAYED option for the INSERT statement is a MySQL extension to standard SQL that is very useful if you have clients that can't wait for the INSERT to complete. This is a common problem when you use MySQL for logging and you also periodically run SELECT and UPDATE statements that take a long time to complete. DELAYED was introduced in MySQL 3.22.15.

When a client uses INSERT DELAYED, it gets an okay from the server at once, and the row is queued to be inserted when the table is not in use by any other thread.

Another major benefit of using INSERT DELAYED is that inserts from many clients are bundled together and written in one block. This is much faster than doing many separate inserts.

There are some constraints on the use of DELAYED:

- INSERT DELAYED works only with MyISAM and ISAM tables. For MyISAM tables, if there are no free blocks in the middle of the data file, concurrent SELECT and INSERT statements are supported. Under these circumstances, you very seldom need to use INSERT DELAYED with MyISAM. See Section 14.1, “The MyISAM Storage Engine”.
- INSERT DELAYED should be used only for INSERT statements that specify value lists. This is enforced as of MySQL 4.0.18. The server ignores DELAYED for INSERT DELAYED ... SELECT statements.
- The server ignores DELAYED for INSERT DELAYED ... ON DUPLICATE UPDATE statements.
- Because the statement returns immediately before the rows are inserted, you cannot use LAST_INSERT_ID() to get the AUTO_INCREMENT value the statement might generate.
- DELAYED rows are not visible to SELECT statements until they actually have been inserted.

Note that currently the queued rows are held only in memory until they are inserted into the table. This means that if you terminate **mysqld** forcibly (for example, with kill -9) or if **mysqld** dies unexpectedly, any queued rows that have not been written to disk are lost!

The following describes in detail what happens when you use the DELAYED option to INSERT or REPLACE. In this description, the ``thread'' is the thread that received an INSERT DELAYED statement and ``handler'' is the thread that handles all INSERT DELAYED statements for a particular table.

- When a thread executes a DELAYED statement for a table, a handler thread is created to process all DELAYED statements for the table, if no such handler previously exists.
- The thread checks whether the handler has previously acquired a DELAYED lock; if not, it tells the handler thread to do so. The DELAYED lock can be obtained even if other threads have a READ or WRITE lock on the table. However, the handler waits for all ALTER TABLE locks or FLUSH

TABLES to ensure that the table structure is up to date.

- The thread executes the INSERT statement, but instead of writing the row to the table, it puts a copy of the final row into a queue that is managed by the handler thread. Any syntax errors are noticed by the thread and reported to the client program.
- The client cannot obtain from the server the number of duplicate records or the AUTO_INCREMENT value for the resulting row, because the INSERT returns before the insert operation has been completed. (If you use the C API, the mysql_info() function doesn't return anything meaningful, for the same reason.)
- The binary log is updated by the handler thread when the row is inserted into the table. In case of multiple-row inserts, the binary log is updated when the first row is inserted.
- After every delayed_insert_limit rows are written, the handler checks whether any SELECT statements are still pending. If so, it allows these to execute before continuing.
- When the handler has no more rows in its queue, the table is unlocked. If no new INSERT DELAYED statements are received within delayed_insert_timeout seconds, the handler terminates.
- If more than delayed_queue_size rows are pending in a specific handler queue, the thread requesting INSERT DELAYED waits until there is room in the queue. This is done to ensure that the **mysqld** server doesn't use all memory for the delayed memory queue.
- The handler thread shows up in the MySQL process list with delayed_insert in the Command column. It is killed if you execute a FLUSH TABLES statement or kill it with KILL thread_id. However, before exiting, it first stores all queued rows into the table. During this time it does not accept any new INSERT statements from another thread. If you execute an INSERT DELAYED statement after this, a new handler thread is created.

  Note that this means that INSERT DELAYED statements have higher priority than normal INSERT statements if there is an INSERT DELAYED handler running. Other update statements have to wait until the INSERT DELAYED queue is empty, someone terminates the handler thread (with KILL thread_id), or someone executes FLUSH TABLES.
- The following status variables provide information about INSERT DELAYED statements:

Status Variable	Meaning
Delayed_insert_threads	Number of handler threads
Delayed_writes	Number of rows written with INSERT DELAYED
Not_flushed_delayed_rows	Number of rows waiting to be written

  You can view these variables by issuing a SHOW STATUS statement or by executing a **mysqladmin extended-status** command.

Note that INSERT DELAYED is slower than a normal INSERT if the table is not in use. There is also the additional overhead for the server to handle a separate thread for each table for which there are delayed rows. This means that you should use INSERT DELAYED only when you are really sure that you need it!

13.1.5. LOAD DATA INFILE Syntax

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name.txt'
    [REPLACE | IGNORE]
```

```
INTO TABLE tbl_name
[FIELDS
    [TERMINATED BY '\t']
    [[OPTIONALLY] ENCLOSED BY '']
    [ESCAPED BY '\\' ]
]
[LINES
    [STARTING BY '']
    [TERMINATED BY '\n']
]
[IGNORE number LINES]
[(col_name,...)]
```

The `LOAD DATA INFILE` statement reads rows from a text file into a table at a very high speed. For more information about the efficiency of `INSERT` versus `LOAD DATA INFILE` and speeding up `LOAD DATA INFILE`, Section 7.2.14, “Speed of `INSERT` Statements”.

You can also load data files by using the **mysqlimport** utility; it operates by sending a `LOAD DATA INFILE` statement to the server. The `--local` option causes **mysqlimport** to read data files from the client host. You can specify the `--compress` option to get better performance over slow networks if the client and server support the compressed protocol. See Section 8.10, “The **mysqlimport** Data Import Program”.

If you specify the `LOW_PRIORITY` keyword, execution of the `LOAD DATA` statement is delayed until no other clients are reading from the table.

If you specify the `CONCURRENT` keyword with a `MyISAM` table that satisfies the condition for concurrent inserts (that is, it contains no free blocks in the middle), then other threads can retrieve data from the table while `LOAD DATA` is executing. Using this option affects the performance of `LOAD DATA` a bit, even if no other thread is using the table at the same time.

If the `LOCAL` keyword is specified, it is interpreted with respect to the client end of the connection:

- If `LOCAL` is specified, the file is read by the client program on the client host and sent to the server. The file can be given as a full pathname to specify its exact location. If given as a relative pathname, the name is interpreted relative to the directory in which the client program was started.
- If `LOCAL` is not specified, the file must be located on the server host and is read directly by the server.

`LOCAL` is available in MySQL 3.22.6 or later.

When locating files on the server host, the server uses the following rules:

- If an absolute pathname is given, the server uses the pathname as is.
- If a relative pathname with one or more leading components is given, the server searches for the file relative to the server's data directory.
- If a filename with no leading components is given, the server looks for the file in the database directory of the default database.

Note that these rules mean that a file named as `./myfile.txt` is read from the server's data directory, whereas the same file named as `myfile.txt` is read from the database directory of the default database. For example, the following `LOAD DATA` statement reads the file `data.txt` from the database directory for `db1` because `db1` is the current database, even though the statement explicitly loads the

file into a table in the db2 database:

```
mysql> USE db1;
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE db2.my_table;
```

Note that Windows pathnames are specified using forward slashes rather than backslashes. If you do use backslashes, you must double them.

For security reasons, when reading text files located on the server, the files must either reside in the database directory or be readable by all. Also, to use LOAD DATA INFILE on server files, you must have the FILE privilege.

See Section 5.5.3, “Privileges Provided by MySQL”.

Using LOCAL is a bit slower than letting the server access the files directly, because the contents of the file must be sent over the connection by the client to the server. On the other hand, you do not need the FILE privilege to load local files.

As of MySQL 3.23.49 and MySQL 4.0.2 (4.0.13 on Windows), LOCAL works only if your server and your client both have been enabled to allow it. For example, if **mysqld** was started with --local-infile=0, LOCAL does not work. See Section 5.4.4, “Security Issues with LOAD DATA LOCAL”.

If you need LOAD DATA to read from a pipe, you can use the following technique (here we load the listing of the '/' directory into a table):

```
mkfifo /mysql/db/x/x
chmod 666 /mysql/db/x/x
find / -ls > /mysql/db/x/x
mysql -e "LOAD DATA INFILE 'x' INTO TABLE x" x
```

If you are using a version of MySQL older than 3.23.25, you can use this technique only with LOAD DATA LOCAL INFILE.

If you are using MySQL before Version 3.23.24, you can't read from a FIFO with LOAD DATA INFILE. If you need to read from a FIFO (for example, the output from gunzip), use LOAD DATA LOCAL INFILE instead.

The REPLACE and IGNORE keywords control handling of input records that duplicate existing records on unique key values.

If you specify REPLACE, input rows replace existing rows (in other words, rows that have the same value for a primary or unique index as an existing row). See Section 13.1.6, “REPLACE Syntax”.

If you specify IGNORE, input rows that duplicate an existing row on a unique key value are skipped. If you don't specify either option, the behavior depends on whether or not the LOCAL keyword is specified. Without LOCAL, an error occurs when a duplicate key value is found, and the rest of the text file is ignored. With LOCAL, the default behavior is the same as if IGNORE is specified; this is because the server has no way to stop transmission of the file in the middle of the operation.

If you want to ignore foreign key constraints during the load operation, you can issue a SET FOREIGN_KEY_CHECKS=0 statement before executing LOAD DATA.

If you use LOAD DATA INFILE on an empty MyISAM table, all non-unique indexes are created in a separate batch (as for REPAIR TABLE). This normally makes LOAD DATA INFILE much faster when you have many indexes. Normally this is very fast, but in some extreme cases, you can create the indexes even faster by turning them off with ALTER TABLE .. DISABLE KEYS before loading the file into the table and using ALTER TABLE .. ENABLE KEYS to re-create the indexes after loading

the file. See Section 7.2.14, “Speed of INSERT Statements”.

LOAD DATA INFILE is the complement of SELECT ... INTO OUTFILE. See Section 13.1.7, “SELECT Syntax”. To write data from a table to a file, use SELECT ... INTO OUTFILE. To read the file back into a table, use LOAD DATA INFILE. The syntax of the FIELDS and LINES clauses is the same for both statements. Both clauses are optional, but FIELDS must precede LINES if both are specified.

If you specify a FIELDS clause, each of its subclauses (TERMINATED BY, [OPTIONALLY] ENCLOSED BY, and ESCAPED BY) is also optional, except that you must specify at least one of them.

If you don't specify a FIELDS clause, the defaults are the same as if you had written this:

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
```

If you don't specify a LINES clause, the default is the same as if you had written this:

```
LINES TERMINATED BY '\n' STARTING BY ''
```

In other words, the defaults cause LOAD DATA INFILE to act as follows when reading input:

- Look for line boundaries at newlines.
- Do not skip over any line prefix.
- Break lines into fields at tabs.
- Do not expect fields to be enclosed within any quoting characters.
- Interpret occurrences of tab, newline, or '\' preceded by '\' as literal characters that are part of field values.

Conversely, the defaults cause SELECT ... INTO OUTFILE to act as follows when writing output:

- Write tabs between fields.
- Do not enclose fields within any quoting characters.
- Use '\' to escape instances of tab, newline, or '\' that occur within field values.
- Write newlines at the ends of lines.

Note that to write FIELDS ESCAPED BY '\\', you must specify two backslashes for the value to be read as a single backslash.

Note: If you have generated the text file on a Windows system, you might have to use LINES TERMINATED BY '\r\n' to read the file properly, because Windows programs typically use two characters as a line terminator. Some programs, such as WordPad, might use \r as a line terminator when writing files. To read such files, use LINES TERMINATED BY '\r'.

If all the lines you want to read in have a common prefix that you want to ignore, you can use LINES STARTING BY 'prefix_string' to skip over the prefix (and anything before it). If a line doesn't include the prefix, the entire line is skipped. **Note** that prefix_string may be in the middle of the line!

Example:

```
mysql> LOAD DATA INFILE '/tmp/test.txt'
    -> INTO TABLE test LINES STARTING BY "xxx";
```

With this you can read in a file that contains something like:

```
xxx"Row",1
something xxx"Row",2
```

And just get the data `("row",1)` and `("row",2)`.

The `IGNORE number LINES` option can be used to ignore lines at the start of the file. For example, you can use `IGNORE 1 LINES` to skip over an initial header line containing column names:

```
mysql> LOAD DATA INFILE '/tmp/test.txt'
    -> INTO TABLE test IGNORE 1 LINES;
```

When you use `SELECT ... INTO OUTFILE` in tandem with `LOAD DATA INFILE` to write data from a database into a file and then read the file back into the database later, the field- and line-handling options for both statements must match. Otherwise, `LOAD DATA INFILE` does not interpret the contents of the file properly. Suppose that you use `SELECT ... INTO OUTFILE` to write a file with fields delimited by commas:

```
mysql> SELECT * INTO OUTFILE 'data.txt'
    ->          FIELDS TERMINATED BY ','
    ->          FROM table2;
```

To read the comma-delimited file back in, the correct statement would be:

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE table2
    ->           FIELDS TERMINATED BY ',';
```

If instead you tried to read in the file with the statement shown here, it wouldn't work because it instructs `LOAD DATA INFILE` to look for tabs between fields:

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE table2
    ->           FIELDS TERMINATED BY '\t';
```

The likely result is that each input line would be interpreted as a single field.

`LOAD DATA INFILE` can be used to read files obtained from external sources, too. For example, a file in dBASE format has fields separated by commas and enclosed within double quotes. If lines in the file are terminated by newlines, the statement shown here illustrates the field- and line-handling options you would use to load the file:

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE tbl_name
    ->           FIELDS TERMINATED BY ',' ENCLOSED BY '"'
    ->           LINES TERMINATED BY '\n';
```

Any of the field- or line-handling options can specify an empty string (`''`). If not empty, the `FIELDS [OPTIONALLY] ENCLOSED BY` and `FIELDS ESCAPED BY` values must be a single character. The `FIELDS TERMINATED BY`, `LINES STARTING BY`, and `LINES TERMINATED BY` values can be more than one character. For example, to write lines that are terminated by carriage return/line-

feed pairs, or to read a file containing such lines, specify a `LINES TERMINATED BY '\r\n'` clause.

To read a file containing jokes that are separated by lines consisting of %%, you can do this

```
mysql> CREATE TABLE jokes
    ->     (a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->     joke TEXT NOT NULL);
mysql> LOAD DATA INFILE '/tmp/jokes.txt' INTO TABLE jokes
    ->     FIELDS TERMINATED BY ''
    ->     LINES TERMINATED BY '\n%%\n' (joke);
```

`FIELDS [OPTIONALLY] ENCLOSED BY` controls quoting of fields. For output (`SELECT ... INTO OUTFILE`), if you omit the word `OPTIONALLY`, all fields are enclosed by the `ENCLOSED BY` character. An example of such output (using a comma as the field delimiter) is shown here:

```
"1","a string","100.20"
"2","a string containing a , comma","102.20"
"3","a string containing a \" quote","102.20"
"4","a string containing a \", quote and comma","102.20"
```

If you specify `OPTIONALLY`, the `ENCLOSED BY` character is used only to enclose `CHAR` and `VARCHAR` fields:

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a \" quote",102.20
4,"a string containing a \", quote and comma",102.20
```

Note that occurrences of the `ENCLOSED BY` character within a field value are escaped by prefixing them with the `ESCAPED BY` character. Also note that if you specify an empty `ESCAPED BY` value, it is possible to generate output that cannot be read properly by `LOAD DATA INFILE`. For example, the preceding output just shown would appear as follows if the escape character is empty. Observe that the second field in the fourth line contains a comma following the quote, which (erroneously) appears to terminate the field:

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a " quote",102.20
4,"a string containing a ", quote and comma",102.20
```

For input, the `ENCLOSED BY` character, if present, is stripped from the ends of field values. (This is true whether or not `OPTIONALLY` is specified; `OPTIONALLY` has no effect on input interpretation.) Occurrences of the `ENCLOSED BY` character preceded by the `ESCAPED BY` character are interpreted as part of the current field value.

If the field begins with the `ENCLOSED BY` character, instances of that character are recognized as terminating a field value only if followed by the field or line `TERMINATED BY` sequence. To avoid ambiguity, occurrences of the `ENCLOSED BY` character within a field value can be doubled and are interpreted as a single instance of the character. For example, if `ENCLOSED BY '"'` is specified, quotes are handled as shown here:

```
"The ""BIG"" boss"  -> The "BIG" boss
The "BIG" boss      -> The "BIG" boss
The ""BIG"" boss    -> The ""BIG"" boss
```

`FIELDS ESCAPED BY` controls how to write or read special characters. If the `FIELDS ESCAPED`

BY character is not empty, it is used to prefix the following characters on output:

- The FIELDS ESCAPED BY character
- The FIELDS [OPTIONALLY] ENCLOSED BY character
- The first character of the FIELDS TERMINATED BY and LINES TERMINATED BY values
- ASCII 0 (what is actually written following the escape character is ASCII '0', not a zero-valued byte)

If the FIELDS ESCAPED BY character is empty, no characters are escaped and NULL is output as NULL, not \N. It is probably not a good idea to specify an empty escape character, particularly if field values in your data contain any of the characters in the list just given.

For input, if the FIELDS ESCAPED BY character is not empty, occurrences of that character are stripped and the following character is taken literally as part of a field value. The exceptions are an escaped '0' or 'N' (for example, \0 or \N if the escape character is '\'). These sequences are interpreted as ASCII NUL (a zero-valued byte) and NULL. The rules for NULL handling are described later in this section.

For more information about '\'-escape syntax, see Section 9.1, “Literal Values”.

In certain cases, field- and line-handling options interact:

- If LINES TERMINATED BY is an empty string and FIELDS TERMINATED BY is non-empty, lines are also terminated with FIELDS TERMINATED BY.
- If the FIELDS TERMINATED BY and FIELDS ENCLOSED BY values are both empty (''), a fixed-row (non-delimited) format is used. With fixed-row format, no delimiters are used between fields (but you can still have a line terminator). Instead, column values are written and read using the ``display" widths of the columns. For example, if a column is declared as INT(7), values for the column are written using seven-character fields. On input, values for the column are obtained by reading seven characters.

  LINES TERMINATED BY is still used to separate lines. If a line doesn't contain all fields, the rest of the columns are set to their default values. If you don't have a line terminator, you should set this to ''. In this case, the text file must contain all fields for each row.

  Fixed-row format also affects handling of NULL values, as described later. Note that fixed-size format does not work if you are using a multi-byte character set.

Handling of NULL values varies according to the FIELDS and LINES options in use:

- For the default FIELDS and LINES values, NULL is written as a field value of \N for output, and a field value of \N is read as NULL for input (assuming that the ESCAPED BY character is '\').
- If FIELDS ENCLOSED BY is not empty, a field containing the literal word NULL as its value is read as a NULL value. This differs from the word NULL enclosed within FIELDS ENCLOSED BY characters, which is read as the string 'NULL'.
- If FIELDS ESCAPED BY is empty, NULL is written as the word NULL.
- With fixed-row format (which happens when FIELDS TERMINATED BY and FIELDS ENCLOSED BY are both empty), NULL is written as an empty string. Note that this causes both NULL

values and empty strings in the table to be indistinguishable when written to the file because they are both written as empty strings. If you need to be able to tell the two apart when reading the file back in, you should not use fixed-row format.

Some cases are not supported by `LOAD DATA INFILE`:

- Fixed-size rows (`FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` both empty) and `BLOB` or `TEXT` columns.
- If you specify one separator that is the same as or a prefix of another, `LOAD DATA INFILE` won't be able to interpret the input properly. For example, the following `FIELDS` clause would cause problems:

  ```
  FIELDS TERMINATED BY '"' ENCLOSED BY '"'
  ```

- If `FIELDS ESCAPED BY` is empty, a field value that contains an occurrence of `FIELDS ENCLOSED BY` or `LINES TERMINATED BY` followed by the `FIELDS TERMINATED BY` value causes `LOAD DATA INFILE` to stop reading a field or line too early. This happens because `LOAD DATA INFILE` cannot properly determine where the field or line value ends.

The following example loads all columns of the `persondata` table:

```
mysql> LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata;
```

By default, when no column list is provided at the end of the `LOAD DATA INFILE` statement, input lines are expected to contain a field for each table column. If you want to load only some of a table's columns, specify a column list:

```
mysql> LOAD DATA INFILE 'persondata.txt'
    ->           INTO TABLE persondata (col1,col2,...);
```

You must also specify a column list if the order of the fields in the input file differs from the order of the columns in the table. Otherwise, MySQL cannot tell how to match up input fields with table columns.

If an input line has too many fields, the extra fields are ignored and the number of warnings is incremented.

If an input line has too few fields, the table columns for which input fields are missing are set to their default values. Default value assignment is described in Section 13.2.6, “`CREATE TABLE` Syntax”.

An empty field value is interpreted differently than if the field value is missing:

- For string types, the column is set to the empty string.
- For numeric types, the column is set to `0`.
- For date and time types, the column is set to the appropriate ``zero'' value for the type. See Section 11.3, “Date and Time Types”.

These are the same values that result if you assign an empty string explicitly to a string, numeric, or date or time type explicitly in an `INSERT` or `UPDATE` statement.

TIMESTAMP columns are set to the current date and time only if there is a NULL value for the column (that is, \N), or (for the first TIMESTAMP column only) if the TIMESTAMP column is omitted from the field list when a field list is specified.

LOAD DATA INFILE regards all input as strings, so you can't use numeric values for ENUM or SET columns the way you can with INSERT statements. All ENUM and SET values must be specified as strings!

When the LOAD DATA INFILE statement finishes, it returns an information string in the following format:

```
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0
```

If you are using the C API, you can get information about the statement by calling the mysql_info() function. See Section 22.2.3.31, "mysql_info()".

Warnings occur under the same circumstances as when values are inserted via the INSERT statement (see Section 13.1.4, "INSERT Syntax"), except that LOAD DATA INFILE also generates warnings when there are too few or too many fields in the input row. The warnings are not stored anywhere; the number of warnings can be used only as an indication of whether everything went well.

From MySQL 4.1.1 on, you can use SHOW WARNINGS to get a list of the first max_error_count warnings as information about what went wrong. See Section 13.5.4.20, "SHOW WARNINGS Syntax".

Before MySQL 4.1.1, only a warning count is available to indicate that something went wrong. If you get warnings and want to know exactly why you got them, one way to do this is to dump the table into another file using SELECT ... INTO OUTFILE and compare the file to your original input file.

13.1.6. REPLACE Syntax

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name [(col_name,...)]
    VALUES ({expr | DEFAULT},...),(...),...
```

Or:

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    SET col_name={expr | DEFAULT}, ...
```

Or:

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name [(col_name,...)]
    SELECT ...
```

REPLACE works exactly like INSERT, except that if an old record in the table has the same value as a new record for a PRIMARY KEY or a UNIQUE index, the old record is deleted before the new record is inserted. See Section 13.1.4, "INSERT Syntax".

Note that unless the table has a PRIMARY KEY or UNIQUE index, using a REPLACE statement makes no sense. It becomes equivalent to INSERT, because there is no index to be used to determine whether a new row duplicates another.

Values for all columns are taken from the values specified in the REPLACE statement. Any missing columns are set to their default values, just as happens for INSERT. You can't refer to values from the

current row and use them in the new row. If you use an assignment such as `SET col_name = col_name + 1`, the reference to the column name on the right hand side is treated as `DEFAULT(col_name)`, so the assignment is equivalent to `SET col_name = DEFAULT(col_name) + 1`.

To be able to use `REPLACE`, you must have `INSERT` and `DELETE` privileges for the table.

The `REPLACE` statement returns a count to indicate the number of rows affected. This is the sum of the rows deleted and inserted. If the count is 1 for a single-row `REPLACE`, a row was inserted and no rows were deleted. If the count is greater than 1, one or more old rows were deleted before the new row was inserted. It is possible for a single row to replace more than one old row if the table contains multiple unique indexes and the new row duplicates values for different old rows in different unique indexes.

The affected-rows count makes it easy to determine whether `REPLACE` only added a row or whether it also replaced any rows: Check whether the count is 1 (added) or greater (replaced).

If you are using the C API, the affected-rows count can be obtained using the `mysql_affected_rows()` function.

Currently, you cannot replace into a table and select from the same table in a subquery.

Here follows in more detail the algorithm that is used (it is also used with `LOAD DATA ... REPLACE`):

1. Try to insert the new row into the table
2. While the insertion fails because a duplicate-key error occurs for a primary or unique key:
   a. Delete from the table the conflicting row that has the duplicate key value
   b. Try again to insert the new row into the table

13.1.7. SELECT Syntax

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr, ...
    [INTO OUTFILE 'file_name' export_options
      | INTO DUMPFILE 'file_name']
    [FROM table_references
      [WHERE where_definition]
      [GROUP BY {col_name | expr | position}
        [ASC | DESC], ... [WITH ROLLUP]]
      [HAVING where_definition]
      [ORDER BY {col_name | expr | position}
        [ASC | DESC] , ...]
      [LIMIT {[offset,] row_count | row_count OFFSET offset}]
      [PROCEDURE procedure_name(argument_list)]
      [FOR UPDATE | LOCK IN SHARE MODE]]
```

`SELECT` is used to retrieve rows selected from one or more tables. Support for `UNION` statements and subqueries is available as of MySQL 4.0 and 4.1, respectively. See Section 13.1.7.2, “`UNION` Syntax” and Section 13.1.8, “Subquery Syntax”.

- Each *select_expr* indicates a column you want to retrieve.
- *table_references* indicates the table or tables from which to retrieve rows. Its syntax is described in Section 13.1.7.1, "JOIN Syntax".
- *where_definition* consists of the keyword WHERE followed by an expression that indicates the condition or conditions that rows must satisfy to be selected.

SELECT can also be used to retrieve rows computed without reference to any table.

For example:

```
mysql> SELECT 1 + 1;
        -> 2
```

All clauses used must be given in exactly the order shown in the syntax description. For example, a HAVING clause must come after any GROUP BY clause and before any ORDER BY clause.

- A *select_expr* can be given an alias using AS alias_name. The alias is used as the expression's column name and can be used in GROUP BY, ORDER BY, or HAVING clauses. For example:

  ```
  mysql> SELECT CONCAT(last_name,', ',first_name) AS full_name
      -> FROM mytable ORDER BY full_name;
  ```

  The AS keyword is optional when aliasing a *select_expr*. The preceding example could have been written like this:

  ```
  mysql> SELECT CONCAT(last_name,', ',first_name) full_name
      -> FROM mytable ORDER BY full_name;
  ```

  Because the AS is optional, a subtle problem can occur if you forget the comma between two *select_expr* expressions: MySQL interprets the second as an alias name. For example, in the following statement, columnb is treated as an alias name:

  ```
  mysql> SELECT columna columnb FROM mytable;
  ```

- It is not allowable to use a column alias in a WHERE clause, because the column value might not yet be determined when the WHERE clause is executed. See Section A.5.4, "Problems with Column Aliases".
- The FROM table_references clause indicates the tables from which to retrieve rows. If you name more than one table, you are performing a join. For information on join syntax, see Section 13.1.7.1, "JOIN Syntax". For each table specified, you can optionally specify an alias.

  ```
  tbl_name [[AS] alias]
      [[USE INDEX (key_list)]
        | [IGNORE INDEX (key_list)]
        | [FORCE INDEX (key_list)]]
  ```

  The use of USE INDEX, IGNORE INDEX, FORCE INDEX to give the optimizer hints about how to choose indexes is described in Section 13.1.7.1, "JOIN Syntax".

  In MySQL 4.0.14, you can use SET max_seeks_for_key=*value* as an alternative way to

force MySQL to prefer key scans instead of table scans.

- You can refer to a table within the current database as *tbl_name* (within the current database), or as *db_name.tbl_name* to explicitly specify a database. You can refer to a column as *col_name*, *tbl_name.col_name*, or *db_name.tbl_name.col_name*. You need not specify a *tbl_name* or *db_name.tbl_name* prefix for a column reference unless the reference would be ambiguous. See Section 9.2, "Database, Table, Index, Column, and Alias Names" for examples of ambiguity that require the more explicit column reference forms.

- From MySQL 4.1.0 on, you are allowed to specify DUAL as a dummy table name in situations where no tables are referenced:

  ```
  mysql> SELECT 1 + 1 FROM DUAL;
          -> 2
  ```

  DUAL is purely a compatibility feature. Some other servers require this syntax.

- A table reference can be aliased using *tbl_name* AS *alias_name* or *tbl_name alias_name*:

  ```
  mysql> SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
      ->     WHERE t1.name = t2.name;
  mysql> SELECT t1.name, t2.salary FROM employee t1, info t2
      ->     WHERE t1.name = t2.name;
  ```

- In the WHERE clause, you can use any of the functions that MySQL supports, except for aggregate (summary) functions. See Chapter 12, *Functions and Operators*.

- Columns selected for output can be referred to in ORDER BY and GROUP BY clauses using column names, column aliases, or column positions. Column positions are integers and begin with 1:

  ```
  mysql> SELECT college, region, seed FROM tournament
      ->     ORDER BY region, seed;
  mysql> SELECT college, region AS r, seed AS s FROM tournament
      ->     ORDER BY r, s;
  mysql> SELECT college, region, seed FROM tournament
      ->     ORDER BY 2, 3;
  ```

  To sort in reverse order, add the DESC (descending) keyword to the name of the column in the ORDER BY clause that you are sorting by. The default is ascending order; this can be specified explicitly using the ASC keyword.

  Use of column positions is deprecated because the syntax has been removed from the SQL standard.

- If you use GROUP BY, output rows are sorted according to the GROUP BY columns as if you had an ORDER BY for the same columns. MySQL has extended the GROUP BY clause as of version 3.23.34 so that you can also specify ASC and DESC after columns named in the clause:

  ```
  SELECT a, COUNT(b) FROM test_table GROUP BY a DESC
  ```

- MySQL extends the use of GROUP BY to allow you to select fields that are not mentioned in the GROUP BY clause. If you are not getting the results you expect from your query, please read the GROUP BY description. See Section 12.9, "Functions and Modifiers for Use with GROUP BY Clauses".

- As of MySQL 4.1.1, GROUP BY allows a WITH ROLLUP modifier. See Section 12.9.2, "GROUP

BY Modifiers".

- The HAVING clause is applied nearly last, just before items are sent to the client, with no optimization. (LIMIT is applied after HAVING.)

  Before MySQL 5.0.2, a HAVING clause can refer to any column or alias named in a *select_expr* in the SELECT list or in outer subqueries, and to aggregate functions. Standard SQL requires that HAVING must reference only columns in the GROUP BY clause or columns used in aggregate functions. To accommodate both standard SQL and the MySQL-specific behavior of being able to refer columns in the SELECT list, MySQL 5.0.2 and up allows HAVING to refer to columns in the SELECT list, columns in the GROUP BY clause, columns in outer subqueries, and to aggregate functions.

  For example, the following statement works in MySQL 5.0.2 but produces an error for earlier versions:

  ```
  mysql> SELECT COUNT(*) FROM t GROUP BY col1 HAVING col1 = 2;
  ```

  If the HAVING clause refers to a column that is ambiguous, a warning occurs. In the following statement, col2 is ambiguous because it is used both as an alias and as a column name:

  ```
  mysql> SELECT COUNT(col1) AS col2 FROM t GROUP BY col2 HAVING col2 = 2;
  ```

  Preference is given to standard SQL behavior, so that if a HAVING column name is used both in GROUP BY and as an aliased column in the output column list, preferences is given to the column in the GROUP BY column.

- Don't use HAVING for items that should be in the WHERE clause. For example, do not write this:

  ```
  mysql> SELECT col_name FROM tbl_name HAVING col_name > 0;
  ```

  Write this instead:

  ```
  mysql> SELECT col_name FROM tbl_name WHERE col_name > 0;
  ```

- The HAVING clause can refer to aggregate functions, which the WHERE clause cannot:

  ```
  mysql> SELECT user, MAX(salary) FROM users
      ->     GROUP BY user HAVING MAX(salary)>10;
  ```

  However, that does not work in older MySQL servers (before version 3.22.5). Instead, you can use a column alias in the select list and refer to the alias in the HAVING clause:

  ```
  mysql> SELECT user, MAX(salary) AS max_salary FROM users
      ->     GROUP BY user HAVING max_salary>10;
  ```

- The LIMIT clause can be used to constrain the number of rows returned by the SELECT statement. LIMIT takes one or two numeric arguments, which must be integer constants.

  With two arguments, the first argument specifies the offset of the first row to return, and the second specifies the maximum number of rows to return. The offset of the initial row is 0 (not 1):

  ```
  mysql> SELECT * FROM table LIMIT 5,10;  # Retrieve rows 6-15
  ```

For compatibility with PostgreSQL, MySQL also supports the `LIMIT row_count OFFSET offset` syntax.

To retrieve all rows from a certain offset up to the end of the result set, you can use some large number for the second parameter. This statement retrieves all rows from the 96th row to the last:

```
mysql> SELECT * FROM table LIMIT 95,18446744073709551615;
```

With one argument, the value specifies the number of rows to return from the beginning of the result set:

```
mysql> SELECT * FROM table LIMIT 5;     # Retrieve first 5 rows
```

In other words, `LIMIT n` is equivalent to `LIMIT 0,n`.

- The `SELECT ... INTO OUTFILE 'file_name'` form of `SELECT` writes the selected rows to a file. The file is created on the server host, so you must have the `FILE` privilege to use this syntax. The file cannot currently exist, which among other things prevents files such as `/etc/passwd` and database tables from being destroyed.

  The `SELECT ... INTO OUTFILE` statement is intended primarily to let you very quickly dump a table on the server machine. If you want to create the resulting file on some client host other than the server host, you can't use `SELECT ... INTO OUTFILE`. In that case, you should instead use some command like `mysql -e "SELECT ..." > file_name` on the client host to generate the file.

  `SELECT ... INTO OUTFILE` is the complement of `LOAD DATA INFILE`; the syntax for the `export_options` part of the statement consists of the same `FIELDS` and `LINES` clauses that are used with the `LOAD DATA INFILE` statement. See Section 13.1.5, “`LOAD DATA INFILE` Syntax”.

  `FIELDS ESCAPED BY` controls how to write special characters. If the `FIELDS ESCAPED BY` character is not empty, it is used to prefix the following characters on output:

  - The `FIELDS ESCAPED BY` character
  - The `FIELDS [OPTIONALLY] ENCLOSED BY` character
  - The first character of the `FIELDS TERMINATED BY` and `LINES TERMINATED BY` values
  - ASCII `0` (what is actually written following the escape character is ASCII '`0`', not a zero-valued byte)

  If the `FIELDS ESCAPED BY` character is empty, no characters are escaped and `NULL` is output as `NULL`, not `\N`. It is probably not a good idea to specify an empty escape character, particularly if field values in your data contain any of the characters in the list just given.

  The reason for the above is that you *must* escape any `FIELDS TERMINATED BY`, `ENCLOSED BY`, `ESCAPED BY`, or `LINES TERMINATED BY` characters to reliably be able to read the file back. ASCII NUL is escaped to make it easier to view with some pagers.

  The resulting file doesn't have to conform to SQL syntax, so nothing else need be escaped.

  Here is an example that produces a file in the comma-separated values format used by many programs:

  ```
  SELECT a,b,a+b INTO OUTFILE '/tmp/result.text'
  ```

```
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
FROM test_table;
```

- If you use `INTO DUMPFILE` instead of `INTO OUTFILE`, MySQL writes only one row into the file, without any column or line termination and without performing any escape processing. This is useful if you want to store a `BLOB` value in a file.
- **Note**: Any file created by `INTO OUTFILE` or `INTO DUMPFILE` is writable by all users on the server host. The reason for this is that the MySQL server can't create a file that is owned by anyone other than the user it's running as (you should never run **mysqld** as `root`). The file thus must be world-writable so that you can manipulate its contents.
- A `PROCEDURE` clause names a procedure that should process the data in the result set. For an example, see Section 25.3.1, "Procedure Analyse".
- If you use `FOR UPDATE` on a storage engine that uses page or row locks, rows examined by the query are write-locked until the end of the current transaction. Using `LOCK IN SHARE MODE` sets a shared lock that prevents other transactions from updating or deleting the examined rows. See Section 15.11.4, "Locking Reads `SELECT ... FOR UPDATE` and `SELECT ... LOCK IN SHARE MODE`".

Following the `SELECT` keyword, you can give a number of options that affect the operation of the statement.

The `ALL`, `DISTINCT`, and `DISTINCTROW` options specify whether duplicate rows should be returned. If none of these options are given, the default is `ALL` (all matching rows are returned). `DISTINCT` and `DISTINCTROW` are synonyms and specify that duplicate rows in the result set should be removed.

`HIGH_PRIORITY`, `STRAIGHT_JOIN`, and options beginning with `SQL_` are MySQL extensions to standard SQL.

- `HIGH_PRIORITY` gives the `SELECT` higher priority than a statement that updates a table. You should use this only for queries that are very fast and must be done at once. A `SELECT HIGH_PRIORITY` query that is issued while the table is locked for reading runs even if there is an update statement waiting for the table to be free.

  `HIGH_PRIORITY` cannot be used with `SELECT` statements that are part of a `UNION`.
- `STRAIGHT_JOIN` forces the optimizer to join the tables in the order in which they are listed in the `FROM` clause. You can use this to speed up a query if the optimizer joins the tables in non-optimal order. See Section 7.2.1, "`EXPLAIN` Syntax (Get Information About a `SELECT`)". `STRAIGHT_JOIN` also can be used in the *`table_references`* list. See Section 13.1.7.1, "`JOIN` Syntax".
- `SQL_BIG_RESULT` can be used with `GROUP BY` or `DISTINCT` to tell the optimizer that the result set has many rows. In this case, MySQL directly uses disk-based temporary tables if needed. MySQL also, in this case, prefers sorting to using a temporary table with a key on the `GROUP BY` elements.
- `SQL_BUFFER_RESULT` forces the result to be put into a temporary table. This helps MySQL free the table locks early and helps in cases where it takes a long time to send the result set to the client.
- `SQL_SMALL_RESULT` can be used with `GROUP BY` or `DISTINCT` to tell the optimizer that the result set is small. In this case, MySQL uses fast temporary tables to store the resulting table instead

of using sorting. In MySQL 3.23 and up, this shouldn't normally be needed.

- SQL_CALC_FOUND_ROWS (available in MySQL 4.0.0 and up) tells MySQL to calculate how many rows there would be in the result set, disregarding any LIMIT clause. The number of rows can then be retrieved with SELECT FOUND_ROWS(). See Section 12.8.3, "Information Functions".

  Before MySQL 4.1.0, this option does not work with LIMIT 0, which is optimized to return instantly (resulting in a row count of 0). See Section 7.2.12, "How MySQL Optimizes LIMIT".

- SQL_CACHE tells MySQL to store the query result in the query cache if you are using a query_cache_type value of 2 or DEMAND. For a query that uses UNION or subqueries, this option takes effect to be used in any SELECT of the query. See Section 5.11, "The MySQL Query Cache".

- SQL_NO_CACHE tells MySQL not to store the query result in the query cache. See Section 5.11, "The MySQL Query Cache". For a query that uses UNION or subqueries, this option takes effect to be used in any SELECT of the query.

13.1.7.1. JOIN Syntax

MySQL supports the following JOIN syntaxes for the *table_references* part of SELECT statements and multiple-table DELETE and UPDATE statements:

```
table_reference, table_reference
table_reference [INNER | CROSS] JOIN table_reference [join_condition]
table_reference STRAIGHT_JOIN table_reference
table_reference LEFT [OUTER] JOIN table_reference join_condition
table_reference NATURAL [LEFT [OUTER]] JOIN table_reference
{ OJ table_reference LEFT OUTER JOIN table_reference
    ON conditional_expr }
table_reference RIGHT [OUTER] JOIN table_reference join_condition
table_reference NATURAL [RIGHT [OUTER]] JOIN table_reference
```

table_reference is defined as:

```
tbl_name [[AS] alias]
    [[USE INDEX (key_list)]
      | [IGNORE INDEX (key_list)]
      | [FORCE INDEX (key_list)]]
```

join_condition is defined as:

```
ON conditional_expr | USING (column_list)
```

You should generally not have any conditions in the ON part that are used to restrict which rows you want in the result set, but rather specify these conditions in the WHERE clause. There are exceptions to this rule.

Note that INNER JOIN syntax allows a join_condition only from MySQL 3.23.17 on. The same is true for JOIN and CROSS JOIN only as of MySQL 4.0.11.

The { OJ ... LEFT OUTER JOIN ...} syntax shown in the preceding list exists only for compatibility with ODBC.

- A table reference can be aliased using *tbl_name* AS *alias_name* or *tbl_name ali-*

as_name:

```
mysql> SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
    ->        WHERE t1.name = t2.name;
mysql> SELECT t1.name, t2.salary FROM employee t1, info t2
    ->        WHERE t1.name = t2.name;
```

- The `ON` conditional is any conditional expression of the form that can be used in a `WHERE` clause.
- If there is no matching record for the right table in the `ON` or `USING` part in a `LEFT JOIN`, a row with all columns set to `NULL` is used for the right table. You can use this fact to find records in a table that have no counterpart in another table:

  ```
  mysql> SELECT table1.* FROM table1
      ->        LEFT JOIN table2 ON table1.id=table2.id
      ->        WHERE table2.id IS NULL;
  ```

  This example finds all rows in `table1` with an `id` value that is not present in `table2` (that is, all rows in `table1` with no corresponding row in `table2`). This assumes that `table2.id` is declared `NOT NULL`. See Section 7.2.9, “How MySQL Optimizes `LEFT JOIN` and `RIGHT JOIN`”.
- The `USING (column_list)` clause names a list of columns that must exist in both tables. The following two clauses are semantically identical:

  ```
  a LEFT JOIN b USING (c1,c2,c3)
  a LEFT JOIN b ON a.c1=b.c1 AND a.c2=b.c2 AND a.c3=b.c3
  ```
- The `NATURAL [LEFT] JOIN` of two tables is defined to be semantically equivalent to an `INNER JOIN` or a `LEFT JOIN` with a `USING` clause that names all columns that exist in both tables.
- `INNER JOIN` and `,` (comma) are semantically equivalent in the absence of a join condition: both produce a Cartesian product between the specified tables (that is, each and every row in the first table are joined to all rows in the second table).
- `RIGHT JOIN` works analogously to `LEFT JOIN`. To keep code portable across databases, it's recommended to use `LEFT JOIN` instead of `RIGHT JOIN`.
- `STRAIGHT_JOIN` is identical to `JOIN`, except that the left table is always read before the right table. This can be used for those (few) cases for which the join optimizer puts the tables in the wrong order.

As of MySQL 3.23.12, you can give hints about which index MySQL should use when retrieving information from a table. By specifying `USE INDEX (key_list)`, you can tell MySQL to use only one of the possible indexes to find rows in the table. The alternative syntax `IGNORE INDEX (key_list)` can be used to tell MySQL to not use some particular index. These hints are useful if `EXPLAIN` shows that MySQL is using the wrong index from the list of possible indexes.

From MySQL 4.0.9 on, you can also use `FORCE INDEX`. This acts likes `USE INDEX (key_list)` but with the addition that a table scan is assumed to be *very* expensive. In other words, a table scan is only used if there is no way to use one of the given indexes to find rows in the table.

`USE KEY`, `IGNORE KEY`, and `FORCE KEY` are synonyms for `USE INDEX`, `IGNORE INDEX`, and `FORCE INDEX`.

Note: USE INDEX, IGNORE INDEX, and FORCE INDEX only affect which indexes are used when MySQL decides how to find rows in the table and how to do the join. They do not affect whether an index is used when resolving an ORDER BY or GROUP BY.

Some join examples:

```
mysql> SELECT * FROM table1,table2 WHERE table1.id=table2.id;
mysql> SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id;
mysql> SELECT * FROM table1 LEFT JOIN table2 USING (id);
mysql> SELECT * FROM table1 LEFT JOIN table2 ON table1.id=table2.id
    ->          LEFT JOIN table3 ON table2.id=table3.id;
mysql> SELECT * FROM table1 USE INDEX (key1,key2)
    ->          WHERE key1=1 AND key2=2 AND key3=3;
mysql> SELECT * FROM table1 IGNORE INDEX (key3)
    ->          WHERE key1=1 AND key2=2 AND key3=3;
```

See Section 7.2.9, “How MySQL Optimizes LEFT JOIN and RIGHT JOIN”.

13.1.7.2. UNION Syntax

```
SELECT ...
UNION [ALL | DISTINCT]
SELECT ...
  [UNION [ALL | DISTINCT]
   SELECT ...]
```

UNION is used to combine the result from many SELECT statements into one result set. UNION is available from MySQL 4.0.0 on.

Selected columns listed in corresponding positions of each SELECT statement should have the same type. (For example, the first column selected by the first statement should have the same type as the first column selected by the other statements.) The column names used in the first SELECT statement are used as the column names for the results returned.

The SELECT statements are normal select statements, but with the following restrictions:

- Only the last SELECT statement can have INTO OUTFILE.
- HIGH_PRIORITY cannot be used with SELECT statements that are part of a UNION. If you specify it for the first SELECT, it has no effect. If you specify it for any subsequent SELECT statements, a syntax error results.

If you don't use the keyword ALL for the UNION, all returned rows are unique, as if you had done a DISTINCT for the total result set. If you specify ALL, you get all matching rows from all the used SELECT statements.

The DISTINCT keyword is an optional word (introduced in MySQL 4.0.17). It does nothing, but is allowed in the syntax as required by the SQL standard.

Before MySQL 4.1.2, you cannot mix UNION ALL and UNION DISTINCT in the same query. If you use ALL for one UNION, it is used for all of them. As of MySQL 4.1.2, mixed UNION types are treated such that a DISTINCT union overrides any ALL union to its left. A DISTINCT union can be produced explicitly by using UNION DISTINCT or implicitly by using UNION with no following DISTINCT or ALL keyword.

If you want to use an ORDER BY or LIMIT clause to sort or limit the entire UNION result, parenthesize

the individual SELECT statements and place the ORDER BY or LIMIT after the last one. The following example uses both clauses:

```
(SELECT a FROM tbl_name WHERE a=10 AND B=1)
UNION
(SELECT a FROM tbl_name WHERE a=11 AND B=2)
ORDER BY a LIMIT 10;
```

This kind of ORDER BY cannot use column references that include a table name (that is, names in *tbl_name.col_name* format). Instead, provide a column alias in the first SELECT statement and refer to the alias in the ORDER BY, or else refer to the column in the ORDER BY using its column position. (An alias is preferable because use of column positions is deprecated.)

To apply ORDER BY or LIMIT to an individual SELECT, place the clause inside the parentheses that enclose the SELECT:

```
(SELECT a FROM tbl_name WHERE a=10 AND B=1 ORDER BY a LIMIT 10)
UNION
(SELECT a FROM tbl_name WHERE a=11 AND B=2 ORDER BY a LIMIT 10);
```

The types and lengths of the columns in the result set of a UNION take into account the values retrieved by all the SELECT statements. Before MySQL 4.1.1, a limitation of UNION is that only the values from the first SELECT are used to determine result column types and lengths. This could result in value truncation if, for example, the first SELECT retrieves shorter values than the second SELECT:

```
mysql> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);
+---------------+
| REPEAT('a',1) |
+---------------+
| a             |
| b             |
+---------------+
```

That limitation has been removed as of MySQL 4.1.1:

```
mysql> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);
+---------------+
| REPEAT('a',1) |
+---------------+
| a             |
| bbbbbbbbbb    |
+---------------+
```

13.1.8. Subquery Syntax

A subquery is a SELECT statement inside another statement.

Starting with MySQL 4.1, all subquery forms and operations that the SQL standard requires are supported, as well as a few features that are MySQL-specific.

With MySQL versions prior to 4.1, it was necessary to work around or avoid the use of subqueries. In many cases, subqueries can successfully be rewritten using joins and other methods. See Section 13.1.8.11, “Rewriting Subqueries as Joins for Earlier MySQL Versions”.

Here is an example of a subquery:

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

In this example, `SELECT * FROM t1 ...` is the *outer query* (or *outer statement*), and `(SELECT column1 FROM t2)` is the *subquery*. We say that the subquery is *nested* in the outer query, and in fact it's possible to nest subqueries within other subqueries, to a great depth. A subquery must always appear within parentheses.

The main advantages of subqueries are:

- They allow queries that are *structured* so that it's possible to isolate each part of a statement.
- They provide alternative ways to perform operations that would otherwise require complex joins and unions.
- They are, in many people's opinion, readable. Indeed, it was the innovation of subqueries that gave people the original idea of calling the early SQL ``Structured Query Language."

Here is an example statement that shows the major points about subquery syntax as specified by the SQL standard and supported in MySQL:

```
DELETE FROM t1
WHERE s11 > ANY
 (SELECT COUNT(*) /* no hint */ FROM t2
 WHERE NOT EXISTS
  (SELECT * FROM t3
   WHERE ROW(5*t2.s1,77)=
    (SELECT 50,11*s1 FROM t4 UNION SELECT 50,77 FROM
     (SELECT * FROM t5) AS t5)));
```

A subquery can return a scalar (a single value), a single row, a single column, or a table (one or more rows of one or more columns). These are called scalar, column, row, and table subqueries. Subqueries that return a particular kind of result often can be used only in certain contexts, as described in the following sections.

There are few restrictions on the type of statements in which subqueries can be used:

- A subquery's outer statement can be any one of: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `SET`, or `DO`.
- A subquery can contain any of the keywords or clauses that an ordinary `SELECT` can contain: `DISTINCT`, `GROUP BY`, `ORDER BY`, `LIMIT`, joins, index hints, `UNION` constructs, comments, functions, and so on.

One restriction is that currently you cannot modify a table and select from the same table in a subquery. This applies to statements such as `DELETE`, `INSERT`, `REPLACE`, and `UPDATE`.

13.1.8.1. The Subquery as Scalar Operand

In its simplest form, a subquery is a scalar subquery that returns a single value. A scalar subquery is a simple operand, and you can use it wherever a single column value or literal is legal, and you can expect it to have those characteristics that all operands have: a data type, a length, an indication whether it can be `NULL`, and so on. For example:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5) NOT NULL);
INSERT INTO t1 VALUES(100, 'abcde');
```

```
SELECT (SELECT s2 FROM t1);
```

The subquery in this SELECT returns a single value ('abcde') that has a data type of CHAR, a length of 5, a character set and collation equal to the defaults in effect at CREATE TABLE time, and an indication that the value in the column can be NULL. In fact, almost all subqueries can be NULL. If the table used in the example were empty, the value of the subquery would be NULL.

When you see examples in the following sections that contain the rather spartan construct (SELECT column1 FROM t1), imagine that your own code contains much more diverse and complex constructions.

For example, suppose that we make two tables:

```
CREATE TABLE t1 (s1 INT);
INSERT INTO t1 VALUES (1);
CREATE TABLE t2 (s1 INT);
INSERT INTO t2 VALUES (2);
```

Then perform a SELECT:

```
SELECT (SELECT s1 FROM t2) FROM t1;
```

The result is 2 because there is a row in t2 containing a column s1 that has a value of 2.

A scalar subquery can be part of an expression. Don't forget the parentheses, even if the subquery is an operand that provides an argument for a function. For example:

```
SELECT UPPER((SELECT s1 FROM t1)) FROM t2;
```

13.1.8.2. Comparisons Using Subqueries

The most common use of a subquery is in the form:

non_subquery_operand comparison_operator (*subquery*)

Where *comparison_operator* is one of these operators:

```
=  >  <  >=  <=  <>
```

For example:

```
... 'a' = (SELECT column1 FROM t1)
```

At one time the only legal place for a subquery was on the right side of a comparison, and you might still find some old DBMSs that insist on this.

Here is an example of a common-form subquery comparison that you cannot do with a join. It finds all the values in table t1 that are equal to a maximum value in table t2:

```
SELECT column1 FROM t1
       WHERE column1 = (SELECT MAX(column2) FROM t2);
```

Here is another example, which again is impossible with a join because it involves aggregating for one

of the tables. It finds all rows in table `t1` containing a value that occurs twice in a given column:

```
SELECT * FROM t1 AS t
    WHERE 2 = (SELECT COUNT(*) FROM t1 WHERE t1.id = t.id);
```

For a comparison performed with one of these comparison operators, the subquery must return a scalar, with the exception that `=` can be used with row subqueries. See Section 13.1.8.5, “Row Subqueries”.

13.1.8.3. Subqueries with ANY, IN, and SOME

Syntax:

```
operand comparison_operator ANY (subquery)
operand IN (subquery)
operand comparison_operator SOME (subquery)
```

The `ANY` keyword, which must follow a comparison operator, means ``return `TRUE` if the comparison is `TRUE` for `ANY` of the values in the column that the subquery returns." For example:

```
SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);
```

Suppose that there is a row in table `t1` containing `(10)`. The expression is `TRUE` if table `t2` contains `(21,14,7)` because there is a value `7` in `t2` that is less than `10`. The expression is `FALSE` if table `t2` contains `(20,10)`, or if table `t2` is empty. The expression is `UNKNOWN` if table `t2` contains `(NULL,NULL,NULL)`.

The word `IN` is an alias for `= ANY`. Thus these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 IN    (SELECT s1 FROM t2);
```

However, `NOT IN` is not an alias for `<> ANY`, but for `<> ALL`. See Section 13.1.8.4, “Subqueries with `ALL`”.

The word `SOME` is an alias for `ANY`. Thus these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 <> ANY  (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);
```

Use of the word `SOME` is rare, but this example shows why it might be useful. To most people's ears, the English phrase ``a is not equal to any b" means ``there is no b which is equal to a," but that isn't what is meant by the SQL syntax. The syntax means ``there is some b to which a is not equal." Using `<> SOME` instead helps ensure that everyone understands the true meaning of the query.

13.1.8.4. Subqueries with ALL

Syntax:

```
operand comparison_operator ALL (subquery)
```

The word `ALL`, which must follow a comparison operator, means ``return `TRUE` if the comparison is `TRUE` for `ALL` of the values in the column that the subquery returns." For example:

```
SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);
```

Suppose that there is a row in table `t1` containing `(10)`. The expression is `TRUE` if table `t2` contains `(-5,0,+5)` because `10` is greater than all three values in `t2`. The expression is `FALSE` if table `t2` contains `(12,6,NULL,-100)` because there is a single value `12` in table `t2` that is greater than `10`. The expression is `UNKNOWN` if table `t2` contains `(0,NULL,1)`.

Finally, if table `t2` is empty, the result is `TRUE`. You might think the result should be `UNKNOWN`, but sorry, it's `TRUE`. So, rather oddly, the following statement is `TRUE` when table `t2` is empty:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT s1 FROM t2);
```

But this statement is `UNKNOWN` when table `t2` is empty:

```
SELECT * FROM t1 WHERE 1 > (SELECT s1 FROM t2);
```

In addition, the following statement is `UNKNOWN` when table `t2` is empty:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT MAX(s1) FROM t2);
```

In general, *tables with `NULL` values* and *empty tables* are *edge cases*. When writing subquery code, always consider whether you have taken those two possibilities into account.

`NOT IN` is an alias for `<> ALL`. Thus these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 <> ALL (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 NOT IN (SELECT s1 FROM t2);
```

13.1.8.5. Row Subqueries

The discussion to this point has been of scalar or column subqueries, that is, subqueries that return a single value or a column of values. A *row subquery* is a subquery variant that returns a single row and can thus return more than one column value. Here are two examples:

```
SELECT * FROM t1 WHERE (1,2) = (SELECT column1, column2 FROM t2);
SELECT * FROM t1 WHERE ROW(1,2) = (SELECT column1, column2 FROM t2);
```

The queries here are both `TRUE` if table `t2` has a row where `column1 = 1` and `column2 = 2`.

The expressions `(1,2)` and `ROW(1,2)` are sometimes called *row constructors*. The two are equivalent. They are legal in other contexts, too. For example, the following two statements are semantically equivalent (although currently only the second one can be optimized):

```
SELECT * FROM t1 WHERE (column1,column2) = (1,1);
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

The normal use of row constructors, though, is for comparisons with subqueries that return two or more columns. For example, the following query answers the request, ``find all rows in table `t1` that also exist in table `t2`'':

```
SELECT column1,column2,column3
       FROM t1
       WHERE (column1,column2,column3) IN
             (SELECT column1,column2,column3 FROM t2);
```

13.1.8.6. EXISTS and NOT EXISTS

If a subquery returns any rows at all, then `EXISTS subquery` is `TRUE`, and `NOT EXISTS subquery` is `FALSE`. For example:

```
SELECT column1 FROM t1 WHERE EXISTS (SELECT * FROM t2);
```

Traditionally, an `EXISTS` subquery starts with `SELECT *`, but it could begin with `SELECT 5` or `SELECT column1` or anything at all. MySQL ignores the `SELECT` list in such a subquery, so it doesn't matter.

For the preceding example, if `t2` contains any rows, even rows with nothing but `NULL` values, then the `EXISTS` condition is `TRUE`. This is actually an unlikely example, since almost always a `[NOT] EXISTS` subquery contains correlations. Here are some more realistic examples:

- What kind of store is present in one or more cities?

  ```
  SELECT DISTINCT store_type FROM Stores
    WHERE EXISTS (SELECT * FROM Cities_Stores
                  WHERE Cities_Stores.store_type = Stores.store_type);
  ```

- What kind of store is present in no cities?

  ```
  SELECT DISTINCT store_type FROM Stores
    WHERE NOT EXISTS (SELECT * FROM Cities_Stores
                      WHERE Cities_Stores.store_type = Stores.store_type);
  ```

- What kind of store is present in all cities?

  ```
  SELECT DISTINCT store_type FROM Stores S1
    WHERE NOT EXISTS (
      SELECT * FROM Cities WHERE NOT EXISTS (
        SELECT * FROM Cities_Stores
         WHERE Cities_Stores.city = Cities.city
         AND Cities_Stores.store_type = Stores.store_type));
  ```

The last example is a double-nested `NOT EXISTS` query. That is, it has a `NOT EXISTS` clause within a `NOT EXISTS` clause. Formally, it answers the question ``does a city exist with a store that is not in `Stores`?'' But it's easier to say that a nested `NOT EXISTS` answers the question ``is x TRUE for all y?''

13.1.8.7. Correlated Subqueries

A *correlated subquery* is a subquery that contains a reference to a table that also appears in the outer query. For example:

```
SELECT * FROM t1 WHERE column1 = ANY
       (SELECT column1 FROM t2 WHERE t2.column2 = t1.column2);
```

Notice that the subquery contains a reference to a column of `t1`, even though the subquery's `FROM` clause doesn't mention a table `t1`. So, MySQL looks outside the subquery, and finds `t1` in the outer query.

Suppose that table `t1` contains a row where `column1 = 5` and `column2 = 6`; meanwhile, table `t2` contains a row where `column1 = 5` and `column2 = 7`. The simple expression `... WHERE column1 = ANY (SELECT column1 FROM t2)` would be `TRUE`, but in this example, the `WHERE` clause within the subquery is `FALSE` (because `(5,6)` is not equal to `(5,7)`), so the subquery as a whole is `FALSE`.

Scoping rule: MySQL evaluates from inside to outside. For example:

```
SELECT column1 FROM t1 AS x
  WHERE x.column1 = (SELECT column1 FROM t2 AS x
    WHERE x.column1 = (SELECT column1 FROM t3
      WHERE x.column2 = t3.column1));
```

In this statement, `x.column2` must be a column in table `t2` because `SELECT column1 FROM t2 AS x ...` renames `t2`. It is not a column in table `t1` because `SELECT column1 FROM t1 ...` is an outer query that is *farther out*.

For subqueries in `HAVING` or `ORDER BY` clauses, MySQL also looks for column names in the outer select list.

For certain cases, a correlated subquery is optimized. For example:

```
val IN (SELECT key_val FROM tbl_name WHERE correlated_condition)
```

Otherwise, they are inefficient and likely to be slow. Rewriting the query as a join might improve performance.

Correlated subqueries cannot refer to the results of aggregate functions from the outer query.

13.1.8.8. Subqueries in the `FROM` clause

Subqueries are legal in a `SELECT` statement's `FROM` clause. The syntax that you'll actually see is:

```
SELECT ... FROM (subquery) AS name ...
```

The `AS name` clause is mandatory, because every table in a `FROM` clause must have a name. Any columns in the *`subquery`* select list must have unique names. You can find this syntax described elsewhere in this manual, where the term used is ``derived tables."

For illustration, assume that you have this table:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5), s3 FLOAT);
```

Here's how to use a subquery in the `FROM` clause, using the example table:

```
INSERT INTO t1 VALUES (1,'1',1.0);
INSERT INTO t1 VALUES (2,'2',2.0);
SELECT sb1,sb2,sb3
       FROM (SELECT s1 AS sb1, s2 AS sb2, s3*2 AS sb3 FROM t1) AS sb
       WHERE sb1 > 1;
```

Result: `2, '2', 4.0`.

Here's another example: Suppose that you want to know the average of a set of sums for a grouped table. This won't work:

```
SELECT AVG(SUM(column1)) FROM t1 GROUP BY column1;
```

But this query provides the desired information:

```
SELECT AVG(sum_column1)
       FROM (SELECT SUM(column1) AS sum_column1
             FROM t1 GROUP BY column1) AS t1;
```

Notice that the column name used within the subquery `(sum_column1)` is recognized in the outer query.

Subqueries in the `FROM` clause can return a scalar, column, row, or table. At the moment, subqueries in the `FROM` clause cannot be correlated subqueries.

Subqueries in the `FROM` clause are executed even for the `EXPLAIN` statement (that is, derived temporary tables are built). This occurs because upper level queries need information about all tables during optimization phase.

13.1.8.9. Subquery Errors

There are some new error returns that apply only to subqueries. This section groups them together because reviewing them helps remind you of some points.

- Unsupported subquery syntax:

  ```
  ERROR 1235 (ER_NOT_SUPPORTED_YET)
  SQLSTATE = 42000
  Message = "This version of MySQL doesn't yet support
  'LIMIT & IN/ALL/ANY/SOME subquery'"
  ```

  This means that statements of the following form do not work, although this happens only in some early versions, such as MySQL 4.1.1:

  ```
  SELECT * FROM t1 WHERE s1 IN (SELECT s2 FROM t2 ORDER BY s1 LIMIT 1)
  ```

- Incorrect number of columns from subquery:

  ```
  ERROR 1241 (ER_OPERAND_COL)
  SQLSTATE = 21000
  Message = "Operand should contain 1 column(s)"
  ```

  This error occurs in cases like this:

  ```
  SELECT (SELECT column1, column2 FROM t2) FROM t1;
  ```

  It's okay to use a subquery that returns multiple columns, if the purpose is comparison. See Section 13.1.8.5, “Row Subqueries”. But in other contexts, the subquery must be a scalar operand.

- Incorrect number of rows from subquery:

  ```
  ERROR 1242 (ER_SUBSELECT_NO_1_ROW)
  SQLSTATE = 21000
  Message = "Subquery returns more than 1 row"
  ```

This error occurs for statements such as the following one, but only when there is more than one row in `t2`:

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

That means this error might occur in code that had been working for years, because somebody happened to make a change that affected the number of rows that the subquery can return. Remember that if the object is to find any number of rows, not just one, then the correct statement would look like this:

```
SELECT * FROM t1 WHERE column1 = ANY (SELECT column1 FROM t2);
```

- Incorrectly used table in subquery:

  ```
  Error 1093 (ER_UPDATE_TABLE_USED)
  SQLSTATE = HY000
  Message = "You can't specify target table 'x'
  for update in FROM clause"
  ```

  This error occurs in cases like this:

  ```
  UPDATE t1 SET column2 = (SELECT MAX(column1) FROM t1);
  ```

  It's okay to use a subquery for assignment within an `UPDATE` statement, since subqueries are legal in `UPDATE` and `DELETE` statements as well as in `SELECT` statements. However, you cannot use the same table, in this case table `t1`, for both the subquery's `FROM` clause and the update target.

For transactional storage engines, a failure of a subquery causes the entire statement to fail. For non-transactional storage engines, data modifications made before the error was encountered are preserved.

13.1.8.10. Optimizing Subqueries

Development is ongoing, so no optimization tip is reliable for the long term. Some interesting tricks that you might want to play with are:

- Use subquery clauses that affect the number or order of the rows in the subquery. For example:

  ```
  SELECT * FROM t1 WHERE t1.column1 IN
    (SELECT column1 FROM t2 ORDER BY column1);
  SELECT * FROM t1 WHERE t1.column1 IN
    (SELECT DISTINCT column1 FROM t2);
  SELECT * FROM t1 WHERE EXISTS
    (SELECT * FROM t2 LIMIT 1);
  ```

- Replace a join with a subquery. For example, use this query:

  ```
  SELECT DISTINCT column1 FROM t1 WHERE t1.column1 IN (
    SELECT column1 FROM t2);
  ```

  Instead of this query:

  ```
  SELECT DISTINCT t1.column1 FROM t1, t2
  ```

```
  WHERE t1.column1 = t2.column1;
```

- Some subqueries can be transformed to joins for compatibility with older versions of MySQL before 4.1 that do not support subqueries. However, in some cases, converting a subquery to a join may also improve performance. See Section 13.1.8.11, “Rewriting Subqueries as Joins for Earlier MySQL Versions”.

- Move clauses from outside to inside the subquery. For example, use this query:

  ```
  SELECT * FROM t1
    WHERE s1 IN (SELECT s1 FROM t1 UNION ALL SELECT s1 FROM t2);
  ```

  Instead of this query:

  ```
  SELECT * FROM t1
    WHERE s1 IN (SELECT s1 FROM t1) OR s1 IN (SELECT s1 FROM t2);
  ```

  For another example, use this query:

  ```
  SELECT (SELECT column1 + 5 FROM t1) FROM t2;
  ```

  Instead of this query:

  ```
  SELECT (SELECT column1 FROM t1) + 5 FROM t2;
  ```

- Use a row subquery instead of a correlated subquery. For example, use this query:

  ```
  SELECT * FROM t1
    WHERE (column1,column2) IN (SELECT column1,column2 FROM t2);
  ```

  Instead of this query:

  ```
  SELECT * FROM t1
    WHERE EXISTS (SELECT * FROM t2 WHERE t2.column1=t1.column1
    AND t2.column2=t1.column2);
  ```

- Use `NOT (a = ANY (...))` rather than `a <> ALL (...)`.

- Use `x = ANY (table containing (1,2))` rather than `x=1 OR x=2`.

- Use `= ANY` rather than `EXISTS`.

- For uncorrelated subqueries tht always return one row, `IN` is always slower than `=`. For example, use this query:

  ```
  SELECT * FROM t1 WHERE t1.col_name
    = (SELECT a FROM t2 WHERE b = some_const);
  ```

  Instead of this query:

  ```
  SELECT * FROM t1 WHERE t1.col_name
    IN (SELECT a FROM t2 WHERE b = some_const);
  ```

These tricks might cause programs to go faster or slower. Using MySQL facilities like the `BENCHMARK()` function, you can get an idea about what helps in your own situation.

Some optimizations that MySQL itself makes are:

- MySQL executes non-correlated subqueries only once. Use `EXPLAIN` to make sure that a given subquery really is non-correlated.
- MySQL rewrites `IN`/`ALL`/`ANY`/`SOME` subqueries in an attempt to take advantage of the possibility that the select-list columns in the subquery are indexed.
- MySQL replaces subqueries of the following form with an index-lookup function, which `EXPLAIN` describes as a special join type:

  ```
  ... IN (SELECT indexed_column FROM single_table ...)
  ```

- MySQL enhances expressions of the following form with an expression involving `MIN()` or `MAX()`, unless `NULL` values or empty sets are involved:

  ```
  value {ALL|ANY|SOME} {> | < | >= | <=} (non-correlated subquery)
  ```

  For example, this `WHERE` clause:

  ```
  WHERE 5 > ALL (SELECT x FROM t)
  ```

  might be treated by the optimizer like this:

  ```
  WHERE 5 > (SELECT MAX(x) FROM t)
  ```

There is a chapter titled ``How MySQL Transforms Subqueries'' in the MySQL Internals Manual. You can obtain this document by downloading the MySQL source package and looking for a file named `internals.texi` in the `Docs` directory.

13.1.8.11. Rewriting Subqueries as Joins for Earlier MySQL Versions

Before MySQL 4.1, only nested queries of the form `INSERT ... SELECT ...` and `REPLACE ... SELECT ...` are supported. The `IN()` construct can be used in other contexts to test membership in a set of values.

It is often possible to rewrite a query without a subquery:

```
SELECT * FROM t1 WHERE id IN (SELECT id FROM t2);
```

This can be rewritten as:

```
SELECT DISTINCT t1.* FROM t1, t2 WHERE t1.id=t2.id;
```

The queries:

```
SELECT * FROM t1 WHERE id NOT IN (SELECT id FROM t2);
SELECT * FROM t1 WHERE NOT EXISTS (SELECT id FROM t2 WHERE t1.id=t2.id);
```

Can be rewritten as:

```
SELECT table1.* FROM table1 LEFT JOIN table2 ON table1.id=table2.id
                                     WHERE table2.id IS NULL;
```

A `LEFT [OUTER] JOIN` can be faster than an equivalent subquery because the server might be able to optimize it better---a fact that is not specific to MySQL Server alone. Prior to SQL-92, outer joins did not exist, so subqueries were the only way to do certain things in those bygone days. Today, MySQL Server and many other modern database systems offer a whole range of outer join types.

For more complicated subqueries, you can often create temporary tables to hold the subquery. In some cases, however, this option does not work. The most frequently encountered of these cases arises with `DELETE` statements, for which standard SQL does not support joins (except in subqueries). For this situation, there are three options available:

- The first option is to upgrade to MySQL 4.1, which does support subqueries in `DELETE` statements.
- The second option is to use a procedural programming language (such as Perl or PHP) to submit a `SELECT` query to obtain the primary keys for the records to be deleted, and then use these values to construct the `DELETE` statement (`DELETE FROM ... WHERE key_col IN (key1, key2, ...)`).
- The third option is to use interactive SQL to construct a set of `DELETE` statements automatically, using the MySQL extension `CONCAT()` (in lieu of the standard `||` operator). For example:

  ```
  SELECT
    CONCAT('DELETE FROM tab1 WHERE pkid = ', "'", tab1.pkid, "'", ';')
    FROM tab1, tab2
   WHERE tab1.col1 = tab2.col2;
  ```

  You can place this query in a script file, use the file as input to one instance of the **mysql** program, and use the program output as input to a second instance of **mysql**:

  ```
  shell> mysql --skip-column-names mydb < myscript.sql | mysql mydb
  ```

MySQL Server 4.0 supports multiple-table `DELETE` statements that can be used to efficiently delete rows based on information from one table or even from many tables at the same time. Multiple-table `UPDATE` statements are also supported as of MySQL 4.0.

13.1.9. TRUNCATE Syntax

`TRUNCATE TABLE` *`tbl_name`*

`TRUNCATE TABLE` empties a table completely. Logically, this is equivalent to a `DELETE` statement that deletes all rows, but there are practical differences under some circumstances.

For `InnoDB` before version 5.0.3, `TRUNCATE TABLE` is mapped to `DELETE`, so there is no difference. Starting with MySQL/InnoDB-5.0.3, fast `TRUNCATE TABLE` is available. The operation is still mapped to `DELETE` if there are foreign key constraints that reference the table.

For other storage engines, `TRUNCATE TABLE` differs from `DELETE FROM` in the following ways from MySQL 4.0 and up:

- Truncate operations drop and re-create the table, which is much faster than deleting rows one by one.
- Truncate operations are not transaction-safe; you get an error if you have an active transaction or an active table lock.
- The number of deleted rows is not returned.
- As long as the table definition file *tbl_name*.frm is valid, the table can be re-created as an empty table with TRUNCATE TABLE, even if the data or index files have become corrupted.
- The table handler does not remember the last used AUTO_INCREMENT value, but starts counting from the beginning. This is true even for MyISAM and InnoDB, which normally does not reuse sequence values.

In MySQL 3.23, TRUNCATE TABLE is mapped to COMMIT; DELETE FROM *tbl_name*, so it behaves like DELETE. See Section 13.1.1, "DELETE Syntax".

TRUNCATE TABLE is an Oracle SQL extension. This statement was added in MySQL 3.23.28, although from 3.23.28 to 3.23.32, the keyword TABLE must be omitted.

13.1.10. UPDATE Syntax

Single-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
    SET col_name1=expr1 [, col_name2=expr2 ...]
    [WHERE where_definition]
    [ORDER BY ...]
    [LIMIT row_count]
```

Multiple-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name [, tbl_name ...]
    SET col_name1=expr1 [, col_name2=expr2 ...]
    [WHERE where_definition]
```

The UPDATE statement updates columns in existing table rows with new values. The SET clause indicates which columns to modify and the values they should be given. The WHERE clause, if given, specifies which rows should be updated. Otherwise, all rows are updated. If the ORDER BY clause is specified, the rows are updated in the order that is specified. The LIMIT clause places a limit on the number of rows that can be updated.

The UPDATE statement supports the following modifiers:

- If you specify the LOW_PRIORITY keyword, execution of the UPDATE is delayed until no other clients are reading from the table.
- If you specify the IGNORE keyword, the update statement does not abort even if errors occur during the update. Rows for which duplicate-key conflicts occur are not updated. Rows for which columns are updated to values that would cause data conversion errors are updated to the closet valid values instead.

If you access a column from *tbl_name* in an expression, UPDATE uses the current value of the column. For example, the following statement sets the age column to one more than its current value:

```
mysql> UPDATE persondata SET age=age+1;
```

UPDATE assignments are evaluated from left to right. For example, the following statement doubles the age column, then increments it:

```
mysql> UPDATE persondata SET age=age*2, age=age+1;
```

If you set a column to the value it currently has, MySQL notices this and doesn't update it.

If you update a column that has been declared NOT NULL by setting to NULL, the column is set to the default value appropriate for the column type and the warning count is incremented. The default value is 0 for numeric types, the empty string (''') for string types, and the ``zero'' value for date and time types.

UPDATE returns the number of rows that were actually changed. In MySQL 3.22 or later, the mysql_info() C API function returns the number of rows that were matched and updated and the number of warnings that occurred during the UPDATE.

Starting from MySQL 3.23, you can use LIMIT *row_count* to restrict the scope of the UPDATE. A LIMIT clause works as follows:

- Before MySQL 4.0.13, LIMIT is a rows-affected restriction. The statement stops as soon as it has changed *row_count* rows that satisfy the WHERE clause.
- From 4.0.13 on, LIMIT is a rows-matched restriction. The statement stops as soon as it has found *row_count* rows that satisfy the WHERE clause, whether or not they actually were changed.

If an UPDATE statement includes an ORDER BY clause, the rows are updated in the order specified by the clause. ORDER BY can be used from MySQL 4.0.0.

Starting with MySQL 4.0.4, you can also perform UPDATE operations that cover multiple tables:

```
UPDATE items,month SET items.price=month.price
WHERE items.id=month.id;
```

The example shows an inner join using the comma operator, but multiple-table UPDATE statements can use any type of join allowed in SELECT statements, such as LEFT JOIN.

Note: You cannot use ORDER BY or LIMIT with multiple-table UPDATE.

Before MySQL 4.0.18, you need the UPDATE privilege for all tables used in a multiple-table UPDATE, even if they were not updated. As of MySQL 4.0.18, you need only the SELECT privilege for any columns that are read but not modified.

If you use a multiple-table UPDATE statement involving InnoDB tables for which there are foreign key constraints, the MySQL optimizer might process tables in an order that differs from that of their parent/child relationship. In this case, the statement fails and rolls back. Instead, update a single table and rely on the ON UPDATE capabilities that InnoDB provides to cause the other tables to be modified accordingly.

Currently, you cannot update a table and select from the same table in a subquery.

13.2. Data Definition Statements

13.2.1. ALTER DATABASE Syntax

```
ALTER {DATABASE | SCHEMA} [db_name]
    alter_specification [, alter_specification] ...

alter_specification:
    [DEFAULT] CHARACTER SET charset_name
  | [DEFAULT] COLLATE collation_name
```

ALTER DATABASE allows you to change the overall characteristics of a database. These characteristics are stored in the db.opt file in the database directory. To use ALTER DATABASE, you need the ALTER privilege on the database.

The CHARACTER SET clause changes the default database character set. The COLLATE clause changes the default database collation. Character set and collation names are discussed in Chapter 10, *Character Set Support*.

ALTER DATABASE was added in MySQL 4.1.1. Beginning with MySQL 4.1.8, the database name can be omitted. The statement applies to the default database in this case. ALTER SCHEMA can be used as of MySQL 5.0.2.

13.2.2. ALTER TABLE Syntax

```
ALTER [IGNORE] TABLE tbl_name
    alter_specification [, alter_specification] ...

alter_specification:
    ADD [COLUMN] column_definition [FIRST | AFTER col_name ]
  | ADD [COLUMN] (column_definition,...)
  | ADD INDEX [index_name] [index_type] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]]
        PRIMARY KEY [index_type] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]]
        UNIQUE [index_name] [index_type] (index_col_name,...)
  | ADD [FULLTEXT|SPATIAL] [index_name] (index_col_name,...)
  | ADD [CONSTRAINT [symbol]]
        FOREIGN KEY [index_name] (index_col_name,...)
        [reference_definition]
  | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
  | CHANGE [COLUMN] old_col_name column_definition
        [FIRST|AFTER col_name]
  | MODIFY [COLUMN] column_definition [FIRST | AFTER col_name]
  | DROP [COLUMN] col_name
  | DROP PRIMARY KEY
  | DROP INDEX index_name
  | DROP FOREIGN KEY fk_symbol
  | DISABLE KEYS
  | ENABLE KEYS
  | RENAME [TO] new_tbl_name
  | ORDER BY col_name
  | CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
  | [DEFAULT] CHARACTER SET charset_name [COLLATE collation_name]
  | DISCARD TABLESPACE
  | IMPORT TABLESPACE
  | table_options
```

ALTER TABLE allows you to change the structure of an existing table. For example, you can add or delete columns, create or destroy indexes, change the type of existing columns, or rename columns or the table itself. You can also change the comment for the table and type of the table.

The syntax for many of the allowable alterations is similar to clauses of the `CREATE TABLE` statement. See Section 13.2.6, "`CREATE TABLE` Syntax".

Some operations may result in warnings if attempted on a table for which the storage engine does not support the operation. In MySQL 4.1 and up, these warnings can be displayed with `SHOW WARNINGS`. See Section 13.5.4.20, "`SHOW WARNINGS` Syntax".

If you use `ALTER TABLE` to change a column specification but `DESCRIBE` *`tbl_name`* indicates that your column was not changed, it is possible that MySQL ignored your modification for one of the reasons described in Section 13.2.6.1, "Silent Column Specification Changes". For example, if you try to change a `VARCHAR` column to `CHAR`, MySQL still uses `VARCHAR` if the table contains other variable-length columns.

`ALTER TABLE` works by making a temporary copy of the original table. The alteration is performed on the copy, then the original table is deleted and the new one is renamed. While `ALTER TABLE` is executing, the original table is readable by other clients. Updates and writes to the table are stalled until the new table is ready, then are automatically redirected to the new table without any failed updates.

Note that if you use any other option to `ALTER TABLE` than `RENAME`, MySQL always creates a temporary table, even if the data wouldn't strictly need to be copied (such as when you change the name of a column). We plan to fix this in the future, but because `ALTER TABLE` is not a statement that is normally used frequently, this isn't high on our TODO list. For `MyISAM` tables, you can speed up the index re-creation operation (which is the slowest part of the alteration process) by setting the `myisam_sort_buffer_size` system variable to a high value.

- To use `ALTER TABLE`, you need `ALTER`, `INSERT`, and `CREATE` privileges for the table.
- `IGNORE` is a MySQL extension to standard SQL. It controls how `ALTER TABLE` works if there are duplicates on unique keys in the new table. If `IGNORE` isn't specified, the copy is aborted and rolled back if duplicate-key errors occur. If `IGNORE` is specified, then for rows with duplicates on a unique key, only the first row is used. The others are deleted.
- You can issue multiple `ADD`, `ALTER`, `DROP`, and `CHANGE` clauses in a single `ALTER TABLE` statement. This is a MySQL extension to standard SQL, which allows only one of each clause per `ALTER TABLE` statement. For example, to drop multiple columns in a single statement:

  ```
  mysql> ALTER TABLE t2 DROP COLUMN c, DROP COLUMN d;
  ```

- `CHANGE` *`col_name`*, `DROP` *`col_name`*, and `DROP INDEX` are MySQL extensions to standard SQL.
- `MODIFY` is an Oracle extension to `ALTER TABLE`.
- The word `COLUMN` is purely optional and can be omitted.
- If you use `ALTER TABLE` *`tbl_name`* `RENAME TO` *`new_tbl_name`* without any other options, MySQL simply renames any files that correspond to the table *`tbl_name`*. There is no need to create a temporary table. (You can also use the `RENAME TABLE` statement to rename tables. See Section 13.2.12, "`RENAME TABLE` Syntax".)
- *`column_definition`* clauses use the same syntax for `ADD` and `CHANGE` as for `CREATE TABLE`. Note that this syntax includes the column name, not just the column type. See Section 13.2.6, "`CREATE TABLE` Syntax".
- You can rename a column using a `CHANGE` *`old_col_name column_definition`* clause. To do so, specify the old and new column names and the type that the column currently has. For example, to rename an `INTEGER` column from `a` to `b`, you can do this:

```
mysql> ALTER TABLE t1 CHANGE a b INTEGER;
```

If you want to change a column's type but not the name, CHANGE syntax still requires an old and new column name, even if they are the same. For example:

```
mysql> ALTER TABLE t1 CHANGE b b BIGINT NOT NULL;
```

However, as of MySQL 3.22.16a, you can also use MODIFY to change a column's type without renaming it:

```
mysql> ALTER TABLE t1 MODIFY b BIGINT NOT NULL;
```

- If you use CHANGE or MODIFY to shorten a column for which an index exists on part of the column (for example, if you have an index on the first 10 characters of a VARCHAR column), you cannot make the column shorter than the number of characters that are indexed.
- When you change a column type using CHANGE or MODIFY, MySQL tries to convert existing column values to the new type as well as possible.
- In MySQL 3.22 or later, you can use FIRST or AFTER *col_name* to add a column at a specific position within a table row. The default is to add the column last. From MySQL 4.0.1 on, you can also use FIRST and AFTER in CHANGE or MODIFY operations.
- ALTER COLUMN specifies a new default value for a column or removes the old default value. If the old default is removed and the column can be NULL, the new default is NULL. If the column cannot be NULL, MySQL assigns a default value, as described in Section 13.2.6, "CREATE TABLE Syntax".
- DROP INDEX removes an index. This is a MySQL extension to standard SQL. See Section 13.2.9, "DROP INDEX Syntax".
- If columns are dropped from a table, the columns are also removed from any index of which they are a part. If all columns that make up an index are dropped, the index is dropped as well.
- If a table contains only one column, the column cannot be dropped. If what you intend is to remove the table, use DROP TABLE instead.
- DROP PRIMARY KEY drops the primary index. (Prior to MySQL 4.1.2, if no primary index exists, DROP PRIMARY KEY drops the first UNIQUE index in the table. MySQL marks the first UNIQUE key as the PRIMARY KEY if no PRIMARY KEY was specified explicitly.)

  If you add a UNIQUE INDEX or PRIMARY KEY to a table, it is stored before any non-unique index so that MySQL can detect duplicate keys as early as possible.
- ORDER BY allows you to create the new table with the rows in a specific order. Note that the table does not remain in this order after inserts and deletes. This option is mainly useful when you know that you are mostly going to query the rows in a certain order; by using this option after big changes to the table, you might be able to get higher performance. In some cases, it might make sorting easier for MySQL if the table is in order by the column that you want to order it by later.
- If you use ALTER TABLE on a MyISAM table, all non-unique indexes are created in a separate batch (as for REPAIR TABLE). This should make ALTER TABLE much faster when you have many indexes.

  As of MySQL 4.0, this feature can be activated explicitly. ALTER TABLE ... DISABLE KEYS

tells MySQL to stop updating non-unique indexes for a `MyISAM` table. `ALTER TABLE ... ENABLE KEYS` then should be used to re-create missing indexes. MySQL does this with a special algorithm that is much faster than inserting keys one by one, so disabling keys before performing bulk insert operations should give a considerable speedup. Using `ALTER TABLE ... DISABLE KEYS` requires the `INDEX` privilege in addition to the privileges mentioned earlier.

- The `FOREIGN KEY` and `REFERENCES` clauses are supported by the `InnoDB` storage engine, which implements `ADD [CONSTRAINT [symbol]] FOREIGN KEY (...) REFERENCES ... (...)`. See Section 15.7.4, "`FOREIGN KEY` Constraints". For other storage engines, the clauses are parsed but ignored. The `CHECK` clause is parsed but ignored by all storage engines. See Section 13.2.6, "`CREATE TABLE` Syntax". The reason for accepting but ignoring syntax clauses is for compatibility, to make it easier to port code from other SQL servers, and to run applications that create tables with references. See Section 1.5.5, "MySQL Differences from Standard SQL".

- Starting from MySQL 4.0.13, `InnoDB` supports the use of `ALTER TABLE` to drop foreign keys:

  ```
  ALTER TABLE yourtablename DROP FOREIGN KEY fk_symbol;
  ```

  For more information, see Section 15.7.4, "`FOREIGN KEY` Constraints".

- `ALTER TABLE` ignores the `DATA DIRECTORY` and `INDEX DIRECTORY` table options.

- From MySQL 4.1.2 on, if you want to change all character columns (`CHAR`, `VARCHAR`, `TEXT`) to a new character set, use a statement like this:

  ```
  ALTER TABLE tbl_name CONVERT TO CHARACTER SET charset_name;
  ```

  This is useful, for example, after upgrading from MySQL 4.0.x to 4.1.x. See Section 10.10, "Upgrading Character Sets from MySQL 4.0".

  Warning: The preceding operation converts column values between the character sets. This is *not* what you want if you have a column in one character set (like `latin1`) but the stored values actually use some other, incompatible character set (like `utf8`). In this case, you have to do the following for each such column:

  ```
  ALTER TABLE t1 CHANGE c1 c1 BLOB;
  ALTER TABLE t1 CHANGE c1 c1 TEXT CHARACTER SET utf8;
  ```

  The reason this works is that there is no conversion when you convert to or from `BLOB` columns.

  To change only the *default* character set for a table, use this statement:

  ```
  ALTER TABLE tbl_name DEFAULT CHARACTER SET charset_name;
  ```

  The word `DEFAULT` is optional. The default character set is the character set that is used if you don't specify the character set for a new column you add to a table (for example, with `ALTER TABLE ... ADD column`).

  Warning: From MySQL 4.1.2 and up, `ALTER TABLE ... DEFAULT CHARACTER SET` and `ALTER TABLE ... CHARACTER SET` are equivalent and change only the default table character set. In MySQL 4.1 releases before 4.1.2, `ALTER TABLE ... DEFAULT CHARACTER SET` changes the default character set, but `ALTER TABLE ... CHARACTER SET` (without `DEFAULT`) changes the default character set *and also* converts all columns to the new character set.

- For an InnoDB table that is created with its own tablespace in an .ibd file, that file can be discarded and imported. To discard the .ibd file, use this statement:

  ```
  ALTER TABLE tbl_name DISCARD TABLESPACE;
  ```

  This deletes the current .ibd file, so be sure that you have a backup first. Attempting to access the table while the tablespace file is discarded results in an error.

  To import the backup .ibd file back into the table, copy it into the database directory, then issue this statement:

  ```
  ALTER TABLE tbl_name IMPORT TABLESPACE;
  ```

  See Section 15.7.6, “Using Per-Table Tablespaces”.

- With the mysql_info() C API function, you can find out how many records were copied, and (when IGNORE is used) how many records were deleted due to duplication of unique key values. See Section 22.2.3.31, “mysql_info()”.

Here are some examples that show uses of ALTER TABLE. Begin with a table t1 that is created as shown here:

```
mysql> CREATE TABLE t1 (a INTEGER,b CHAR(10));
```

To rename the table from t1 to t2:

```
mysql> ALTER TABLE t1 RENAME t2;
```

To change column a from INTEGER to TINYINT NOT NULL (leaving the name the same), and to change column b from CHAR(10) to CHAR(20) as well as renaming it from b to c:

```
mysql> ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

To add a new TIMESTAMP column named d:

```
mysql> ALTER TABLE t2 ADD d TIMESTAMP;
```

To add indexes on column d and on column a:

```
mysql> ALTER TABLE t2 ADD INDEX (d), ADD INDEX (a);
```

To remove column c:

```
mysql> ALTER TABLE t2 DROP COLUMN c;
```

To add a new AUTO_INCREMENT integer column named c:

```
mysql> ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
    ->     ADD PRIMARY KEY (c);
```

Note that we indexed c (as a PRIMARY KEY), because AUTO_INCREMENT columns must be indexed, and also that we declare c as NOT NULL, because primary key columns cannot be NULL.

When you add an AUTO_INCREMENT column, column values are filled in with sequence numbers for you automatically. For MyISAM tables, you can set the first sequence number by executing SET INSERT_ID=*value* before ALTER TABLE or by using the AUTO_INCREMENT=*value* table option. See Section 13.5.3, "SET Syntax".

From MySQL 5.0.3, you can use the ALTER TABLE ... AUTO_INCREMENT=*value* table option for InnoDB tables to set the sequence number for new rows if the value is greater than the maximum value in the AUTO_INCREMENT column. If the value is less than the maximum column value, no error message is given and the current sequence value is not changed.

With MyISAM tables, if you don't change the AUTO_INCREMENT column, the sequence number is not affected. If you drop an AUTO_INCREMENT column and then add another AUTO_INCREMENT column, the numbers are resequenced beginning with 1.

See Section A.7.1, "Problems with ALTER TABLE".

13.2.3. ALTER VIEW Syntax

```
ALTER [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

This statement changes the definition of an existing view. The syntax is similar to that for CREATE VIEW. See Section 13.2.7, "CREATE VIEW Syntax". This statement requires the CREATE VIEW and DELETE privileges for the view, and some privilege for each column referred to in the SELECT statement.

This statement was added in MySQL 5.0.1.

13.2.4. CREATE DATABASE Syntax

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_specification [, create_specification] ...]

create_specification:
    [DEFAULT] CHARACTER SET charset_name
  | [DEFAULT] COLLATE collation_name
```

CREATE DATABASE creates a database with the given name. To use CREATE DATABASE, you need the CREATE privilege on the database.

Rules for allowable database names are given in Section 9.2, "Database, Table, Index, Column, and Alias Names". An error occurs if the database exists and you didn't specify IF NOT EXISTS.

As of MySQL 4.1.1, create_specification options can be given to specify database characteristics. Database characteristics are stored in the db.opt file in the database directory. The CHARACTER SET clause specifies the default database character set. The COLLATE clause specifies the default database collation. Character set and collation names are discussed in Chapter 10, *Character Set Support*.

Databases in MySQL are implemented as directories containing files that correspond to tables in the database. Because there are no tables in a database when it is initially created, the CREATE DATABASE

statement only creates a directory under the MySQL data directory (and the `db.opt` file, for MySQL 4.1.1 and up).

If you manually create a directory under the data directory (for example, with **mkdir**), the server considers it a database directory and it shows up in the output of `SHOW DATABASES`.

`CREATE SCHEMA` can be used as of MySQL 5.0.2.

You can also use the **mysqladmin** program to create databases. See Section 8.4, "**mysqladmin**, Administering a MySQL Server".

13.2.5. CREATE INDEX Syntax

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
    [USING index_type]
    ON tbl_name (index_col_name,...)

index_col_name:
    col_name [(length)] [ASC | DESC]
```

In MySQL 3.22 or later, `CREATE INDEX` is mapped to an `ALTER TABLE` statement to create indexes. See Section 13.2.2, "`ALTER TABLE` Syntax". The `CREATE INDEX` statement doesn't do anything prior to MySQL 3.22.

Normally, you create all indexes on a table at the time the table itself is created with `CREATE TABLE`. See Section 13.2.6, "`CREATE TABLE` Syntax". `CREATE INDEX` allows you to add indexes to existing tables.

A column list of the form `(col1,col2,...)` creates a multiple-column index. Index values are formed by concatenating the values of the given columns.

For `CHAR` and `VARCHAR` columns, indexes can be created that use only part of a column, using *`col_name(length)`* syntax to index a prefix consisting of the first *`length`* characters of each column value. `BLOB` and `TEXT` columns also can be indexed, but a prefix length *must* be given.

The statement shown here creates an index using the first 10 characters of the `name` column:

```
CREATE INDEX part_of_name ON customer (name(10));
```

Because most names usually differ in the first 10 characters, this index should not be much slower than an index created from the entire `name` column. Also, using partial columns for indexes can make the index file much smaller, which could save a lot of disk space and might also speed up `INSERT` operations!

Prefixes can be up to 255 bytes long (or 1000 bytes for `MyISAM` and `InnoDB` tables as of MySQL 4.1.2). Note that prefix limits are measured in bytes, whereas the prefix length in `CREATE INDEX` statements is interpreted as number of characters. Take this into account when specifying a prefix length for a column that uses a multi-byte character set.

You can add an index on a column that can have `NULL` values only if you are using MySQL 3.23.2 or newer and are using the `MyISAM`, `InnoDB`, or `BDB` table type. You can only add an index on a `BLOB` or `TEXT` column if you are using MySQL 3.23.2 or newer and are using the `MyISAM` or `BDB` table type, or MySQL 4.0.14 or newer and the `InnoDB` table type.

An *`index_col_name`* specification can end with `ASC` or `DESC`. These keywords are allowed for future extensions for specifying ascending or descending index value storage. Currently they are parsed but ignored; index values are always stored in ascending order.

From MySQL 4.1.0 on, some storage engines allow you to specify an index type when creating an index. The syntax for the *index_type* specifier is `USING type_name`. The allowable `type_name` values supported by different storage engines are shown in the following table. Where multiple index types are listed, the first one is the default when no *index_type* specifier is given.

Storage Engine	Allowable Index Types
MyISAM	BTREE
InnoDB	BTREE
MEMORY/HEAP	HASH, BTREE

Example:

```
CREATE TABLE lookup (id INT) ENGINE = MEMORY;
CREATE INDEX id_index USING BTREE ON lookup (id);
```

`TYPE type_name` can be used as a synonym for `USING type_name` to specify an index type. However, `USING` is the preferred form. Also, the index name that precedes the index type in the index specification syntax is not optional with `TYPE`. This is because, unlike `USING`, `TYPE` is not a reserved word and thus is interpreted as an index name.

If you specify an index type that is not legal for a storage engine, but there is another index type available that the engine can use without affecting query results, the engine uses the available type.

For more information about how MySQL uses indexes, see Section 7.4.5, “How MySQL Uses Indexes”.

`FULLTEXT` indexes can index only `CHAR`, `VARCHAR`, and `TEXT` columns, and only in `MyISAM` tables. `FULLTEXT` indexes are available in MySQL 3.23.23 or later. Section 12.6, “Full-Text Search Functions”.

`SPATIAL` indexes can index only spatial columns, and only in `MyISAM` tables. `SPATIAL` indexes are available in MySQL 4.1 or later. Spatial column types are described in Chapter 18, *Spatial Extensions in MySQL*.

13.2.6. CREATE TABLE Syntax

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [(create_definition,...)]
    [table_options] [select_statement]
```

Or:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [(] LIKE old_tbl_name [)];

create_definition:
    column_definition
  | [CONSTRAINT [symbol]] PRIMARY KEY [index_type] (index_col_name,...)
  | KEY [index_name] [index_type] (index_col_name,...)
  | INDEX [index_name] [index_type] (index_col_name,...)
  | [CONSTRAINT [symbol]] UNIQUE [INDEX]
        [index_name] [index_type] (index_col_name,...)
  | [FULLTEXT|SPATIAL] [INDEX] [index_name] (index_col_name,...)
  | [CONSTRAINT [symbol]] FOREIGN KEY
        [index_name] (index_col_name,...) [reference_definition]
  | CHECK (expr)
```

```
column_definition:
    col_name type [NOT NULL | NULL] [DEFAULT default_value]
        [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
        [COMMENT 'string'] [reference_definition]

type:
    TINYINT[(length)] [UNSIGNED] [ZEROFILL]
  | SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
  | MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
  | INT[(length)] [UNSIGNED] [ZEROFILL]
  | INTEGER[(length)] [UNSIGNED] [ZEROFILL]
  | BIGINT[(length)] [UNSIGNED] [ZEROFILL]
  | REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
  | DECIMAL(length,decimals) [UNSIGNED] [ZEROFILL]
  | NUMERIC(length,decimals) [UNSIGNED] [ZEROFILL]
  | DATE
  | TIME
  | TIMESTAMP
  | DATETIME
  | CHAR(length) [BINARY | ASCII | UNICODE]
  | VARCHAR(length) [BINARY]
  | TINYBLOB
  | BLOB
  | MEDIUMBLOB
  | LONGBLOB
  | TINYTEXT [BINARY]
  | TEXT [BINARY]
  | MEDIUMTEXT [BINARY]
  | LONGTEXT [BINARY]
  | ENUM(value1,value2,value3,...)
  | SET(value1,value2,value3,...)
  | spatial_type

index_col_name:
    col_name [(length)] [ASC | DESC]

reference_definition:
    REFERENCES tbl_name [(index_col_name,...)]
               [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
               [ON DELETE reference_option]
               [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

table_options: table_option [table_option] ...

table_option:
    {ENGINE|TYPE} = engine_name
  | AUTO_INCREMENT = value
  | AVG_ROW_LENGTH = value
  | CHECKSUM = {0 | 1}
  | COMMENT = 'string'
  | MAX_ROWS = value
  | MIN_ROWS = value
  | PACK_KEYS = {0 | 1 | DEFAULT}
  | PASSWORD = 'string'
  | DELAY_KEY_WRITE = {0 | 1}
  | ROW_FORMAT = {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}
  | RAID_TYPE = { 1 | STRIPED | RAID0 }
        RAID_CHUNKS = value
        RAID_CHUNKSIZE = value
```

```
  | UNION = (tbl_name[,tbl_name]...)
  | INSERT_METHOD = { NO | FIRST | LAST }
  | DATA DIRECTORY = 'absolute path to directory'
  | INDEX DIRECTORY = 'absolute path to directory'
  | [DEFAULT] CHARACTER SET charset_name [COLLATE collation_name]

select_statement:
    [IGNORE | REPLACE] [AS] SELECT ...   (Some legal select statement)
```

`CREATE TABLE` creates a table with the given name. You must have the `CREATE` privilege for the table.

Rules for allowable table names are given in Section 9.2, "Database, Table, Index, Column, and Alias Names". By default, the table is created in the current database. An error occurs if the table exists, if there is no current database, or if the database does not exist.

In MySQL 3.22 or later, the table name can be specified as *`db_name.tbl_name`* to create the table in a specific database. This works whether or not there is a current database. If you use quoted identifiers, quote the database and table names separately. For example, `` `mydb`.`mytbl` `` is legal, but `` `mydb.mytbl` `` is not.

From MySQL 3.23 on, you can use the `TEMPORARY` keyword when creating a table. A `TEMPORARY` table is visible only to the current connection, and is dropped automatically when the connection is closed. This means that two different connections can use the same temporary table name without conflicting with each other or with an existing non-`TEMPORARY` table of the same name. (The existing table is hidden until the temporary table is dropped.) From MySQL 4.0.2 on, you must have the `CREATE TEMPORARY TABLES` privilege to be able to create temporary tables.

In MySQL 3.23 or later, you can use the keywords `IF NOT EXISTS` so that an error does not occur if the table exists. Note that there is no verification that the existing table has a structure identical to that indicated by the `CREATE TABLE` statement.

MySQL represents each table by an `.frm` table format (definition) file in the database directory. The storage engine for the table might create other files as well. In the case of `MyISAM` tables, the storage engine creates three files for a table named *`tbl_name`*:

File	Purpose
`tbl_name``.frm`	Table format (definition) file
`tbl_name``.MYD`	Data file
`tbl_name``.MYI`	Index file

The files created by each storage engine to represent tables are described in Chapter 14, *MySQL Storage Engines and Table Types*.

For general information on the properties of the various column types, see Chapter 11, *Column Types*. For information about spatial column types, see Chapter 18, *Spatial Extensions in MySQL*.

- If neither `NULL` nor `NOT NULL` is specified, the column is treated as though `NULL` had been specified.
- An integer column can have the additional attribute `AUTO_INCREMENT`. When you insert a value of `NULL` (recommended) or `0` into an indexed `AUTO_INCREMENT` column, the column is set to the next sequence value. Typically this is *`value`*`+1`, where *`value`* is the largest value for the column currently in the table. `AUTO_INCREMENT` sequences begin with `1`. See Section 22.2.3.33, "`mysql_insert_id()`".

As of MySQL 4.1.1, specifying the NO_AUTO_VALUE_ON_ZERO flag for the --sql-mode server option or the sql_mode system variable allows you to store 0 in AUTO_INCREMENT columns as 0 without generating a new sequence value. See Section 5.2.1, "**mysqld** Command-Line Options".

Note: There can be only one AUTO_INCREMENT column per table, it must be indexed, and it cannot have a DEFAULT value. As of MySQL 3.23, an AUTO_INCREMENT column works properly only if it contains only positive values. Inserting a negative number is regarded as inserting a very large positive number. This is done to avoid precision problems when numbers ``wrap'' over from positive to negative and also to ensure that you don't accidentally get an AUTO_INCREMENT column that contains 0.

For MyISAM and BDB tables, you can specify an AUTO_INCREMENT secondary column in a multiple-column key. See Section 3.6.9, "Using AUTO_INCREMENT".

To make MySQL compatible with some ODBC applications, you can find the AUTO_INCREMENT value for the last inserted row with the following query:

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

- As of MySQL 4.1, character column definitions can include a CHARACTER SET attribute to specify the character set and, optionally, a collation for the column. For details, see Chapter 10, *Character Set Support*.

  ```
  CREATE TABLE t (c CHAR(20) CHARACTER SET utf8 COLLATE utf8_bin);
  ```

  Also as of 4.1, MySQL interprets length specifications in character column definitions in characters. (Earlier versions interpret them in bytes.)

- NULL values are handled differently for TIMESTAMP columns than for other column types. Before MySQL 4.1.6, you cannot store a literal NULL in a TIMESTAMP column; setting the column to NULL sets it to the current date and time. Because TIMESTAMP columns behave this way, the NULL and NOT NULL attributes do not apply in the normal way and are ignored if you specify them. On the other hand, to make it easier for MySQL clients to use TIMESTAMP columns, the server reports that such columns can be assigned NULL values (which is true), even though TIMESTAMP never actually contains a NULL value. You can see this when you use DESCRIBE *tbl_name* to get a description of your table.

  Note that setting a TIMESTAMP column to 0 is not the same as setting it to NULL, because 0 is a valid TIMESTAMP value.

- The DEFAULT clause specifies a default value for a column. With one exception, the default value must be a constant; it cannot be a function or an expression. This means, for example, that you cannot set the default for a date column to be the value of a function such as NOW() or CURRENT_DATE. The exception is that you can specify CURRENT_TIMESTAMP as the default for a TIMESTAMP column as of MySQL 4.1.2. See Section 11.3.1.2, "TIMESTAMP Properties as of MySQL 4.1".

  Prior to MySQL 5.0.2, if a column definition includes no explicit DEFAULT value, MySQL determines the default value as follows:

  If the column can take NULL as a value, the column is defined with an explicit DEFAULT NULL clause.

  If the column cannot take NULL as the value, MySQL defines the column with an explicit DEFAULT

clause, using the implicit default value for the column data type. Implicit defaults are defined as follows:

- For numeric types other than those declared with the `AUTO_INCREMENT` attribute, the default is `0`. For an `AUTO_INCREMENT` column, the default value is the next value in the sequence.
- For date and time types other than `TIMESTAMP`, the default is the appropriate ``zero'' value for the type. For the first `TIMESTAMP` column in a table, the default value is the current date and time. See Section 11.3, “Date and Time Types”.
- For string types other than `ENUM`, the default value is the empty string. For `ENUM`, the default is the first enumeration value.

`BLOB` and `TEXT` columns cannot be assigned a default value.

As of MySQL 5.0.2, if a column definition includes no explicit `DEFAULT` value, MySQL determines the default value as follows:

If the column can take `NULL` as a value, the column is defined with an explicit `DEFAULT NULL` clause. This is the same as before 5.0.2.

If the column cannot take `NULL` as the value, MySQL defines the column with no explicit `DEFAULT` clause. For data entry, if an `INSERT` or `REPLACE` statement includes no value for the column, MySQL handles the column according to the SQL mode in effect at the time:

- If strict mode is not enabled, MySQL sets the column to the implicit value for the column data type.
- If strict mode is enabled, an error occurs for transactional tables and the statement is rolled back. For non-transactional tables, an error occurs unless the row is the second or subsequent row of a multiple-row statement, in which case a warning occurs.

See Section 5.2.2, “The Server SQL Mode”.

For a given table, you can use the `SHOW CREATE TABLE` statement to see which columns have an explicit `DEFAULT` clause.

- A comment for a column can be specified with the `COMMENT` option. The comment is displayed by the `SHOW CREATE TABLE` and `SHOW FULL COLUMNS` statements. This option is operational as of MySQL 4.1. (It is allowed but ignored in earlier versions.)
- From MySQL 4.1.0 on, the attribute `SERIAL` can be used as an alias for `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE`.
- `KEY` is normally a synonym for `INDEX`. From MySQL 4.1, the key attribute `PRIMARY KEY` can also be specified as just `KEY` when given in a column definition. This was implemented for compatibility with other database systems.
- In MySQL, a `UNIQUE` index is one in which all values in the index must be distinct. An error occurs if you try to add a new row with a key that matches an existing row. The exception to this is that if a column in the index is allowed to contain `NULL` values, it can contain multiple `NULL` values. This exception does not apply to `BDB` tables, for which an indexed column allows only a single `NULL`.
- A `PRIMARY KEY` is a unique `KEY` where all key columns must be defined as `NOT NULL`. If they are not explicitly declared as `NOT NULL`, MySQL declares them so implicitly (and silently). A table can have only one `PRIMARY KEY`. If you don't have a `PRIMARY KEY` and an application asks for the `PRIMARY KEY` in your tables, MySQL returns the first `UNIQUE` index that has no `NULL`

columns as the PRIMARY KEY.

- In the created table, a PRIMARY KEY is placed first, followed by all UNIQUE indexes, and then the non-unique indexes. This helps the MySQL optimizer to prioritize which index to use and also more quickly to detect duplicated UNIQUE keys.

- A PRIMARY KEY can be a multiple-column index. However, you cannot create a multiple-column index using the PRIMARY KEY key attribute in a column specification. Doing so only marks that single column as primary. You must use a separate PRIMARY KEY(index_col_name, ...) clause.

- If a PRIMARY KEY or UNIQUE index consists of only one column that has an integer type, you can also refer to the column as _rowid in SELECT statements (new in MySQL 3.23.11).

- In MySQL, the name of a PRIMARY KEY is PRIMARY. For other indexes, if you don't assign a name, the index is assigned the same name as the first indexed column, with an optional suffix (_2, _3, ...) to make it unique. You can see index names for a table using SHOW INDEX FROM *tbl_name*. See Section 13.5.4.11, "SHOW INDEX Syntax".

- From MySQL 4.1.0 on, some storage engines allow you to specify an index type when creating an index. The syntax for the *index_type* specifier is USING type_name.

  Example:

  ```
  CREATE TABLE lookup
      (id INT, INDEX USING BTREE (id))
      ENGINE = MEMORY;
  ```

  For details about USING, see Section 13.2.5, "CREATE INDEX Syntax".

  For more information about how MySQL uses indexes, see Section 7.4.5, "How MySQL Uses Indexes".

- Only the MyISAM, InnoDB, BDB, and (as of MySQL 4.0.2) MEMORY storage engines support indexes on columns that can have NULL values. In other cases, you must declare indexed columns as NOT NULL or an error results.

- With *col_name*(*length*) syntax in an index specification, you can create an index that uses only the first *length* characters of a CHAR or VARCHAR column. Indexing only a prefix of column values like this can make the index file much smaller. See Section 7.4.3, "Column Indexes".

  The MyISAM and (as of MySQL 4.0.14) InnoDB storage engines also support indexing on BLOB and TEXT columns. When indexing a BLOB or TEXT column, you *must* specify a prefix length for the index. For example:

  ```
  CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
  ```

  Prefixes can be up to 255 bytes long (or 1000 bytes for MyISAM and InnoDB tables as of MySQL 4.1.2). Note that prefix limits are measured in bytes, whereas the prefix length in CREATE TABLE statements is interpreted as number of characters. Take this into account when specifying a prefix length for a column that uses a multi-byte character set.

- An *index_col_name* specification can end with ASC or DESC. These keywords are allowed for future extensions for specifying ascending or descending index value storage. Currently they are parsed but ignored; index values are always stored in ascending order.

- When you use ORDER BY or GROUP BY with a TEXT or BLOB column, the server sorts values us-

ing only the initial number of bytes indicated by the `max_sort_length` system variable. See Section 11.4.3, "The `BLOB` and `TEXT` Types".

- In MySQL 3.23.23 or later, you can create special `FULLTEXT` indexes. They are used for full-text search. Only the `MyISAM` table type supports `FULLTEXT` indexes. They can be created only from `CHAR`, `VARCHAR`, and `TEXT` columns. Indexing always happens over the entire column; partial indexing is not supported and any prefix length is ignored if specified. See Section 12.6, "Full-Text Search Functions" for details of operation.

- In MySQL 4.1 or later, you can create `SPATIAL` indexes on spatial column types. Spatial types are supported only for `MyISAM` tables and indexed columns must be declared as `NOT NULL`. See Chapter 18, *Spatial Extensions in MySQL*.

- In MySQL 3.23.44 or later, `InnoDB` tables support checking of foreign key constraints. See Chapter 15, *The `InnoDB` Storage Engine*. Note that the `FOREIGN KEY` syntax in `InnoDB` is more restrictive than the syntax presented for the `CREATE TABLE` statement at the beginning of this section: The columns of the referenced table must always be explicitly named. `InnoDB` supports both `ON DELETE` and `ON UPDATE` actions on foreign keys as of MySQL 3.23.50 and 4.0.8, respectively. For the precise syntax, see Section 15.7.4, "`FOREIGN KEY` Constraints".

  For other storage engines, MySQL Server parses the `FOREIGN KEY` and `REFERENCES` syntax in `CREATE TABLE` statements, but without further action being taken. The `CHECK` clause is parsed but ignored by all storage engines. See Section 1.5.5.5, "Foreign Keys".

- For `MyISAM` and `ISAM` tables, each `NULL` column takes one bit extra, rounded up to the nearest byte. The maximum record length in bytes can be calculated as follows:

  ```
  row length = 1
               + (sum of column lengths)
               + (number of NULL columns + delete_flag + 7)/8
               + (number of variable-length columns)
  ```

  `delete_flag` is 1 for tables with static record format. Static tables use a bit in the row record for a flag that indicates whether the row has been deleted. *`delete_flag`* is 0 for dynamic tables because the flag is stored in the dynamic row header.

  These calculations do not apply for `InnoDB` tables, for which storage size is no different for `NULL` columns than for `NOT NULL` columns.

The *`table_options`* part of the `CREATE TABLE` syntax can be used in MySQL 3.23 and above.

The `ENGINE` and `TYPE` options specify the storage engine for the table. `ENGINE` was added in MySQL 4.0.18 (for 4.0) and 4.1.2 (for 4.1). It is the preferred option name as of those versions, and `TYPE` has become deprecated. `TYPE` is supported throughout the 4.x series, but likely will be removed in MySQL 5.1.

The `ENGINE` and `TYPE` options take the following values:

Storage Engine	Description
`ARCHIVE`	The archving storage engine. See Section 14.7, "The `ARCHIVE` Storage Engine".
`BDB`	Transaction-safe tables with page locking. See Section 14.4, "The `BDB` (`BerkeleyDB`) Storage Engine".
`BerkeleyDB`	An alias for `BDB`.
`CSV`	Tables that store rows in comma-separated values format. See Section 14.8, "The `CSV` Storage Engine".

EXAMPLE	An example engine. See Section 14.5, “The EXAMPLE Storage Engine”.
FEDERATED	Storage engine that accesses remote tables. See Section 14.6, “The FEDERATED Storage Engine”.
HEAP	The data for this table is stored only in memory. See Section 14.3, “The MEMORY (HEAP) Storage Engine”.
ISAM	The original MySQL storage engine. See Section 14.9, “The ISAM Storage Engine”.
InnoDB	Transaction-safe tables with row locking and foreign keys. See Chapter 15, *The InnoDB Storage Engine*.
MEMORY	An alias for HEAP. (Actually, as of MySQL 4.1, MEMORY is the preferred term.)
MERGE	A collection of MyISAM tables used as one table. See Section 14.2, “The MERGE Storage Engine”.
MRG_MyISAM	An alias for MERGE.
MyISAM	The binary portable storage engine that is the improved replacement for ISAM. See Section 14.1, “The MyISAM Storage Engine”.
NDB	Alias for NDBCLUSTER.
NDBCLUSTER	Clustered, fault-tolerant, memory-based tables. See Chapter 16, *MySQL Cluster*.

See Chapter 14, *MySQL Storage Engines and Table Types*.

If a storage engine is specified that is not available, MySQL uses MyISAM instead. For example, if a table definition includes the ENGINE=BDB option but the MySQL server does not support BDB tables, the table is created as a MyISAM table. This makes it possible to have a replication setup where you have transactional tables on the master but tables created on the slave are non-transactional (to get more speed). In MySQL 4.1.1, a warning occurs if the storage engine specification is not honored.

The other table options are used to optimize the behavior of the table. In most cases, you don't have to specify any of them. The options work for all storage engines unless otherwise indicated:

- AUTO_INCREMENT

  The initial AUTO_INCREMENT value for the table. This works for MyISAM only. To set the first auto-increment value for an InnoDB table, insert a dummy row with a value one less than the desired value after creating the table, and then delete the dummy row.

  In MySQL 5.0.3 or later, the initial AUTO_INCREMENT value for the table works also for the InnoDB table.

- AVG_ROW_LENGTH

  An approximation of the average row length for your table. You need to set this only for large tables with variable-size records.

  When you create a MyISAM table, MySQL uses the product of the MAX_ROWS and AVG_ROW_LENGTH options to decide how big the resulting table is. If you don't specify either option, the maximum size for a table is 4GB (or 2GB if your operating system only supports 2GB tables). The reason for this is just to keep down the pointer sizes to make the index smaller and faster if you don't really need big files. If you want all your tables to be able to grow above the 4GB limit and are willing to have your smaller tables slightly slower and larger than necessary, you may increase the default pointer size by setting the myisam_data_pointer_size system variable,

which was added in MySQL 4.1.2. See Section 5.2.3, “Server System Variables”.

- `CHECKSUM`

  Set this to 1 if you want MySQL to maintain a live checksum for all rows (that is, a checksum that MySQL updates automatically as the table changes). This makes the table a little slower to update, but also makes it easier to find corrupted tables. The `CHECKSUM TABLE` statement reports the checksum. (`MyISAM` only.)

- `COMMENT`

  A comment for your table, up to 60 characters long.

- `MAX_ROWS`

  The maximum number of rows you plan to store in the table.

- `MIN_ROWS`

  The minimum number of rows you plan to store in the table.

- `PACK_KEYS`

  Set this option to 1 if you want to have smaller indexes. This usually makes updates slower and reads faster. Setting the option to 0 disables all packing of keys. Setting it to `DEFAULT` (MySQL 4.0) tells the storage engine to only pack long `CHAR`/`VARCHAR` columns. (`MyISAM` and `ISAM` only.)

  If you don't use `PACK_KEYS`, the default is to only pack strings, not numbers. If you use `PACK_KEYS=1`, numbers are packed as well.

  When packing binary number keys, MySQL uses prefix compression:

  - Every key needs one extra byte to indicate how many bytes of the previous key are the same for the next key.
  - The pointer to the row is stored in high-byte-first order directly after the key, to improve compression.

  This means that if you have many equal keys on two consecutive rows, all following ``same'' keys usually only take two bytes (including the pointer to the row). Compare this to the ordinary case where the following keys takes `storage_size_for_key + pointer_size` (where the pointer size is usually 4). Conversely, you get a big benefit from prefix compression only if you have many numbers that are the same. If all keys are totally different, you use one byte more per key, if the key isn't a key that can have `NULL` values. (In this case, the packed key length is stored in the same byte that is used to mark if a key is `NULL`.)

- `PASSWORD`

  Encrypt the `.frm` file with a password. This option doesn't do anything in the standard MySQL version.

- `DELAY_KEY_WRITE`

  Set this to 1 if you want to delay key updates for the table until the table is closed. (`MyISAM` only.)

- `ROW_FORMAT`

  Defines how the rows should be stored. Currently this option works only with `MyISAM` tables. The

option value can FIXED or DYNAMIC for static or variable-length row format. **myisampack** sets the type to COMPRESSED. See Section 14.1.3, "MyISAM Table Storage Formats".

Starting with MySQL/InnoDB-5.0.3, InnoDB records are stored in a more compact format (ROW_FORMAT=COMPACT) by default. The old format can be requested by specifying ROW_FORMAT=REDUNDANT.

- RAID_TYPE

  Note: This information applies only before MySQL 5.0. RAID support has been removed as of MySQL 5.0.

  The RAID_TYPE option can help you to exceed the 2GB/4GB limit for the MyISAM data file (not the index file) on operating systems that don't support big files. This option is unnecessary and not recommended for filesystems that support big files.

  You can get more speed from the I/O bottleneck by putting RAID directories on different physical disks. The only allowed RAID_TYPE is STRIPED. 1 and RAID0 are aliases for STRIPED.

  If you specify the RAID_TYPE option for a MyISAM table, specify the RAID_CHUNKS and RAID_CHUNKSIZE options as well. The maximum RAID_CHUNKS value is 255. MyISAM creates RAID_CHUNKS subdirectories named 00, 01, 02, ... 09, 0a, 0b, ... in the database directory. In each of these directories, MyISAM creates a file *tbl_name*.MYD. When writing data to the data file, the RAID handler maps the first RAID_CHUNKSIZE*1024 bytes to the first file, the next RAID_CHUNKSIZE*1024 bytes to the next file, and so on.

  RAID_TYPE works on any operating system, as long as you have built MySQL with the --with-raid option to **configure**. To determine whether a server supports RAID tables, use SHOW VARIABLES LIKE 'have_raid' to see whether the variable value is YES.

- UNION

  UNION is used when you want to use a collection of identical tables as one. This works only with MERGE tables. See Section 14.2, "The MERGE Storage Engine".

  For the moment, you must have SELECT, UPDATE, and DELETE privileges for the tables you map to a MERGE table. Originally, all used tables had to be in the same database as the MERGE table itself. This restriction has been lifted as of MySQL 4.1.1.

- INSERT_METHOD

  If you want to insert data in a MERGE table, you have to specify with INSERT_METHOD into which table the row should be inserted. INSERT_METHOD is an option useful for MERGE tables only. This option was introduced in MySQL 4.0.0. See Section 14.2, "The MERGE Storage Engine".

- DATA DIRECTORY , INDEX DIRECTORY

  By using DATA DIRECTORY='*directory*' or INDEX DIRECTORY='*directory*' you can specify where the MyISAM storage engine should put a table's data file and index file. Note that the directory should be a full path to the directory (not a relative path).

  These options work only for MyISAM tables from MySQL 4.0 on, when you are not using the --skip-symbolic-links option. Your operating system must also have a working, thread-safe realpath() call. See Section 7.6.1.2, "Using Symbolic Links for Tables on Unix".

As of MySQL 3.23, you can create one table from another by adding a SELECT statement at the end of the CREATE TABLE statement:

```
CREATE TABLE new_tbl SELECT * FROM orig_tbl;
```

MySQL creates new columns for all elements in the `SELECT`. For example:

```
mysql> CREATE TABLE test (a INT NOT NULL AUTO_INCREMENT,
    ->        PRIMARY KEY (a), KEY(b))
    ->        TYPE=MyISAM SELECT b,c FROM test2;
```

This creates a `MyISAM` table with three columns, `a`, `b`, and `c`. Notice that the columns from the `SELECT` statement are appended to the right side of the table, not overlapped onto it. Take the following example:

```
mysql> SELECT * FROM foo;
+---+
| n |
+---+
| 1 |
+---+

mysql> CREATE TABLE bar (m INT) SELECT n FROM foo;
Query OK, 1 row affected (0.02 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM bar;
+------+---+
| m    | n |
+------+---+
| NULL | 1 |
+------+---+
1 row in set (0.00 sec)
```

For each row in table `foo`, a row is inserted in `bar` with the values from `foo` and default values for the new columns.

If any errors occur while copying the data to the table, it is automatically dropped and not created.

`CREATE TABLE ... SELECT` does not automatically create any indexes for you. This is done intentionally to make the statement as flexible as possible. If you want to have indexes in the created table, you should specify these before the `SELECT` statement:

```
mysql> CREATE TABLE bar (UNIQUE (n)) SELECT n FROM foo;
```

Some conversion of column types might occur. For example, the `AUTO_INCREMENT` attribute is not preserved, and `VARCHAR` columns can become `CHAR` columns.

When creating a table with `CREATE ... SELECT`, make sure to alias any function calls or expressions in the query. If you do not, the `CREATE` statement might fail or result in undesirable column names.

```
CREATE TABLE artists_and_works
SELECT artist.name, COUNT(work.artist_id) AS number_of_works
FROM artist LEFT JOIN work ON artist.id = work.artist_id
GROUP BY artist.id;
```

As of MySQL 4.1, you can explicitly specify the type for a generated column:

```
CREATE TABLE foo (a TINYINT NOT NULL) SELECT b+1 AS a FROM bar;
```

In MySQL 4.1, you can also use LIKE to create an empty table based on the definition of another table, including any column attributes and indexes the original table has:

```
CREATE TABLE new_tbl LIKE orig_tbl;
```

CREATE TABLE ... LIKE does not copy any DATA DIRECTORY or INDEX DIRECTORY table options that were specified for the original table, or any foreign key definitions.

You can precede the SELECT by IGNORE or REPLACE to indicate how to handle records that duplicate unique key values. With IGNORE, new records that duplicate an existing record on a unique key value are discarded. With REPLACE, new records replace records that have the same unique key value. If neither IGNORE nor REPLACE is specified, duplicate unique key values result in an error.

To ensure that the update log/binary log can be used to re-create the original tables, MySQL does not allow concurrent inserts during CREATE TABLE ... SELECT.

13.2.6.1. Silent Column Specification Changes

In some cases, MySQL silently changes column specifications from those given in a CREATE TABLE or ALTER TABLE statement:

- VARCHAR columns with a length less than four are changed to CHAR.
- If any column in a table has a variable length, the entire row becomes variable-length as a result. Therefore, if a table contains any variable-length columns (VARCHAR, TEXT, or BLOB), all CHAR columns longer than three characters are changed to VARCHAR columns. This doesn't affect how you use the columns in any way; in MySQL, VARCHAR is just a different way to store characters. MySQL performs this conversion because it saves space and makes table operations faster. See Chapter 14, *MySQL Storage Engines and Table Types*.
- From MySQL 4.1.0 to MySQL 5.0.2, a CHAR or VARCHAR column with a length specification greater than 255 is converted to the smallest TEXT type that can hold values of the given length. For example, VARCHAR(500) is converted to TEXT, and VARCHAR(200000) is converted to MEDIUMTEXT. Note that this conversion results in a change in behavior with regard to treatment of trailing spaces.
- TIMESTAMP display sizes are discarded from MySQL 4.1 on, due to changes made to the TIMESTAMP column type in that version. Before MySQL 4.1, TIMESTAMP display sizes must be even and in the range from 2 to 14. If you specify a display size of 0 or greater than 14, the size is coerced to 14. Odd-valued sizes in the range from 1 to 13 are coerced to the next higher even number.
- Before MySQL 4.1.6, you cannot store a literal NULL in a TIMESTAMP column; setting it to NULL sets it to the current date and time. Because TIMESTAMP columns behave this way, the NULL and NOT NULL attributes do not apply in the normal way and are ignored if you specify them. DESCRIBE *tbl_name* always reports that a TIMESTAMP column can be assigned NULL values.
- Columns that are part of a PRIMARY KEY are made NOT NULL even if not declared that way.
- Starting from MySQL 3.23.51, trailing spaces are automatically deleted from ENUM and SET member values when the table is created.
- MySQL maps certain column types used by other SQL database vendors to MySQL types. See Section 11.7, “Using Column Types from Other Database Engines”.
- If you include a USING clause to specify an index type that is not legal for a storage engine, but there is another index type available that the engine can use without affecting query results, the en-

gine uses the available type.

To see whether MySQL used a column type other than the one you specified, issue a DESCRIBE or SHOW CREATE TABLE statement after creating or altering your table.

Certain other column type changes can occur if you compress a table using **myisampack**. See Section 14.1.3.3, “Compressed Table Characteristics”.

13.2.7. CREATE VIEW Syntax

```
CREATE [OR REPLACE] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

This statement creates a new view, or replaces an existing one if the OR REPLACE clause is given. The *select_statement* is a SELECT statement that provides the definition of the view. The statement can select from base tables or other views.

This statement requires the CREATE VIEW privilege for the view, and some privilege for each column referred to in the SELECT statement. If the OR REPLACE clause is present, you must also have the DELETE privilege for the view.

A view belongs to a database. By default, a new view is created in the current database. To create the view explicitly in a given database, specify the name as *db_name.view_name* when you create it.

```
mysql> CREATE VIEW test.v AS SELECT * FROM t;
```

Tables and views share the same namespace within a database, so a database cannot contain a table and a view that have the same name.

Views must have unique column names with no duplicates, just like base tables. By default, the names of the columns retrieved by the SELECT statement are used for the view column names. To define explicit names for the view columns, the optional *column_list* clause can be given as a list of comma-separated identifiers. The number of names in *column_list* must be the same as the number of columns retrieved by the SELECT statement.

Columns retrieved by the SELECT statement can be simple references to table columns. They can also be expressions that use functions, constant values, operators, and so forth.

Unqualified table or view names in the SELECT statement are interpreted with respect to the default database. A view can refer to tables or views in other databases by qualifying the table or view name with the proper database name.

A view can be created from many kinds of SELECT statements. It can refer to base tables or other views. It can use joins, UNION, and subqueries. The SELECT need not even refer to any tables. The following example defines a view that selects two columns from another table, as well as an expression calculated from those columns:

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
```

```
|    3 |    50 |   150 |
+------+-------+-------+
```

A view definition is subject to the following restrictions:

- The `SELECT` statement cannot contain a subquery in the `FROM` clause.
- The `SELECT` statement cannot refer to system or user variables.
- Within a stored procedure, the definition cannot refer to procedure parameters or local variables.
- Any table or view referred to in the definition must exist. However, after a view has been created, it is possible to drop a table or view that the definition refers to. To check a view definition for problems of this kind, use the `CHECK TABLE` statement.
- The definition cannot refer to a `TEMPORARY` table, and you cannot create a `TEMPORARY` view.
- You cannot associate a trigger with a view.

`ORDER BY` is allowed in a view definition, but it is ignored if you select from a view using a statement that has its own `ORDER BY`.

For other options or clauses in the definition, they are added to the options or clauses of the statement that references the view, but the effect is undefined. For example, if a view definition includes a `LIMIT` clause, and you select from the view using a statement that has its own `LIMIT` clause, it is undefined which limit applies. This same principle applies to options such as `ALL`, `DISTINCT`, or `SQL_SMALL_RESULT` that follow the `SELECT` keyword, and to clauses such as `INTO`, `FOR UPDATE`, `LOCK IN SHARE MODE`, and `PROCEDURE`.

If you create a view and then change the query processing environment by changing system variables, that may affect the results you get from the view:

```
mysql> CREATE VIEW v AS SELECT CHARSET(CHAR(65)), COLLATION(CHAR(65));
Query OK, 0 rows affected (0.00 sec)

mysql> SET NAMES 'latin1';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM v;
+-------------------+----------------------+
| CHARSET(CHAR(65)) | COLLATION(CHAR(65)) |
+-------------------+----------------------+
| latin1            | latin1_swedish_ci    |
+-------------------+----------------------+
1 row in set (0.00 sec)

mysql> SET NAMES 'utf8';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM v;
+-------------------+----------------------+
| CHARSET(CHAR(65)) | COLLATION(CHAR(65)) |
+-------------------+----------------------+
| utf8              | utf8_general_ci      |
+-------------------+----------------------+
1 row in set (0.00 sec)
```

The optional `ALGORITHM` clause is a MySQL extension to standard SQL. `ALGORITHM` takes three val-

ues: MERGE, TEMPTABLE, or UNDEFINED. The default algorithm is UNDEFINED if no ALGORITHM clause is present. The algorithm affects how MySQL processes the view.

For MERGE, the text of a statement that refers to the view and the view definition are merged such that parts of the view definition replace corresponding parts of the statement.

For TEMPTABLE, the results from the view are retrieved into a temporary table, which then is used to execute the statement.

For UNDEFINED, MySQL chooses which algorithm to use. It prefers MERGE over TEMPTABLE if possible, because MERGE is usually more efficient and because a view cannot be updatable if a temporary table is used.

A reason to choose TEMPTABLE explicitly is that locks can be released on underlying tables after the temporary table has been created and before it is used to finish processing the statement. This might result in quicker lock release than the MERGE algorithm so that other clients that use the view are not blocked as long.

A view algorithm can be UNDEFINED three ways:

- No ALGORITHM clause is present in the CREATE VIEW statement.
- The CREATE VIEW statement has an explicit ALGORITHM = UNDEFINED clause.
- ALGORITHM = MERGE is specified for a view that can be processed only with a temporary table. In this case, MySQL generates a warning and sets the algorithm to UNDEFINED.

As mentioned earlier, MERGE is handled by merging corresponding parts of a view definition into the statement that refers to the view. The following examples briefly illustrate how the MERGE algorithm works. The examples assume that there is a view v_merge that has this definition:

```
CREATE ALGORITHM = MERGE VIEW v_merge (vc1, vc2) AS
SELECT c1, c2 FROM t WHERE c3 > 100;
```

Example 1: Suppose that we issue this statement:

```
SELECT * FROM v_merge;
```

MySQL handles the statement as follows:

- v_merge becomes t
- * becomes vc1, vc2, which corresponds to c1, c2
- The view WHERE clause is added

The resulting statement to be executed becomes:

```
SELECT c1, c2 FROM t WHERE c3 > 100;
```

Example 2: Suppose that we issue this statement:

```
SELECT * FROM v_merge WHERE vc1 < 100;
```

This statement is handled similarly to the previous one, except that `vc1 < 100` becomes `c1 < 100` and the view `WHERE` clause is added to the statement `WHERE` clause using an `AND` connective (and parentheses are added to make sure the parts of the clause are executed with correct precedence). The resulting statement to be executed becomes:

```
SELECT c1, c2 FROM t WHERE (c3 > 100) AND (c1 < 100);
```

Effectively, the statement to be executed has a `WHERE` clause of this form:

```
WHERE (select WHERE) AND (view WHERE)
```

The `MERGE` algorithm requires a one-to relationship between the rows in the view and the rows in the underlying table. If this relationship does not hold, a temporary table must be used instead. Lack of a one-to-one relationship occurs if the view contains any of a number of constructs:

- Aggregate functions (`SUM()`, `MIN()`, `MAX()`, `COUNT()`, and so forth)
- `DISTINCT`
- `GROUP BY`
- `HAVING`
- `UNION` or `UNION ALL`
- Refers only to literal values (in this case, there is no underlying table)

Some views are updatable. That is, you can use them in statements such as `UPDATE`, `DELETE`, or `INSERT` to update the contents of the underlying table. For a view to be updatable, there must be a one-to relationship between the rows in the view and the rows in the underlying table. There are also certain other constructs that make a view non-updatable. To be more specific, a view is not updatable if it contains any of the following:

- Aggregate functions (`SUM()`, `MIN()`, `MAX()`, `COUNT()`, and so forth)
- `DISTINCT`
- `GROUP BY`
- `HAVING`
- `UNION` or `UNION ALL`
- Subquery in the select list
- Join
- Non-updatable view in the `FROM` clause
- A subquery in the `WHERE` clause that refers to a table in the `FROM` clause
- Refers only to literal values (in this case, there is no underlying table to update)
- `ALGORITHM = TEMPTABLE` (use of a temporary table always makes a view non-updatable)

With respect to insertability (being updatable with `INSERT` statements), an updatable view is insertable if it also satisfies these additional requirements for the view columns:

- There must be no duplicate view column names.
- The view must contain all columns in the base table that do not have a default value.
- The view columns must be simple column references and not derived columns. A derived column is one that is not a simple column reference but is derived from an expression. These are examples of derived columns:

  ```
  3.14159
  col1 + 3
  UPPER(col2)
  col3 / col4
  (subquery)
  ```

A view that has a mix of simple column references and derived columns is not insertable, but it can be updatable if you update only those columns that are not derived. Consider this view:

```
CREATE VIEW v AS SELECT col1, 1 AS col2 FROM t;
```

This view is not insertable because `col2` is derived from an expression. But it is updatable if the update does not try to update `col2`. This update is allowable:

```
UPDATE v SET col1 = 0;
```

This update is not allowable because it attempts to update a derived column:

```
UPDATE v SET col2 = 0;
```

It is sometimes possible for a multiple-table view to be updatable, assuming that it can be processed with the `MERGE` algorithm. For this to work, the view must use an inner join (not an outer join or a `UNION`). Also, only a single table in the view definition can be updated, so the `SET` clause must name only columns from one of the tables in the view. Views that use `UNION ALL` are disallowed even though they might be theoretically updatable, because the implementation uses temporary tables to process them.

For a multiple-table updatable view, `INSERT` can work if it inserts into a single table. `DELETE` is not supported.

The `WITH CHECK OPTION` clause can be given for an updatable view to prevent inserts or updates to rows except those for which the `WHERE` clause in the *`select_statement`* is true.

In a `WITH CHECK OPTION` clause for an updatable view, the `LOCAL` and `CASCADED` keywords determine the scope of check testing when the view is defined in terms of another view. `LOCAL` keyword restricts the `CHECK OPTION` only to the view being defined. `CASCADED` causes the checks for underlying views to be evaluated as well. When neither keyword is given, the default is `CASCADED`. Consider the definitions for the following table and set of views:

```
mysql> CREATE TABLE t1 (a INT);
mysql> CREATE VIEW v1 AS SELECT * FROM t1 WHERE a < 2
    -> WITH CHECK OPTION;
mysql> CREATE VIEW v2 AS SELECT * FROM v1 WHERE a > 0
    -> WITH LOCAL CHECK OPTION;
```

```
mysql> CREATE VIEW v3 AS SELECT * FROM v1 WHERE a > 0
    -> WITH CASCADED CHECK OPTION;
```

Here the v2 and v3 views are defined in terms of another view, v1. v2 has a LOCAL check option, so inserts are tested only against the v2 check. v3 has a CASCADED check option, so inserts are tested not only against its own check, but against those of underlying views. The following statements illustrate these differences:

```
ql> INSERT INTO v2 VALUES (2);
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO v3 VALUES (2);
ERROR 1369 (HY000): CHECK OPTION failed 'test.v3'
```

The updatability of views may be affected by the value of the updatable_views_with_limit system variable. See Section 5.2.3, “Server System Variables”.

The CREATE VIEW statement was added in MySQL 5.0.1. The WITH CHECK OPTION clause was implemented in MySQL 5.0.2.

INFORMATION_SCHEMA contains a VIEWS table from which information about view objects can be obtained. See Section 21.1.15, “The INFORMATION_SCHEMA VIEWS Table”.

13.2.8. DROP DATABASE Syntax

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

DROP DATABASE drops all tables in the database and deletes the database. Be *very* careful with this statement! To use DROP DATABASE, you need the DROP privilege on the database.

In MySQL 3.22 or later, you can use the keywords IF EXISTS to prevent an error from occurring if the database doesn't exist.

DROP SCHEMA can be used as of MySQL 5.0.2.

If you use DROP DATABASE on a symbolically linked database, both the link and the original database are deleted.

As of MySQL 4.1.2, DROP DATABASE returns the number of tables that were removed. This corresponds to the number of .frm files removed.

The DROP DATABASE statement removes from the given database directory those files and directories that MySQL itself may create during normal operation:

- All files with these extensions:

.BAK	.DAT	.HSH	.ISD
.ISM	.ISM	.MRG	.MYD
.MYI	.db	.frm	

- All subdirectories with names that consist of two hex digits 00-ff. These are subdirectories used for RAID tables. (These directories are not removed as of MySQL 5.0, when support for RAID tables was removed. You should convert your RAID tables and remove these directories manually before upgrading to MySQL 5.0. See Section 2.10.1, “Upgrading from Version 4.1 to 5.0”.)

- The `db.opt` file, if it exists.

If other files or directories remain in the database directory after MySQL removes those just listed, the database directory cannot be removed. In this case, you must remove any remaining files or directories manually and issue the `DROP DATABASE` statement again.

You can also drop databases with **mysqladmin**. See Section 8.4, "**mysqladmin**, Administering a MySQL Server".

13.2.9. DROP INDEX Syntax

```
DROP INDEX index_name ON tbl_name
```

`DROP INDEX` drops the index named *`index_name`* from the table *`tbl_name`*. In MySQL 3.22 or later, `DROP INDEX` is mapped to an `ALTER TABLE` statement to drop the index. See Section 13.2.2, "`ALTER TABLE` Syntax". `DROP INDEX` doesn't do anything prior to MySQL 3.22.

13.2.10. DROP TABLE Syntax

```
DROP [TEMPORARY] TABLE [IF EXISTS]
    tbl_name [, tbl_name] ...
    [RESTRICT | CASCADE]
```

`DROP TABLE` removes one or more tables. You must have the `DROP` privilege for each table. All table data and the table definition are *removed*, so *be careful* with this statement!

In MySQL 3.22 or later, you can use the keywords `IF EXISTS` to prevent an error from occurring for tables that don't exist. As of MySQL 4.1, a `NOTE` is generated for each non-existent table when using `IF EXISTS`. See Section 13.5.4.20, "`SHOW WARNINGS` Syntax".

`RESTRICT` and `CASCADE` are allowed to make porting easier. For the moment, they do nothing.

Note: `DROP TABLE` automatically commits the current active transaction, unless you are using MySQL 4.1 or higher and the `TEMPORARY` keyword.

The `TEMPORARY` keyword is ignored in MySQL 4.0. As of 4.1, it has the following effect:

- The statement drops only `TEMPORARY` tables.
- The statement doesn't end a running transaction.
- No access rights are checked. (A `TEMPORARY` table is visible only to the client that created it, so no check is necessary.)

Using `TEMPORARY` is a good way to ensure that you don't accidentally drop a non-`TEMPORARY` table.

13.2.11. DROP VIEW Syntax

```
DROP VIEW [IF EXISTS]
    view_name [, view_name] ...
    [RESTRICT | CASCADE]
```

`DROP VIEW` removes one or more views. You must have the `DROP` privilege for each view.

You can use the keywords `IF EXISTS` to prevent an error from occurring for views that don't exist. When this clause is given, a `NOTE` is generated for each non-existent view. See Section 13.5.4.20, "`SHOW WARNINGS` Syntax".

`RESTRICT` and `CASCADE`, if given, are parsed and ignored.

This statement was added in MySQL 5.0.1.

13.2.12. RENAME TABLE Syntax

```
RENAME TABLE tbl_name TO new_tbl_name
    [, tbl_name2 TO new_tbl_name2] ...
```

This statement renames one or more tables. It was added in MySQL 3.23.23.

The rename operation is done atomically, which means that no other thread can access any of the tables while the rename is running. For example, if you have an existing table `old_table`, you can create another table `new_table` that has the same structure but is empty, and then replace the existing table with the empty one as follows:

```
CREATE TABLE new_table (...);
RENAME TABLE old_table TO backup_table, new_table TO old_table;
```

If the statement renames more than one table, renaming operations are done from left to right. If you want to swap two table names, you can do so like this (assuming that no table named `tmp_table` currently exists):

```
RENAME TABLE old_table TO tmp_table,
             new_table TO old_table,
             tmp_table TO new_table;
```

As long as two databases are on the same filesystem you can also rename a table to move it from one database to another:

```
RENAME TABLE current_db.tbl_name TO other_db.tbl_name;
```

When you execute `RENAME`, you can't have any locked tables or active transactions. You must also have the `ALTER` and `DROP` privileges on the original table, and the `CREATE` and `INSERT` privileges on the new table.

If MySQL encounters any errors in a multiple-table rename, it does a reverse rename for all renamed tables to get everything back to the original state.

13.3. MySQL Utility Statements

13.3.1. DESCRIBE Syntax (Get Information About Columns)

```
{DESCRIBE | DESC} tbl_name [col_name | wild]
```

DESCRIBE provides information about the columns in a table. It is a shortcut for SHOW COLUMNS FROM. As of MySQL 5.0.1, these statements also display information for views.

See Section 13.5.4.3, “SHOW COLUMNS Syntax”.

col_name can be a column name, or a string containing the SQL '%' and '_' wildcard characters to obtain output only for the columns with names matching the string. There is no need to enclose the string in quotes unless it contains spaces or other special characters.

```
mysql> DESCRIBE city;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| Id         | int(11)  |      | PRI | NULL    | auto_increment |
| Name       | char(35) |      |     |         |                |
| Country    | char(3)  |      | UNI |         |                |
| District   | char(20) | YES  | MUL |         |                |
| Population | int(11)  |      |     | 0       |                |
+------------+----------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

The Null column indicates whether NULL values can be stored, with YES displayed when NULL values are allowed.

The Key column indicates whether the field is indexed. A value of PRI indicates that the field is part of the table's primary key. UNI indicates that the field is part of a UNIQUE index. The MUL value indicates that multiple occurrences of a given value allowed within the field.

A field can be designated as MUL even if a UNIQUE index is used if NULL values are allowed, as multiple rows in a UNIQUE index can hold a NULL value if the column is not declared NOT NULL. Another cause for MUL on a UNIQUE index is when two columns form a composite UNIQUE index; while the combination of the columns is unique, each column can still hold multiple occurences of a given value. Note that in a composite index only the leftmost field of the index has an entry in the Key column.

The Default column indicates the default value that is assigned to the field.

The Extra column contains any additional information that is available about a given field. In our example the Extra column indicates that our Id column was created with the AUTO_INCREMENT keyword.

If the column types are different from what you expect them to be based on a CREATE TABLE statement, note that MySQL sometimes changes column types. See Section 13.2.6.1, “Silent Column Specification Changes”.

The DESCRIBE statement is provided for Oracle compatibility.

The SHOW CREATE TABLE and SHOW TABLE STATUS statements also provide information about tables. See Section 13.5.4, “SHOW Syntax”.

13.3.2. USE Syntax

```
USE db_name
```

The USE *db_name* statement tells MySQL to use the *db_name* database as the default (current) database for subsequent statements. The database remains the default until the end of the session or until another USE statement is issued:

```
mysql> USE db1;
mysql> SELECT COUNT(*) FROM mytable;   # selects from db1.mytable
mysql> USE db2;
mysql> SELECT COUNT(*) FROM mytable;   # selects from db2.mytable
```

Making a particular database current by means of the `USE` statement does not preclude you from accessing tables in other databases. The following example accesses the `author` table from the `db1` database and the `editor` table from the `db2` database:

```
mysql> USE db1;
mysql> SELECT author_name,editor_name FROM author,db2.editor
    ->        WHERE author.editor_id = db2.editor.editor_id;
```

The `USE` statement is provided for Sybase compatibility.

13.4. MySQL Transactional and Locking Statements

13.4.1. START TRANSACTION, COMMIT, and ROLLBACK Syntax

By default, MySQL runs with autocommit mode enabled. This means that as soon as you execute a statement that updates (modifies) a table, MySQL stores the update on disk.

If you are using transaction-safe tables (like `InnoDB` or `BDB`), you can disable autocommit mode with the following statement:

```
SET AUTOCOMMIT=0;
```

After disabling autocommit mode by setting the `AUTOCOMMIT` variable to zero, you must use `COMMIT` to store your changes to disk or `ROLLBACK` if you want to ignore the changes you have made since the beginning of your transaction.

If you want to disable autocommit mode for a single series of statements, you can use the `START TRANSACTION` statement:

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

With `START TRANSACTION`, autocommit remains disabled until you end the transaction with `COMMIT` or `ROLLBACK`. The autocommit mode then reverts to its previous state.

`BEGIN` and `BEGIN WORK` can be used instead of `START TRANSACTION` to initiate a transaction. `START TRANSACTION` was added in MySQL 4.0.11. This is standard SQL syntax and is the recommended way to start an ad-hoc transaction. `BEGIN` and `BEGIN WORK` are available from MySQL 3.23.17 and 3.23.19, respectively.

As of MySQL 4.1.8, you can begin a transaction like this:

```
START TRANSACTION WITH CONSISTENT SNAPSHOT;
```

The WITH CONSISTENT SNAPSHOT clause starts a consistent read for storage engines that are capable of it. Currently, this applies only to InnoDB. The effect is the same as issuing a START TRANSACTION followed by a SELECT from any InnoDB table. See Section 15.11.3, "Consistent Non-Locking Read".

Beginning a transaction causes an implicit UNLOCK TABLES to be performed.

Note that if you are not using transaction-safe tables, any changes are stored at once, regardless of the status of autocommit mode.

If you issue a ROLLBACK statement after updating a non-transactional table within a transaction, an ER_WARNING_NOT_COMPLETE_ROLLBACK warning occurs. Changes to transaction-safe tables are rolled back, but not changes to non-transaction-safe tables.

If you are using START TRANSACTION or SET AUTOCOMMIT=0, you should use the MySQL binary log for backups instead of the older update log. Transactions are stored in the binary log in one chunk, upon COMMIT. Transactions that are rolled back are not logged. (Exception: Modifications to non-transactional tables cannot be rolled back. If a transaction that is rolled back includes modifications to non-transactional tables, the entire transaction is logged with a ROLLBACK statement at the end to ensure that the modifications to those tables are replicated. This is true as of MySQL 4.0.15.) See Section 5.9.4, "The Binary Log".

You can change the isolation level for transactions with SET TRANSACTION ISOLATION LEVEL. See Section 13.4.6, "SET TRANSACTION Syntax".

Rolling back can be a slow operation that can occur without the user having explicitly asked for it (for example, when an error occurs). Because of this, SHOW PROCESSLIST displays Rolling back in the State column for the connection during implicit rollback and explicit (ROLLBACK SQL command) rollbacks, starting from MySQL 4.1.8.

13.4.2. Statements That Cannot Be Rolled Back

Some statements cannot be rolled back. In general, these include data definition language (DDL) statements, such as those that create or drop databases, or those that create, drop, or alter tables.

You should design your transactions not to include such statements. If you issue a statement early in a transaction that cannot be rolled back, and then another statement later fails, the full effect of the transaction cannot be rolled back by issuing a ROLLBACK statement.

13.4.3. Statements That Cause an Implicit Commit

Each of the following statements (and any synonyms for them) implicitly end a transaction, as if you had done a COMMIT before executing the statement:

ALTER TABLE	BEGIN	CREATE INDEX
DROP DATABASE	DROP INDEX	DROP TABLE
LOAD MASTER DATA	LOCK TABLES	RENAME TABLE
SET AUTOCOMMIT=1	START TRANSACTION	TRUNCATE TABLE

UNLOCK TABLES also ends a transaction if any tables currently are locked. Prior to MySQL 4.0.13, CREATE TABLE ends a transaction if the binary update log is enabled.

Transactions cannot be nested. This is a consequence of the implicit COMMIT performed for any current transaction when you issue a START TRANSACTION statement or one of its synonyms.

13.4.4. SAVEPOINT and ROLLBACK TO SAVEPOINT Syntax

```
SAVEPOINT identifier
ROLLBACK TO SAVEPOINT identifier
```

Starting from MySQL 4.0.14 and 4.1.1, InnoDB supports the SQL statements SAVEPOINT and ROLLBACK TO SAVEPOINT.

The SAVEPOINT statement sets a named transaction savepoint with a name of identifier. If the current transaction has a savepoint with the same name, the old savepoint is deleted and a new one is set.

The ROLLBACK TO SAVEPOINT statement rolls back a transaction to the named savepoint. Modifications that the current transaction made to rows after the savepoint was set are undone in the rollback, but InnoDB does *not* release the row locks that were stored in memory after the savepoint. (Note that for a new inserted row, the lock information is carried by the transaction ID stored in the row; the lock is not separately stored in memory. In this case, the row lock is released in the undo.) Savepoints that were set at a later time than the named savepoint are deleted.

If the statement returns the following error, it means that no savepoint with the specified name exists:

```
ERROR 1181: Got error 153 during ROLLBACK
```

All savepoints of the current transaction are deleted if you execute a COMMIT, or a ROLLBACK that does not name a savepoint.

13.4.5. LOCK TABLES and UNLOCK TABLES Syntax

```
LOCK TABLES
    tbl_name [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE}
    [, tbl_name [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE}] ...
UNLOCK TABLES
```

LOCK TABLES locks tables for the current thread. UNLOCK TABLES releases any locks held by the current thread. All tables that are locked by the current thread are implicitly unlocked when the thread issues another LOCK TABLES, or when the connection to the server is closed.

Note the following regarding the use of LOCK TABLES with transactional tables:

- LOCK TABLES is not transaction-safe and implicitly commits any active transactions before attempting to lock the tables. Also, beginning a transaction (for example, with START TRANSACTION) implicitly performs an UNLOCK TABLES.
- The correct way to use LOCK TABLES with transactional tables, like InnoDB, is to set AUTOCOMMIT = 0 and not to call UNLOCK TABLES until you commit the transaction explicitly. When you call LOCK TABLES, InnoDB internally takes its own table lock, and MySQL takes its own table lock. InnoDB releases its table lock at the next commit, but for MySQL to release its table lock, you have to call UNLOCK TABLES. You should not have AUTOCOMMIT = 1, because then InnoDB releases its table lock immediately after the call of LOCK TABLES, and deadlocks can very easily happen. Starting from 4.1.9, we do not acquire the InnoDB table lock at all if AUTOCOMMIT=1. That helps old applications to avoid unnecessary deadlocks.
- ROLLBACK does not release MySQL's non-transactional table locks.

As of MySQL 4.0.2, to use LOCK TABLES you must have the LOCK TABLES privilege and a SELECT privilege for the involved tables. In MySQL 3.23, you must have SELECT, INSERT, DELETE, and UPDATE privileges for the tables.

The main reasons to use LOCK TABLES are for emulating transactions or to get more speed when updating tables. This is explained in more detail later.

If a thread obtains a READ lock on a table, that thread (and all other threads) can only read from the table. If a thread obtains a WRITE lock on a table, only the thread holding the lock can write to the table. Other threads are blocked from doing so until the lock has been released.

The difference between READ LOCAL and READ is that READ LOCAL allows non-conflicting INSERT statements (concurrent inserts) to execute while the lock is held. However, this can't be used if you are going to manipulate the database files outside MySQL while you hold the lock. For InnoDB, READ LOCAL essentially does nothing: it does not lock the table at all. The use of READ LOCAL for InnoDB tables is deprecated, because for InnoDB, a plain consistent read SELECT does the same thing, and no locks are needed.

When you use LOCK TABLES, you must lock all tables that you are going to use in your queries. While the locks obtained with a LOCK TABLES statement are in effect, you cannot access any tables that were not locked by the statement. Also, you cannot use a locked table multiple times in one query - use aliases for that. Note that in that case you must get a lock for each alias separately.

```
mysql> LOCK TABLE t WRITE, t AS t1 WRITE;
mysql> INSERT INTO t SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> INSERT INTO t SELECT * FROM t AS t1;
```

If your queries refer to a table using an alias, then you must lock the table using that same alias. It does not work to lock the table without specifying the alias:

```
mysql> LOCK TABLE t READ;
mysql> SELECT * FROM t AS myalias;
ERROR 1100: Table 'myalias' was not locked with LOCK TABLES
```

Conversely, if you lock a table using an alias, you must refer to it in your queries using that alias:

```
mysql> LOCK TABLE t AS myalias READ;
mysql> SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> SELECT * FROM t AS myalias;
```

WRITE locks normally have higher priority than READ locks to ensure that updates are processed as soon as possible. This means that if one thread obtains a READ lock and then another thread requests a WRITE lock, subsequent READ lock requests wait until the WRITE thread has gotten the lock and released it. You can use LOW_PRIORITY WRITE locks to allow other threads to obtain READ locks while the thread is waiting for the WRITE lock. You should use LOW_PRIORITY WRITE locks only if you are sure that there is eventually a time when no threads have a READ lock.

LOCK TABLES works as follows:

1. Sort all tables to be locked in an internally defined order. From the user standpoint, this order is undefined.

2. If a table is locked with a read and a write lock, put the write lock before the read lock.

3. Lock one table at a time until the thread gets all locks.

This policy ensures that table locking is deadlock free. There are, however, other things you need to be aware of about this policy:

If you are using a `LOW_PRIORITY WRITE` lock for a table, it means only that MySQL waits for this particular lock until there are no threads that want a `READ` lock. When the thread has gotten the `WRITE` lock and is waiting to get the lock for the next table in the lock table list, all other threads wait for the `WRITE` lock to be released. If this becomes a serious problem with your application, you should consider converting some of your tables to transaction-safe tables.

You can safely use `KILL` to terminate a thread that is waiting for a table lock. See Section 13.5.5.3, "`KILL` Syntax".

Note that you should *not* lock any tables that you are using with `INSERT DELAYED` because in that case the `INSERT` is done by a separate thread.

Normally, you don't have to lock tables, because all single `UPDATE` statements are atomic; no other thread can interfere with any other currently executing SQL statement. There are a few cases when you would like to lock tables anyway:

- If you are going to run many operations on a set of `MyISAM` tables, it's much faster to lock the tables you are going to use. Locking `MyISAM` tables speeds up inserting, updating, or deleting on them. The downside is that no thread can update a `READ`-locked table (including the one holding the lock) and no thread can access a `WRITE`-locked table other than the one holding the lock.

  The reason some `MyISAM` operations are faster under `LOCK TABLES` is that MySQL does not flush the key cache for the locked tables until `UNLOCK TABLES` is called. Normally, the key cache is flushed after each SQL statement.

- If you are using a storage engine in MySQL that doesn't support transactions, you must use `LOCK TABLES` if you want to ensure that no other thread comes between a `SELECT` and an `UPDATE`. The example shown here requires `LOCK TABLES` to execute safely:

  ```
  mysql> LOCK TABLES trans READ, customer WRITE;
  mysql> SELECT SUM(value) FROM trans WHERE customer_id=some_id;
  mysql> UPDATE customer
      ->     SET total_value=sum_from_previous_statement
      ->     WHERE customer_id=some_id;
  mysql> UNLOCK TABLES;
  ```

  Without `LOCK TABLES`, it is possible that another thread might insert a new row in the `trans` table between execution of the `SELECT` and `UPDATE` statements.

You can avoid using `LOCK TABLES` in many cases by using relative updates (`UPDATE customer SET value=value+new_value`) or the `LAST_INSERT_ID()` function, See Section 1.5.5.3, "Transactions and Atomic Operations".

You can also avoid locking tables in some cases by using the user-level advisory lock functions `GET_LOCK()` and `RELEASE_LOCK()`. These locks are saved in a hash table in the server and implemented with `pthread_mutex_lock()` and `pthread_mutex_unlock()` for high speed. See Section 12.8.4, "Miscellaneous Functions".

See Section 7.3.1, "Locking Methods", for more information on locking policy.

You can lock all tables in all databases with read locks with the `FLUSH TABLES WITH READ LOCK`

statement. See Section 13.5.5.2, "FLUSH Syntax". This is a very convenient way to get backups if you have a filesystem such as Veritas that can take snapshots in time.

Note: If you use ALTER TABLE on a locked table, it may become unlocked. See Section A.7.1, "Problems with ALTER TABLE".

13.4.6. SET TRANSACTION Syntax

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE }
```

This statement sets the transaction isolation level for the next transaction, globally, or for the current session.

The default behavior of SET TRANSACTION is to set the isolation level for the next (not yet started) transaction. If you use the GLOBAL keyword, the statement sets the default transaction level globally for all new connections created from that point on. Existing connections are unaffected. You need the SUPER privilege to do this. Using the SESSION keyword sets the default transaction level for all future transactions performed on the current connection.

For descriptions of each InnoDB transaction isolation level, see Section 15.11.2, "InnoDB and TRANSACTION ISOLATION LEVEL". InnoDB supports each of these levels from MySQL 4.0.5 on. The default level is REPEATABLE READ.

You can set the initial default global isolation level for **mysqld** with the --transaction-isolation option. See Section 5.2.1, "**mysqld** Command-Line Options".

13.5. Database Administration Statements

13.5.1. Account Management Statements

13.5.1.1. CREATE USER Syntax

```
CREATE USER user [IDENTIFIED BY [PASSWORD] 'password']
    [, user [IDENTIFIED BY [PASSWORD] 'password']] ...
```

The CREATE USER statement creates new MySQL accounts. To use it, you must have the GRANT OPTION privilege for the mysql database. For each account, CREATE USER creates a new record in the mysql.user table that has no privileges. An error occurs if the account already exists. The account can be given a password with the optional IDENTIFIED BY clause. The *user* value and the password are given the same way as for the GRANT statement.

The CREATE USER statement was added in MySQL 5.0.2.

13.5.1.2. DROP USER Syntax

```
DROP USER user [, user] ...
```

The DROP USER statement deletes one or more MySQL accounts. To use it, you must have the GRANT OPTION privilege for the mysql database. Each account is named using the same format as for GRANT or REVOKE; for example, 'jeffrey'@'localhost'. The user and host parts of the account name correspond to the User and Host column values of the user table record for the account.

DROP USER was added in MySQL 4.1.1 and originally removed only accounts that have no privileges.

In MySQL 5.0.2, it was modified to also remove account privileges. This means that the procedure for removing an account depends on your version of MySQL.

As of MySQL 5.0.2, remove an account and its privileges as follows:

```
DROP USER user;
```

The statement removes privilege records for the account from all grant tables.

From MySQL 4.1.1 to 5.0.1, `DROP USER` deletes only MySQL accounts that don't have any privileges. It serves to remove each account record from the `user` table. To remove a MySQL account, you should use the following procedure, performing the steps in the order shown:

1. Use `SHOW GRANTS` to determine what privileges the account has. See Section 13.5.4.10, "`SHOW GRANTS` Syntax".
2. Use `REVOKE` to revoke the privileges displayed by `SHOW GRANTS`. This removes records for the account from all the grant tables except the `user` table, and revokes any global privileges listed in the `user` table. See Section 13.5.1.3, "`GRANT` and `REVOKE` Syntax".
3. Delete the account by using `DROP USER` to remove the `user` table record.

Before MySQL 4.1.1, `DROP USER` is not available. You should first revoke the account privileges as just described. Then delete the `user` table record and flush the grant tables like this:

```
mysql> DELETE FROM mysql.user
    -> WHERE User='user_name' and Host='host_name';
mysql> FLUSH PRIVILEGES;
```

13.5.1.3. GRANT and REVOKE Syntax

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)]] ...
    ON {tbl_name | * | *.* | db_name.*}
    TO user [IDENTIFIED BY [PASSWORD] 'password']
        [, user [IDENTIFIED BY [PASSWORD] 'password']] ...
    [REQUIRE
        NONE |
        [{SSL| X509}]
        [CIPHER 'cipher' [AND]]
        [ISSUER 'issuer' [AND]]
        [SUBJECT 'subject']]
    [WITH [GRANT OPTION | MAX_QUERIES_PER_HOUR count |
                          MAX_UPDATES_PER_HOUR count |
                          MAX_CONNECTIONS_PER_HOUR count |
                          MAX_USER_CONNECTIONS count]]


REVOKE priv_type [(column_list)] [, priv_type [(column_list)]] ...
    ON {tbl_name | * | *.* | db_name.*}
    FROM user [, user] ...

REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

The `GRANT` and `REVOKE` statements allow system administrators to create MySQL user accounts and to grant rights to and revoke them from accounts. `GRANT` and `REVOKE` are implemented in MySQL 3.22.11 or later. For earlier MySQL versions, these statements do nothing.

MySQL account information is stored in the tables of the `mysql` database. This database and the access control system are discussed extensively in Chapter 5, *Database Administration*, which you should consult for additional details.

If the grant tables contain privilege records that contain mixed-case database or table names and the `lower_case_table_names` system variable is set, `REVOKE` cannot be used to revoke the privileges. It will be necessary to manipulate the grant tables directly. (`GRANT` will not create such records when `lower_case_table_names` is set, but such records might have been created prior to setting the variable.)

Privileges can be granted at several levels:

- **Global level**

  Global privileges apply to all databases on a given server. These privileges are stored in the `mysql.user` table. `GRANT ALL ON *.*` and `REVOKE ALL ON *.*` grant and revoke only global privileges.

- **Database level**

  Database privileges apply to all objects in a given database. These privileges are stored in the `mysql.db` and `mysql.host` tables. `GRANT ALL ON db_name.*` and `REVOKE ALL ON db_name.*` grant and revoke only database privileges.

- **Table level**

  Table privileges apply to all columns in a given table. These privileges are stored in the `mysql.tables_priv` table. `GRANT ALL ON db_name.tbl_name` and `REVOKE ALL ON db_name.tbl_name` grant and revoke only table privileges.

- **Column level**

  Column privileges apply to single columns in a given table. These privileges are stored in the `mysql.columns_priv` table. When using `REVOKE`, you must specify the same columns that were granted.

- **Routine level**

  The `CREATE ROUTINE`, `ALTER ROUTINE`, `EXECUTE`, and `GRANT` privileges apply to stored routines. They can be granted at the global and database levels. Also, except for `CREATE ROUTINE`, these privileges can be granted at the routine level for individual routines and are stored in the `mysql.procs_priv` table.

To make it easy to revoke all privileges, MySQL 4.1.2 has added the following syntax, which drops all global, database-, table-, and column-level privileges for the named users:

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

Before MySQL 4.1.2, all privileges cannot be dropped at once. Two statements are necessary:

```
REVOKE ALL PRIVILEGES ON *.* FROM user [, user] ...
REVOKE GRANT OPTION ON *.* FROM user [, user] ...
```

For the `GRANT` and `REVOKE` statements, `priv_type` can be specified as any of the following:

Privilege	Meaning

ALL [PRIVILEGES]	Sets all simple privileges except GRANT OPTION
ALTER	Allows use of ALTER TABLE
ALTER ROUTINE	Alter or drop stored routines
CREATE	Allows use of CREATE TABLE
CREATE ROUTINE	Create stored routines
CREATE TEMPORARY TABLES	Allows use of CREATE TEMPORARY TABLE
CREATE VIEW	Allows use of CREATE VIEW
DELETE	Allows use of DELETE
DROP	Allows use of DROP TABLE
EXECUTE	Allows the user to run stored routines
FILE	Allows use of SELECT ... INTO OUTFILE and LOAD DATA INFILE
INDEX	Allows use of CREATE INDEX and DROP INDEX
INSERT	Allows use of INSERT
LOCK TABLES	Allows use of LOCK TABLES on tables for which you have the SELECT privilege
PROCESS	Allows use of SHOW FULL PROCESSLIST
REFERENCES	Not implemented
RELOAD	Allows use of FLUSH
REPLICATION CLIENT	Allows the user to ask where slave or master servers are
REPLICATION SLAVE	Needed for replication slaves (to read binary log events from the master)
SELECT	Allows use of SELECT
SHOW DATABASES	SHOW DATABASES shows all databases
SHOW VIEW	Allows use of SHOW CREATE VIEW
SHUTDOWN	Allows use of **mysqladmin shutdown**
SUPER	Allows use of CHANGE MASTER, KILL, PURGE MASTER LOGS, and SET GLOBAL statements, the **mysqladmin debug** command; allows you to connect (once) even if max_connections is reached
UPDATE	Allows use of UPDATE
USAGE	Synonym for ``no privileges''
GRANT OPTION	Allows privileges to be granted

The CREATE TEMPORARY TABLES, EXECUTE, LOCK TABLES, REPLICATION CLIENT, REPLICATION SLAVE, SHOW DATABASES, and SUPER privileges were added in MySQL 4.0.2. (EXECUTE is not operational until MySQL 5.0.3.) CREATE VIEW and SHOW VIEW were added in MySQL 5.0.1. CREATE ROUTINE and ALTER ROUTINE were added in MySQL 5.0.3. To use these privileges when upgrading from an earlier version of MySQL that does not have them, , you must upgrade your grant tables. See Section 2.10.7, “Upgrading the Grant Tables”.

The REFERENCES privilege currently is unused.

In older MySQL versions that do not have the SUPER privilege, specify the PROCESS privilege instead.

USAGE can be specified when you want to create a user that has no privileges.

Use `SHOW GRANTS` to determine what privileges the account has. See Section 13.5.4.10, "`SHOW GRANTS` Syntax".

You can assign global privileges by using `ON *.*` syntax or database privileges by using `ON db_name.*` syntax. If you specify `ON *` and you have selected a default database, the privileges are granted in that database. (**Warning:** If you specify `ON *` and you have *not* selected a default database, the privileges granted are global!)

The `EXECUTION`, `FILE`, `PROCESS`, `RELOAD`, `REPLICATION CLIENT`, `REPLICATION SLAVE`, `SHOW DATABASES`, `SHUTDOWN`, and `SUPER` privileges are administrative privileges that can only be granted globally (using `ON *.*` syntax).

Other privileges can be granted globally or at more specific levels.

The only `priv_type` values you can specify for a table are `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CREATE`, `DROP`, `GRANT OPTION`, `INDEX`, and `ALTER`.

The only `priv_type` values you can specify for a column (that is, when you use a `column_list` clause) are `SELECT`, `INSERT`, and `UPDATE`.

The only `priv_type` values you can specify at the routine level are `ALTER ROUTINE`, `EXECUTE`, and `GRANT`. `CREATE ROUTINE` is not a routine-level privilege because you must have this privilege to be able to create a routine in the first place.

For the global, database, table, and routine levels, `GRANT ALL` assigns only the privileges that exist at the level you are granting. For example, if you use `GRANT ALL ON db_name.*`, that is a database-level statement, so none of the global-only privileges such as `FILE` are granted.

For column-level privileges (that is, when you specify `column_list`), you must explicitly name the privileges to be granted. You cannot use `ALL` as a privilege specifier.

MySQL allows you to grant database-level privileges even if the database doesn't exist, to make it easy to prepare for database use. However, MySQL currently does not allow you to grant table-level or column-level privileges if the table doesn't exist. Similarly, you cannot grant routine-level privileges to a non-existent routine.

MySQL does not automatically revoke any privileges even if you drop a table or drop a database. If you drop a routine, and routine-level privileges granted for the routine are revoked.

Note: the '_' and '%' wildcards are allowed when specifying database names in `GRANT` statements that grant privileges at the global or database levels. This means, for example, that if you want to use a '_' character as part of a database name, you should specify it as '_' in the `GRANT` statement, to prevent the user from being able to access additional databases matching the wildcard pattern; for example, `GRANT ... ON `foo_bar`.* TO ....`

In order to accommodate granting rights to users from arbitrary hosts, MySQL supports specifying the `user` value in the form `user_name@host_name`. If a `user_name` or `host_name` value is legal as an unquoted identifier, you need not quote it. However, quotes are necessary to specify a `user_name` string containing special characters (such as '-'), or a `host_name` string containing special characters or wildcard characters (such as '%'); for example, `'test-user'@'test-hostname'`. Quote the username and hostname separately.

You can specify wildcards in the hostname. For example, `user_name@'%.loc.gov'` applies to `user_name` for any host in the `loc.gov` domain, and `user_name@'144.155.166.%'` applies to `user_name` for any host in the `144.155.166` class C subnet.

The simple form `user_name` is a synonym for `user_name@'%'`.

MySQL doesn't support wildcards in usernames. Anonymous users are defined by inserting entries with

`User=''` into the `mysql.user` table or creating a user with an empty name with the `GRANT` statement:

```
mysql> GRANT ALL ON test.* TO ''@'localhost' ...
```

When specifying quoted values, quote database, table, column, and routine names as identifiers, using backticks (`` ` ``). Quote hostnames, usernames, and passwords as strings, using apostrophes (`''`).

Warning: If you allow anonymous users to connect to the MySQL server, you should also grant privileges to all local users as *`user_name`*`@localhost`. Otherwise, the anonymous-user account for the local host in the `mysql.user` table is used when named users try to log in to the MySQL server from the local machine. (This anonymous-user account is created during MySQL installation.)

You can determine whether this applies to you by executing the following query:

```
mysql> SELECT Host, User FROM mysql.user WHERE User='';
```

If you want to delete the local anonymous-user account to avoid the problem just described, use these statements:

```
mysql> DELETE FROM mysql.user WHERE Host='localhost' AND User='';
mysql> FLUSH PRIVILEGES;
```

`GRANT` supports hostnames up to 60 characters long. Database, table, column, and routine names can be up to 64 characters. Usernames can be up to 16 characters.

The privileges for a table or column are formed additively from the logical OR of the privileges at each of the four privilege levels. For example, if the `mysql.user` table specifies that a user has a global `SELECT` privilege, the privilege cannot be denied by an entry at the database, table, or column level.

The privileges for a column can be calculated as follows:

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
```

In most cases, you grant rights to a user at only one of the privilege levels, so life isn't normally this complicated. The details of the privilege-checking procedure are presented in Section 5.5, “The MySQL Access Privilege System”.

If you grant privileges for a username/hostname combination that does not exist in the `mysql.user` table, an entry is added and remains there until deleted with a `DELETE` statement. In other words, `GRANT` may create `user` table entries, but `REVOKE` does not remove them; you must do that explicitly using `DROP USER` or `DELETE`.

In MySQL 3.22.12 or later, if a new user is created or if you have global grant privileges, the user's password is set to the password specified by the `IDENTIFIED BY` clause, if one is given. If the user had a password, it is replaced by the new one.

Warning: If you create a new user but do not specify an `IDENTIFIED BY` clause, the user has no password. This is insecure. As of MySQL 5.0.2, you can enable the `NO_AUTO_CREATE_USER` SQL mode to prevent `GRANT` from creating new users if it would otherwise do so, unless `IDENTIFIED BY` is given to provide a password.

As of MySQL 5.0.3, GRANT follows the following rules:

- If you have GRANT OPTION to a database, but no privileges to mysql privilege database, you can only grant privileges to already existing users. You will not be able to change passwords for other users.
- If you have UPDATE privilege to mysql database, you can also change passwords for other users. UPDATE privilege does not give you right to create any user.
- If you have INSERT privilege to mysql database, you can create new users with or without password. INSERT privilege does not give you right to modify an existing user.
- If you have GRANT privilege to mysql database, you can modify existing users or create new ones.

Passwords can also be set with the SET PASSWORD statement. See Section 13.5.1.5, "SET PASSWORD Syntax".

In the IDENTIFIED BY clause, the password should be given as the literal password value. It is unnecessary to use the PASSWORD() function as it is for the SET PASSWORD statement. For example:

```
GRANT ... IDENTIFIED BY 'mypass';
```

If you don't want to send the password in clear text and you know the hashed value that PASSWORD() would return for the password, you can specify the hashed value preceded by the keyword PASSWORD:

```
GRANT ... IDENTIFIED BY PASSWORD '*6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4';
```

In a C program, you can get the hashed value by using the make_scrambled_password() C API function.

If you grant privileges for a database, an entry in the mysql.db table is created if needed. If all privileges for the database are removed with REVOKE, this entry is deleted.

If a user has no privileges for a table, the table name is not displayed when the user requests a list of tables (for example, with a SHOW TABLES statement).

The SHOW DATABASES privilege allows the account to see database names by issuing the SHOW DATABASE statement. Accounts that do not have this privilege see only databases for which they have some privileges, and cannot use the statement at all if the server was started with the --skip-show-database option.

The WITH GRANT OPTION clause gives the user the ability to give to other users any privileges the user has at the specified privilege level. You should be careful to whom you give the GRANT OPTION privilege, because two users with different privileges may be able to join privileges!

You cannot grant another user a privilege you don't have yourself; the GRANT OPTION privilege allows you to give away only those privileges you possess.

Be aware that when you grant a user the GRANT OPTION privilege at a particular privilege level, any privileges the user possesses (or is given in the future!) at that level are also grantable by that user. Suppose that you grant a user the INSERT privilege on a database. If you then grant the SELECT privilege on the database and specify WITH GRANT OPTION, the user can give away not only the SELECT privilege, but also INSERT. If you then grant the UPDATE privilege to the user on the database, the user can give away INSERT, SELECT, and UPDATE.

You should not grant ALTER privileges to a normal user. If you do that, the user can try to subvert the privilege system by renaming tables!

The `MAX_QUERIES_PER_HOUR` *`count`*, `MAX_UPDATES_PER_HOUR` *`count`*, and `MAX_CONNECTIONS_PER_HOUR` *`count`* options are new in MySQL 4.0.2. They limit the number of queries, updates, and logins a user can perform during one hour. If *`count`* is 0 (the default), this means there is no limitation for that user.

The `MAX_USER_CONNECTIONS` *`count`* option is new in MySQL 5.0.3. It limits the maximum number of simultaneous connections that the account can make. If *`count`* is 0 (the default), the `max_user_connections` system variable determines the number of simultaneous connections for the account.

Note: To specify any of these resource-limit options for an existing user without affecting existing privileges, use `GRANT USAGE ON *.* ... WITH MAX_...`.

See Section 5.6.4, “Limiting Account Resources”.

MySQL can check X509 certificate attributes in addition to the usual authentication that is based on the username and password. To specify SSL-related options for a MySQL account, use the `REQUIRE` clause of the `GRANT` statement. (For background on the use of SSL with MySQL, see Section 5.6.7, “Using Secure Connections”.)

There are different possibilities for limiting connection types for an account:

- If an account has no SSL or X509 requirements, unencrypted connections are allowed if the username and password are valid. However, encrypted connections also can be used at the client's option, if the client has the proper certificate and key files.

- The `REQUIRE SSL` option tells the server to allow only SSL-encrypted connections for the account. Note that this option can be omitted if there are any access-control records that allow non-SSL connections.

  ```
  mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
      -> IDENTIFIED BY 'goodsecret' REQUIRE SSL;
  ```

- `REQUIRE X509` means that the client must have a valid certificate but that the exact certificate, issuer, and subject do not matter. The only requirement is that it should be possible to verify its signature with one of the CA certificates.

  ```
  mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
      -> IDENTIFIED BY 'goodsecret' REQUIRE X509;
  ```

- `REQUIRE ISSUER 'issuer'` places the restriction on connection attempts that the client must present a valid X509 certificate issued by CA `'issuer'`. If the client presents a certificate that is valid but has a different issuer, the server rejects the connection. Use of X509 certificates always implies encryption, so the `SSL` option is unnecessary.

  ```
  mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
      -> IDENTIFIED BY 'goodsecret'
      -> REQUIRE ISSUER '/C=FI/ST=Some-State/L=Helsinki/
         O=MySQL Finland AB/CN=Tonu Samuel/Email=tonu@example.com';
  ```

  Note that the `ISSUER` value should be entered as a single string.

- `REQUIRE SUBJECT 'subject'` places the restriction on connection attempts that the client must present a valid X509 certificate with subject `'subject'` in it. If the client presents a certificate that is valid but has a different subject, the server rejects the connection.

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    -> IDENTIFIED BY 'goodsecret'
    -> REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/
       O=MySQL demo client certificate/
       CN=Tonu Samuel/Email=tonu@example.com';
```

Note that the SUBJECT value should be entered as a single string.

- REQUIRE CIPHER 'cipher' is needed to ensure that strong enough ciphers and key lengths are used. SSL itself can be weak if old algorithms with short encryption keys are used. Using this option, you can ask for some exact cipher method to allow a connection.

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    -> IDENTIFIED BY 'goodsecret'
    -> REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

The SUBJECT, ISSUER, and CIPHER options can be combined in the REQUIRE clause like this:

```
mysql> GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    -> IDENTIFIED BY 'goodsecret'
    -> REQUIRE SUBJECT '/C=EE/ST=Some-State/L=Tallinn/
       O=MySQL demo client certificate/
       CN=Tonu Samuel/Email=tonu@example.com'
    -> AND ISSUER '/C=FI/ST=Some-State/L=Helsinki/
       O=MySQL Finland AB/CN=Tonu Samuel/Email=tonu@example.com'
    -> AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

Note that the SUBJECT and ISSUER values each should be entered as a single string.

Starting from MySQL 4.0.4, the AND keyword is optional between REQUIRE options.

The order of the options does not matter, but no option can be specified twice.

When **mysqld** starts, all privileges are read into memory. Database, table, and column privileges take effect at once, and user-level privileges take effect the next time the user connects. Modifications to the grant tables that you perform using GRANT or REVOKE are noticed by the server immediately. If you modify the grant tables manually (using INSERT, UPDATE, and so on), you should execute a FLUSH PRIVILEGES statement or run **mysqladmin flush-privileges** to tell the server to reload the grant tables. See Section 5.5.7, “When Privilege Changes Take Effect”.

Note that if you are using table or column privileges for even one user, the server examines table and column privileges for all users and this slows down MySQL a bit. Similarly, if you limit the number of queries, updates, or connections for any users, the server must monitor these values.

The biggest differences between the standard SQL and MySQL versions of GRANT are:

- In MySQL, privileges are associated with a username/hostname combination and not with only a username.
- Standard SQL doesn't have global or database-level privileges, nor does it support all the privilege types that MySQL supports.
- MySQL doesn't support the standard SQL TRIGGER or UNDER privileges.
- Standard SQL privileges are structured in a hierarchical manner. If you remove a user, all privileges

the user has been granted are revoked. This is also true in MySQL 5.0.2 and up if you use DROP USER. Before 5.0.2, the granted privileges are not automatically revoked; you must revoke them yourself. See Section 13.5.1.2, "DROP USER Syntax".

- With standard SQL, when you drop a table, all privileges for the table are revoked. With standard SQL, when you revoke a privilege, all privileges that were granted based on the privilege are also revoked. In MySQL, privileges can be dropped only with explicit REVOKE statements or by manipulating the MySQL grant tables.
- In MySQL, if you have the INSERT privilege on only some of the columns in a table, you can execute INSERT statements on the table; the columns for which you don't have the INSERT privilege are set to their default values. Standard SQL requires you to have the INSERT privilege on all columns.

13.5.1.4. RENAME USER Syntax

```
RENAME USER old_user TO new_user
    [, old_user TO new_user] ...
```

The RENAME USER statement renames existing MySQL accounts. To use it, you must have the GRANT OPTION privilege for the mysql database. An error occurs if any old account does not exist or any new account exists. The *old_user* and *new_user* values are given the same way as for the GRANT statement.

The RENAME USER statement was added in MySQL 5.0.2.

13.5.1.5. SET PASSWORD Syntax

```
SET PASSWORD = PASSWORD('some password')
SET PASSWORD FOR user = PASSWORD('some password')
```

The SET PASSWORD statement assigns a password to an existing MySQL user account.

The first syntax sets the password for the current user. Any client that has connected to the server using a non-anonymous account can change the password for that account.

The second syntax sets the password for a specific account on the current server host. Only clients with the UPDATE privilege for the mysql database can do this. The *user* value should be given in *user_name@host_name* format, where *user_name* and *host_name* are exactly as they are listed in the User and Host columns of the mysql.user table entry. For example, if you had an entry with User and Host column values of 'bob' and '%.loc.gov', you would write the statement like this:

```
mysql> SET PASSWORD FOR 'bob'@'%.loc.gov' = PASSWORD('newpass');
```

That is equivalent to the following statements:

```
mysql> UPDATE mysql.user SET Password=PASSWORD('newpass')
    -> WHERE User='bob' AND Host='%.loc.gov';
mysql> FLUSH PRIVILEGES;
```

Note: If you are connecting to a MySQL 4.1 or later server using a pre-4.1 client program, do not use the preceding SET PASSWORD or UPDATE statement without reading Section 5.5.9, "Password Hashing in MySQL 4.1" first. The password format changed in MySQL 4.1, and under certain circumstances

it is possible that if you change your password, you might not be able to connect to the server afterward.

Starting from MySQL 4.1, you can check what your current authentication `user@host` entry is by executing `SELECT CURRENT_USER()`.

13.5.2. Table Maintenance Statements

13.5.2.1. ANALYZE TABLE Syntax

```
ANALYZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...
```

This statement analyzes and stores the key distribution for a table. During the analysis, the table is locked with a read lock. This works on `MyISAM` and `BDB` tables and (as of MySQL 4.0.13) `InnoDB` tables. For `MyISAM` tables, this statement is equivalent to using **myisamchk -a**.

MySQL uses the stored key distribution to decide the order in which tables should be joined when you perform a join on something other than a constant.

The statement returns a table with the following columns:

Column	Value
`Table`	The table name
`Op`	Always `analyze`
`Msg_type`	One of `status`, `error`, `info`, or `warning`
`Msg_text`	The message

You can check the stored key distribution with the `SHOW INDEX` statement. See Section 13.5.4.11, “`SHOW INDEX` Syntax”.

If the table hasn't changed since the last `ANALYZE TABLE` statement, the table is not analyzed again.

Before MySQL 4.1.1, `ANALYZE TABLE` statements are not written to the binary log. As of MySQL 4.1.1, they are written to the binary log unless the optional `NO_WRITE_TO_BINLOG` keyword (or its alias `LOCAL`) is used.

13.5.2.2. BACKUP TABLE Syntax

```
BACKUP TABLE tbl_name [, tbl_name] ... TO '/path/to/backup/directory'
```

Note: This statement is deprecated. We are working on a better replacement for it that will provide online backup capabilities. In the meantime, the **mysqlhotcopy** script can be used instead.

`BACKUP TABLE` copies to the backup directory the minimum number of table files needed to restore the table, after flushing any buffered changes to disk. The statement works only for `MyISAM` tables. It copies the `.frm` definition and `.MYD` data files. The `.MYI` index file can be rebuilt from those two files. The directory should be specified as a full pathname.

Before using this statement, please see Section 5.7.1, “Database Backups”.

During the backup, a read lock is held for each table, one at time, as they are being backed up. If you want to back up several tables as a snapshot (preventing any of them from being changed during the backup operation), you must first issue a `LOCK TABLES` statement to obtain a read lock for every table in the group.

The statement returns a table with the following columns:

Column	Value
Table	The table name
Op	Always backup
Msg_type	One of status, error, info, or warning
Msg_text	The message

BACKUP TABLE is available in MySQL 3.23.25 and later.

13.5.2.3. CHECK TABLE Syntax

```
CHECK TABLE tbl_name [, tbl_name] ... [option] ...

option = {QUICK | FAST | MEDIUM | EXTENDED | CHANGED}
```

Checks a table or tables for errors. CHECK TABLE works for MyISAM and InnoDB tables. For MyISAM tables, the key statistics are updated.

As of MySQL 5.0.2, CHECK TABLE also can check views for problems, such as tables that are referenced in the view definition that no longer exist.

The CHECK TABLE statement returns a table with the following columns:

Column	Value
Table	The table name
Op	Always check
Msg_type	One of status, error, info, or warning
Msg_text	The message

Note that the statement might produce many rows of information for each checked table. The last row has a Msg_type value of status and the Msg_text normally should be OK. If you don't get OK, or Table is already up to date you should normally run a repair of the table. See Section 5.7.3, “Table Maintenance and Crash Recovery”. Table is already up to date means that the storage engine for the table indicated that there was no need to check the table.

The different check options that can be given are shown in the following table. These options apply only to checking MyISAM tables and are ignored for InnoDB tables and views.

Type	Meaning
QUICK	Don't scan the rows to check for incorrect links.
FAST	Only check tables that haven't been closed properly.
CHANGED	Only check tables that have been changed since the last check or haven't been closed properly.
MEDIUM	Scan rows to verify that deleted links are okay. This also calculates a key checksum for the rows and verifies this with a calculated checksum for the keys.
EXTENDED	Do a full key lookup for all keys for each row. This ensures that the table is 100% consistent, but takes a long time!

If none of the options QUICK, MEDIUM, or EXTENDED are specified, the default check type for dynamic-format MyISAM tables is MEDIUM. This is the same thing as running **myisamchk --medium-check *tbl_name*** on the table. The default check type also is MEDIUM for static-format MyISAM tables, unless CHANGED or FAST is specified. In that case, the default is QUICK. The row scan is skipped for CHANGED and FAST because the rows are very seldom corrupted.

You can combine check options, as in the following example, which does a quick check on the table to see whether it was closed properly:

```
CHECK TABLE test_table FAST QUICK;
```

Note: In some cases, CHECK TABLE changes the table. This happens if the table is marked as ``corrupted'' or ``not closed properly'' but CHECK TABLE doesn't find any problems in the table. In this case, CHECK TABLE marks the table as okay.

If a table is corrupted, it's most likely that the problem is in the indexes and not in the data part. All of the preceding check types check the indexes thoroughly and should thus find most errors.

If you just want to check a table that you assume is okay, you should use no check options or the QUICK option. The latter should be used when you are in a hurry and can take the very small risk that QUICK doesn't find an error in the data file. (In most cases, MySQL should find, under normal usage, any error in the data file. If this happens, the table is marked as ``corrupted'' and cannot be used until it's repaired.)

FAST and CHANGED are mostly intended to be used from a script (for example, to be executed from **cron**) if you want to check your table from time to time. In most cases, FAST is to be preferred over CHANGED. (The only case when it isn't preferred is when you suspect that you have found a bug in the MyISAM code.)

EXTENDED is to be used only after you have run a normal check but still get strange errors from a table when MySQL tries to update a row or find a row by key. (This is very unlikely if a normal check has succeeded!)

Some problems reported by CHECK TABLE can't be corrected automatically:

- `Found row where the auto_increment column has the value 0`.

  This means that you have a row in the table where the AUTO_INCREMENT index column contains the value 0. (It's possible to create a row where the AUTO_INCREMENT column is 0 by explicitly setting the column to 0 with an UPDATE statement.)

  This isn't an error in itself, but could cause trouble if you decide to dump the table and restore it or do an ALTER TABLE on the table. In this case, the AUTO_INCREMENT column changes value according to the rules of AUTO_INCREMENT columns, which could cause problems such as a duplicate-key error.

  To get rid of the warning, just execute an UPDATE statement to set the column to some other value than 0.

13.5.2.4. CHECKSUM TABLE Syntax

```
CHECKSUM TABLE tbl_name [, tbl_name] ... [ QUICK | EXTENDED ]
```

Reports a table checksum.

If QUICK is specified, the live table checksum is reported if it is available, or NULL otherwise. This is

very fast. A live checksum is enabled by specifying the `CHECKSUM=1` table option, currently supported only for `MyISAM` tables. See Section 13.2.6, "`CREATE TABLE` Syntax".

In `EXTENDED` mode the whole table is read row by row and the checksum is calculated. This can be very slow for large tables.

By default, if neither `QUICK` nor `EXTENDED` is specified, MySQL returns a live checksum if the table storage engine supports it and scans the table otherwise.

`CHECKSUM TABLE` returns `NULL` for non-existent tables. As of MySQL 5.0.3, a warning is generated for this condition.

This statement is implemented in MySQL 4.1.1.

13.5.2.5. `OPTIMIZE TABLE` Syntax

```
OPTIMIZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...
```

`OPTIMIZE TABLE` should be used if you have deleted a large part of a table or if you have made many changes to a table with variable-length rows (tables that have `VARCHAR`, `BLOB`, or `TEXT` columns). Deleted records are maintained in a linked list and subsequent `INSERT` operations reuse old record positions. You can use `OPTIMIZE TABLE` to reclaim the unused space and to defragment the data file.

In most setups, you need not run `OPTIMIZE TABLE` at all. Even if you do a lot of updates to variable-length rows, it's not likely that you need to do this more than once a week or month and only on certain tables.

For the moment, `OPTIMIZE TABLE` works only on `MyISAM`, `BDB` and `InnoDB` tables. For `BDB` tables, `OPTIMIZE TABLE` is currently mapped to `ANALYZE TABLE`. It was also the case for `InnoDB` tables before MySQL 4.1.3; starting from this version it is mapped to `ALTER TABLE`. See Section 13.5.2.1, "`ANALYZE TABLE` Syntax".

You can get `OPTIMIZE TABLE` to work on other table types by starting **mysqld** with the `--skip-new` or `--safe-mode` option; in this case, `OPTIMIZE TABLE` is just mapped to `ALTER TABLE`.

`OPTIMIZE TABLE` works as follows:

1. If the table has deleted or split rows, repair the table.
2. If the index pages are not sorted, sort them.
3. If the statistics are not up to date (and the repair couldn't be done by sorting the index), update them.

Note that MySQL locks the table during the time `OPTIMIZE TABLE` is running.

Before MySQL 4.1.1, `OPTIMIZE TABLE` statements are not written to the binary log. As of MySQL 4.1.1, they are written to the binary log unless the optional `NO_WRITE_TO_BINLOG` keyword (or its alias `LOCAL`) is used.

13.5.2.6. `REPAIR TABLE` Syntax

```
REPAIR [LOCAL | NO_WRITE_TO_BINLOG] TABLE
    tbl_name [, tbl_name] ... [QUICK] [EXTENDED] [USE_FRM]
```

REPAIR TABLE repairs a possibly corrupted table. By default, it has the same effect as **myisamchk --recover *tbl_name***. REPAIR TABLE works only on MyISAM tables.

Normally you should never have to run this statement. However, if disaster strikes, REPAIR TABLE is very likely to get back all your data from a MyISAM table. If your tables become corrupted often, you should try to find the reason for it, to eliminate the need to use REPAIR TABLE. See Section A.4.2, "What to Do If MySQL Keeps Crashing". See Section 14.1.4, "MyISAM Table Problems".

The statement returns a table with the following columns:

Column	Value
Table	The table name
Op	Always repair
Msg_type	One of status, error, info, or warning
Msg_text	The message

The REPAIR TABLE statement might produce many rows of information for each repaired table. The last row has a Msg_type value of status and Msg_test normally should be OK. If you don't get OK, you should try repairing the table with **myisamchk --safe-recover**, because REPAIR TABLE does not yet implement all the options of **myisamchk**. We plan to make it more flexible in the future.

If QUICK is given, REPAIR TABLE tries to repair only the index tree. This type of repair is like that done by **myisamchk --recover --quick**.

If you use EXTENDED, MySQL creates the index row by row instead of creating one index at a time with sorting. (Before MySQL 4.1, this might be better than sorting on fixed-length keys if you have long CHAR keys that compress very well.) This type of repair is like that done by **myisamchk --safe-recover**.

As of MySQL 4.0.2, there is a USE_FRM mode for REPAIR TABLE. Use it if the .MYI index file is missing or if its header is corrupted. In this mode, MySQL re-creates the .MYI file using information from the .frm file. This kind of repair cannot be done with **myisamchk**. **Note**: Use this mode **only** if you cannot use regular REPAIR modes. .MYI header contains important table metadata (in particular, current AUTO_INCREMENT value and Delete link) that are lost in REPAIR ... USE_FRM. Don't use USE_FRM if the table is compressed, as this information is also stored in .MYI file.

Before MySQL 4.1.1, REPAIR TABLE statements are not written to the binary log. As of MySQL 4.1.1, they are written to the binary log unless the optional NO_WRITE_TO_BINLOG keyword (or its alias LOCAL) is used.

Warning: If the server dies during a REPAIR TABLE operation, it's essential after restarting it that you immediately execute another REPAIR TABLE statement for the table before performing any other operations on it. (It's always good to start by making a backup.) In the worst case, you might have a new clean index file without information about the data file, and then the next operation you perform could overwrite the data file. This is an unlikely, but possible scenario.

13.5.2.7. RESTORE TABLE Syntax

```
RESTORE TABLE tbl_name [, tbl_name] ... FROM '/path/to/backup/directory'
```

Restores the table or tables from a backup that was made with BACKUP TABLE. Existing tables are not overwritten; if you try to restore over an existing table, you get an error. Just as BACKUP TABLE, RESTORE TABLE currently works only for MyISAM tables. The directory should be specified as a full pathname.

The backup for each table consists of its `.frm` format file and `.MYD` data file. The restore operation restores those files, then uses them to rebuild the `.MYI` index file. Restoring takes longer than backing up due to the need to rebuild the indexes. The more indexes the table has, the longer it takes.

The statement returns a table with the following columns:

Column	Value
`Table`	The table name
`Op`	Always `restore`
`Msg_type`	One of `status`, `error`, `info`, or `warning`
`Msg_text`	The message

13.5.3. SET Syntax

```
SET variable_assignment [, variable_assignment] ...

variable_assignment:
      user_var_name = expr
    | [GLOBAL | SESSION] system_var_name = expr
    | @@[global. | session.]system_var_name = expr
```

`SET` sets different types of variables that affect the operation of the server or your client. It can be used to assign values to user variables or system variables.

The `SET PASSWORD` statement for assigning account passwords is described in See Section 13.5.1.5, "`SET PASSWORD` Syntax".

In MySQL 4.0.3, we added the `GLOBAL` and `SESSION` options and allowed most important system variables to be changed dynamically at runtime. The system variables that you can set at runtime are described in Section 5.2.3.1, "Dynamic System Variables".

In older versions of MySQL, `SET OPTION` is used instead of `SET`, but this is deprecated; just leave out the word `OPTION`.

The following example show the different syntaxes you can use to set variables.

A user variable is written as `@var_name` and can be set as follows:

```
SET @var_name = expr;
```

Further information about user variables is given in Section 9.3, "User Variables".

System variables can be referred to in `SET` statements as `var_name`. The name optionally can be preceded by `GLOBAL` or `@@global.` to indicate explicitly that the variable is a global variable, or by `SESSION`, `@@session.`, or `@@` to indicate that it is a session variable. `LOCAL` and `@@local.` are synonyms for `SESSION` and `@@session.`. If no modifier is present, `SET` sets the session variable.

The `@@var_name` syntax for system variables is supported to make MySQL syntax compatible with some other database systems.

If you set several system variables in the same statement, the last used `GLOBAL` or `SESSION` option is used for variables that have no mode specified.

```
SET sort_buffer_size=10000;
```

```
SET @@local.sort_buffer_size=10000;
SET GLOBAL sort_buffer_size=1000000, SESSION sort_buffer_size=1000000;
SET @@sort_buffer_size=1000000;
SET @@global.sort_buffer_size=1000000, @@local.sort_buffer_size=1000000;
```

If you set a system variable using SESSION (the default), the value remains in effect until the current session ends or until you set the variable to a different value. If you set a system variable using GLOBAL, which requires the SUPER privilege, the value is remembered and used for new connections until the server restarts. If you want to make a variable setting permanent, you should put it in an option file. See Section 4.3.2, "Using Option Files".

To prevent incorrect usage, MySQL produces an error if you use SET GLOBAL with a variable that can only be used with SET SESSION or if you do not specify GLOBAL when setting a global variable.

If you want to set a SESSION variable to the GLOBAL value or a GLOBAL value to the compiled-in MySQL default value, you can set it to DEFAULT. For example, the following two statements are identical in setting the session value of max_join_size to the global value:

```
SET max_join_size=DEFAULT;
SET @@session.max_join_size=@@global.max_join_size;
```

You can get a list of most system variables with SHOW VARIABLES. See Section 13.5.4.19, "SHOW VARIABLES Syntax". To get a specific variable name or list of names that match a pattern, use a LIKE clause:

```
SHOW VARIABLES LIKE 'max_join_size';
SHOW GLOBAL VARIABLES LIKE 'max_join_size';
```

You can also get the value for a specific value by using the @@[global.|local.]var_name syntax with SELECT:

```
SELECT @@max_join_size, @@global.max_join_size;
```

When you retrieve a variable with SELECT @@var_name (that is, you do not specify global., session., or local.), MySQL returns the SESSION value if it exists and the GLOBAL value otherwise.

The following list describes variables that have non-standard syntax or that are not described in the list of system variables that is found in Section 5.2.3, "Server System Variables". Although these variables are not displayed by SHOW VARIABLES, you can obtain their values with SELECT (with the exception of CHARACTER SET and SET NAMES). For example:

```
mysql> SELECT @@AUTOCOMMIT;
+--------------+
| @@autocommit |
+--------------+
|            1 |
+--------------+
```

- AUTOCOMMIT = {0 | 1}

  Set the autocommit mode. If set to 1, all changes to a table take effect immediately. If set to 0, you have to use COMMIT to accept a transaction or ROLLBACK to cancel it. If you change AUTOCOMMIT mode from 0 to 1, MySQL performs an automatic COMMIT of any open transaction. Another way to begin a transaction is to use a START TRANSACTION or BEGIN statement. See Sec-

tion 13.4.1, "START TRANSACTION, COMMIT, and ROLLBACK Syntax".

- BIG_TABLES = {0 | 1}

  If set to 1, all temporary tables are stored on disk rather than in memory. This is a little slower, but the error The table *tbl_name* is full does not occur for SELECT operations that require a large temporary table. The default value for a new connection is 0 (use in-memory temporary tables). As of MySQL 4.0, you should normally never need to set this variable, because MySQL automatically converts in-memory tables to disk-based tables as necessary. This variable previously was named SQL_BIG_TABLES.

- CHARACTER SET {*charset_name* | DEFAULT}

  This maps all strings from and to the client with the given mapping. Before MySQL 4.1, the only allowable value for *charset_name* is cp1251_koi8, but you can add new mappings by editing the sql/convert.cc file in the MySQL source distribution. As of MySQL 4.1.1, SET CHARACTER SET sets three session system variables: character_set_client and character_set_results are set to the given character set, and character_set_connection to the value of character_set_database.

  The default mapping can be restored by using a value of DEFAULT.

  Note that the syntax for SET CHARACTER SET differs from that for setting most other options.

- FOREIGN_KEY_CHECKS = {0 | 1}

  If set to 1 (the default), foreign key constraints for InnoDB tables are checked. If set to 0, they are ignored. Disabling foreign key checking can be useful for reloading InnoDB tables in an order different than that required by their parent/child relationships. This variable was added in MySQL 3.23.52. See Section 15.7.4, "FOREIGN KEY Constraints".

- IDENTITY = *value*

  The variable is a synonym for the LAST_INSERT_ID variable. It exists for compatibility with other databases. As of MySQL 3.23.25, you can read its value with SELECT @@IDENTITY. As of MySQL 4.0.3, you can also set its value with SET IDENTITY.

- INSERT_ID = *value*

  Set the value to be used by the following INSERT or ALTER TABLE statement when inserting an AUTO_INCREMENT value. This is mainly used with the binary log.

- LAST_INSERT_ID = *value*

  Set the value to be returned from LAST_INSERT_ID(). This is stored in the binary log when you

use LAST_INSERT_ID() in a statement that updates a table. Setting this variable does not update the value returned by the mysql_insert_id() C API function.

- NAMES {'*charset_name*' | DEFAULT}

  SET NAMES sets the three session system variables character_set_client, character_set_connection, and character_set_results to the given character set. Setting character_set_connection to charset_name also sets collation_connection to the default collation for charset_name.

  The default mapping can be restored by using a value of DEFAULT.

  Note that the syntax for SET NAMES differs from that for setting most other options. This statement is available as of MySQL 4.1.0.

- SQL_NOTES = {0 | 1}

  When set to 1 (the default), warnings of Note level are recorded. When set to 0, Note warnings are suppressed. **mysqldump** includes output to set this variable to 0 so that reloading the dump file does not produce warnings for events that do not affect the integrity of the reload operation. SQL_NOTES was added in MySQL 4.1.11/5.0.3.

- SQL_AUTO_IS_NULL = {0 | 1}

  If set to 1 (the default), you can find the last inserted row for a table that contains an AUTO_INCREMENT column by using the following construct:

  ```
  WHERE auto_increment_column IS NULL
  ```

  This behavior is used by some ODBC programs, such as Access. SQL_AUTO_IS_NULL was added in MySQL 3.23.52.

- SQL_BIG_SELECTS = {0 | 1}

  If set to 0, MySQL aborts SELECT statements that probably take a very long time (that is, statements for which the optimizer estimates that the number of examined rows exceeds the value of max_join_size). This is useful when an inadvisable WHERE statement has been issued. The default value for a new connection is 1, which allows all SELECT statements.

  If you set the max_join_size system variable to a value other than DEFAULT, SQL_BIG_SELECTS is set to 0.

- SQL_BUFFER_RESULT = {0 | 1}

  SQL_BUFFER_RESULT forces results from SELECT statements to be put into temporary tables. This helps MySQL free the table locks early and can be beneficial in cases where it takes a long time to send results to the client. This variable was added in MySQL 3.23.13.

- SQL_LOG_BIN = {0 | 1}

  If set to 0, no logging is done to the binary log for the client. The client must have the SUPER privilege to set this option. This variable was added in MySQL 3.23.16.

- SQL_LOG_OFF = {0 | 1}

  If set to 1, no logging is done to the general query log for this client. The client must have the SUPER privilege to set this option.

- `SQL_LOG_UPDATE = {0 | 1}`

  If set to `0`, no logging is done to the update log for the client. The client must have the `SUPER` privilege to set this option. This variable was added in MySQL 3.22.5. Starting from MySQL 5.0.0, it is deprecated and is mapped to `SQL_LOG_BIN` (see Section D.1.5, “Changes in release 5.0.0 (22 Dec 2003: Alpha)”).

- `SQL_QUOTE_SHOW_CREATE = {0 | 1}`

  If set to `1`, `SHOW CREATE TABLE` quotes table and column names. If set to `0`, quoting is disabled. This option is enabled by default so that replication works for tables with table and column names that require quoting. This variable was added in MySQL 3.23.26. Section 13.5.4.5, “`SHOW CREATE TABLE` Syntax”.

- `SQL_SAFE_UPDATES = {0 | 1}`

  If set to `1`, MySQL aborts `UPDATE` or `DELETE` statements that do not use a key in the `WHERE` clause or a `LIMIT` clause. This makes it possible to catch `UPDATE` or `DELETE` statements where keys are not used properly and that would probably change or delete a large number of rows. This variable was added in MySQL 3.22.32.

- `SQL_SELECT_LIMIT = {`*`value`*` | DEFAULT}`

  The maximum number of records to return from `SELECT` statements. The default value for a new connection is ``unlimited.'' If you have changed the limit, the default value can be restored by using a `SQL_SELECT_LIMIT` value of `DEFAULT`.

  If a `SELECT` has a `LIMIT` clause, the `LIMIT` takes precedence over the value of `SQL_SELECT_LIMIT`.

  `SQL_SELECT_LIMIT` does not apply to `SELECT` statements executed within stored routines. It also does not apply to `SELECT` statements that do not produce a result set to be returned to the client. These include `SELECT` statements in subqueries, `CREATE TABLE ... SELECT`, and `INSERT INTO ... SELECT`.

- `SQL_WARNINGS = {0 | 1}`

  This variable controls whether single-row `INSERT` statements produce an information string if warnings occur. The default is 0. Set the value to 1 to produce an information string. This variable was added in MySQL 3.22.11.

- `TIMESTAMP = {`*`timestamp_value`*` | DEFAULT}`

  Set the time for this client. This is used to get the original timestamp if you use the binary log to restore rows. `timestamp_value` should be a Unix epoch timestamp, not a MySQL timestamp.

- `UNIQUE_CHECKS = {0 | 1}`

  If set to `1` (the default), uniqueness checks for secondary indexes in `InnoDB` tables are performed. If set to `0`, uniqueness checks are not done for index entries inserted into InnoDB's insert buffer. If you know for certain that your data does not contain uniqueness violations, you can set this to 0 to speed up large table imports to InnoDB. This variable was added in MySQL 3.23.52.

13.5.4. SHOW Syntax

SHOW has many forms that provide information about databases, tables, columns, or status information about the server. This section describes those following:

```
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE 'pattern']
SHOW CREATE DATABASE db_name
SHOW CREATE TABLE tbl_name
SHOW DATABASES [LIKE 'pattern']
SHOW [STORAGE] ENGINES
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW GRANTS FOR user
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW INNODB STATUS
SHOW [BDB] LOGS
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
SHOW STATUS [LIKE 'pattern']
SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']
SHOW [OPEN] TABLES [FROM db_name] [LIKE 'pattern']
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern']
SHOW WARNINGS [LIMIT [offset,] row_count]
```

If the syntax for a given SHOW statement includes a LIKE '*pattern*' part, '*pattern*' is a string that can contain the SQL '%' and '_' wildcard characters. The pattern is useful for restricting statement output to matching values.

Note that there are other forms of these statements described elsewhere:

- The SHOW statement has forms that provide information about replication master and slave servers:

  ```
  SHOW BINLOG EVENTS
  SHOW MASTER LOGS
  SHOW MASTER STATUS
  SHOW SLAVE HOSTS
  SHOW SLAVE STATUS
  ```

  These forms of SHOW are described in Section 13.6, “Replication Statements”.

13.5.4.1. SHOW CHARACTER SET Syntax

```
SHOW CHARACTER SET [LIKE 'pattern']
```

The SHOW CHARACTER SET statement shows all available character sets. It takes an optional LIKE clause that indicates which character set names to match. For example:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+---------+-----------------------------+-------------------+--------+
| Charset | Description                 | Default collation | Maxlen |
+---------+-----------------------------+-------------------+--------+
| latin1  | ISO 8859-1 West European    | latin1_swedish_ci |      1 |
| latin2  | ISO 8859-2 Central European | latin2_general_ci |      1 |
| latin5  | ISO 8859-9 Turkish          | latin5_turkish_ci |      1 |
| latin7  | ISO 8859-13 Baltic          | latin7_general_ci |      1 |
+---------+-----------------------------+-------------------+--------+
```

The `Maxlen` column shows the maximum number of bytes used to store one character.

`SHOW CHARACTER SET` is available as of MySQL 4.1.0.

13.5.4.2. SHOW COLLATION Syntax

```
SHOW COLLATION [LIKE 'pattern']
```

The output from `SHOW COLLATION` includes all available character sets. It takes an optional `LIKE` clause that indicates which collation names to match. For example:

```
mysql> SHOW COLLATION LIKE 'latin1%';
+-------------------+---------+----+---------+----------+---------+
| Collation         | Charset | Id | Default | Compiled | Sortlen |
+-------------------+---------+----+---------+----------+---------+
| latin1_german1_ci | latin1  |  5 |         |          |       0 |
| latin1_swedish_ci | latin1  |  8 | Yes     | Yes      |       0 |
| latin1_danish_ci  | latin1  | 15 |         |          |       0 |
| latin1_german2_ci | latin1  | 31 |         | Yes      |       2 |
| latin1_bin        | latin1  | 47 |         | Yes      |       0 |
| latin1_general_ci | latin1  | 48 |         |          |       0 |
| latin1_general_cs | latin1  | 49 |         |          |       0 |
| latin1_spanish_ci | latin1  | 94 |         |          |       0 |
+-------------------+---------+----+---------+----------+---------+
```

The `Default` column indicates whether a collation is the default for its character set. `Compiled` indicates whether the character set is compiled into the server. `Sortlen` is related to the amount of memory required to sort strings expressed in the character set.

`SHOW COLLATION` is available as of MySQL 4.1.0.

13.5.4.3. SHOW COLUMNS Syntax

```
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE 'pattern']
```

`SHOW COLUMNS` displays information about the columns in a given table. It also works for views as of MySQL 5.0.1.

If the column types differ from what you expect them to be based on your `CREATE TABLE` statement, note that MySQL sometimes changes column types when you create or alter a table. The conditions for which this occurs are described in Section 13.2.6.1, “Silent Column Specification Changes”.

The `FULL` keyword can be used from MySQL 3.23.32 on. It causes the output to include the privileges you have for each column. As of MySQL 4.1, `FULL` also causes any per-column comments to be displayed.

You can use *`db_name.tbl_name`* as an alternative to the *`tbl_name`* `FROM` *`db_name`* syntax. These two statements are equivalent:

```
mysql> SHOW COLUMNS FROM mytable FROM mydb;
mysql> SHOW COLUMNS FROM mydb.mytable;
```

`SHOW FIELDS` is a synonym for `SHOW COLUMNS`. You can also list a table's columns with the **mysqlshow** ***`db_name tbl_name`*** command.

The `DESCRIBE` statement provides information similar to `SHOW COLUMNS`. See Section 13.3.1, “`DE-`

SCRIBE Syntax (Get Information About Columns)".

13.5.4.4. SHOW CREATE DATABASE Syntax

```
SHOW CREATE {DATABASE | SCHEMA} db_name
```

Shows a CREATE DATABASE statement that creates the given database. It was added in MySQL 4.1. SHOW CREATE SCHEMA can be used as of MySQL 5.0.2.

```
mysql> SHOW CREATE DATABASE test\G
*************************** 1. row ***************************
       Database: test
Create Database: CREATE DATABASE `test`
                 /*!40100 DEFAULT CHARACTER SET latin1 */
```

13.5.4.5. SHOW CREATE TABLE Syntax

```
SHOW CREATE TABLE tbl_name
```

Shows a CREATE TABLE statement that creates the given table. It was added in MySQL 3.23.20. As of MySQL 5.0.1, this statement also works with views.

```
mysql> SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE t (
  id INT(11) default NULL auto_increment,
  s char(60) default NULL,
  PRIMARY KEY (id)
) TYPE=MyISAM
```

SHOW CREATE TABLE quotes table and column names according to the value of the SQL_QUOTE_SHOW_CREATE option. Section 13.5.3, "SET Syntax".

13.5.4.6. SHOW CREATE VIEW Syntax

```
SHOW CREATE VIEW view_name
```

This statement shows a CREATE VIEW statement that creates the given view.

```
mysql> SHOW CREATE VIEW v;
+-------+-----------------------------------------------------+
| Table | Create Table                                        |
+-------+-----------------------------------------------------+
| v     | CREATE VIEW `test`.`v` AS select 1 AS `a`,2 AS `b` |
+-------+-----------------------------------------------------+
```

This statement was added in MySQL 5.0.1.

13.5.4.7. SHOW DATABASES Syntax

```
SHOW {DATABASES | SCHEMAS} [LIKE 'pattern']
```

`SHOW DATABASES` lists the databases on the MySQL server host. You can also get this list using the **mysqlshow** command. As of MySQL 4.0.2, you see only those databases for which you have some kind of privilege, if you don't have the global `SHOW DATABASES` privilege.

If the server was started with the `--skip-show-database` option, you cannot use this statement at all unless you have the `SHOW DATABASES` privilege.

`SHOW SCHEMAS` can be used as of MySQL 5.0.2

13.5.4.8. SHOW ENGINES Syntax

```
SHOW [STORAGE] ENGINES
```

`SHOW ENGINES` shows you status information about the storage engines. This is particularly useful for checking whether a storage engine is supported, or to see what the default engine is. This statement is implemented in MySQL 4.1.2. `SHOW TABLE TYPES` is a deprecated synonym.

```
mysql> SHOW ENGINES\G
*************************** 1. row ***************************
 Engine: MyISAM
Support: DEFAULT
Comment: Default engine as of MySQL 3.23 with great performance
*************************** 2. row ***************************
 Engine: HEAP
Support: YES
Comment: Alias for MEMORY
*************************** 3. row ***************************
 Engine: MEMORY
Support: YES
Comment: Hash based, stored in memory, useful for temporary tables
*************************** 4. row ***************************
 Engine: MERGE
Support: YES
Comment: Collection of identical MyISAM tables
*************************** 5. row ***************************
 Engine: MRG_MYISAM
Support: YES
Comment: Alias for MERGE
*************************** 6. row ***************************
 Engine: ISAM
Support: NO
Comment: Obsolete storage engine, now replaced by MyISAM
*************************** 7. row ***************************
 Engine: MRG_ISAM
Support: NO
Comment: Obsolete storage engine, now replaced by MERGE
*************************** 8. row ***************************
 Engine: InnoDB
Support: YES
Comment: Supports transactions, row-level locking, and foreign keys
*************************** 9. row ***************************
 Engine: INNOBASE
Support: YES
Comment: Alias for INNODB
*************************** 10. row **************************
 Engine: BDB
Support: YES
Comment: Supports transactions and page-level locking
*************************** 11. row **************************
 Engine: BERKELEYDB
Support: YES
```

```
Comment: Alias for BDB
*************************** 12. row ***************************
 Engine: NDBCLUSTER
Support: YES
Comment: Clustered, fault-tolerant, memory-based tables
*************************** 13. row ***************************
 Engine: NDB
Support: YES
Comment: Alias for NDBCLUSTER
*************************** 14. row ***************************
 Engine: EXAMPLE
Support: YES
Comment: Example storage engine
*************************** 15. row ***************************
 Engine: ARCHIVE
Support: YES
Comment: Archive storage engine
*************************** 16. row ***************************
 Engine: CSV
Support: YES
Comment: CSV storage engine
*************************** 17. row ***************************
 Engine: FEDERATED
Support: YES
Comment: Federated MySQL storage engine
```

A `Support` value indicates whether the particular storage engine is supported, and which is the default engine. For example, if the server is started with the `--default-table-type=InnoDB` option, then the `Support` value for the `InnoDB` row has the value `DEFAULT`.

13.5.4.9. SHOW ERRORS Syntax

```
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW COUNT(*) ERRORS
```

This statement is similar to `SHOW WARNINGS`, except that instead of displaying errors, warnings, and notes, it displays only errors. `SHOW ERRORS` is available as of MySQL 4.1.0.

The `LIMIT` clause has the same syntax as for the `SELECT` statement. See Section 13.1.7, "`SELECT` Syntax".

The `SHOW COUNT(*) ERRORS` statement displays the number of errors. You can also retrieve this number from the `error_count` variable:

```
SHOW COUNT(*) ERRORS;
SELECT @@error_count;
```

For more information, see Section 13.5.4.20, "`SHOW WARNINGS` Syntax".

13.5.4.10. SHOW GRANTS Syntax

```
SHOW GRANTS FOR user
```

This statement lists the `GRANT` statements that must be issued to duplicate the privileges for a MySQL user account.

```
mysql> SHOW GRANTS FOR 'root'@'localhost';
```

```
+---------------------------------------------------------------------+
| Grants for root@localhost                                           |
+---------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' WITH GRANT OPTION |
+---------------------------------------------------------------------+
```

As of MySQL 4.1.2, to list privileges for the current session, you can use any of the following statements:

```
SHOW GRANTS;
SHOW GRANTS FOR CURRENT_USER;
SHOW GRANTS FOR CURRENT_USER();
```

Before MySQL 4.1.2, you can find out what user the session was authenticated as by selecting the value of the `CURRENT_USER()` function (new in MySQL 4.0.6). Then use that value in the `SHOW GRANTS` statement. See Section 12.8.3, “Information Functions”.

`SHOW GRANTS` is available as of MySQL 3.23.4.

13.5.4.11. `SHOW INDEX` Syntax

```
SHOW INDEX FROM tbl_name [FROM db_name]
```

`SHOW INDEX` returns table index information in a format that resembles the `SQLStatistics` call in ODBC.

`SHOW INDEX` returns the following fields:

- `Table`

  The name of the table.

- `Non_unique`

  0 if the index can't contain duplicates, 1 if it can.

- `Key_name`

  The name of the index.

- `Seq_in_index`

  The column sequence number in the index, starting with 1.

- `Column_name`

  The column name.

- `Collation`

  How the column is sorted in the index. In MySQL, this can have values `'A'` (Ascending) or `NULL` (Not sorted).

- `Cardinality`

The number of unique values in the index. This is updated by running `ANALYZE TABLE` or **myisamchk -a**. `Cardinality` is counted based on statistics stored as integers, so it's not necessarily accurate for small tables. The higher the cardinality, the greater the chance that MySQL uses the index when doing joins.

- `Sub_part`

  The number of indexed characters if the column is only partly indexed. `NULL` if the entire column is indexed.

- `Packed`

  Indicates how the key is packed. `NULL` if it is not.

- `Null`

  Contains `YES` if the column may contain `NULL`. If not, the column contains `NO` as of MySQL 5.0.3, and `''` before that.

- `Index_type`

  The index method used (`BTREE`, `FULLTEXT`, `HASH`, `RTREE`).

- `Comment`

  Various remarks. Before MySQL 4.0.2 when the `Index_type` column was added, `Comment` indicates whether an index is `FULLTEXT`.

The `Packed` and `Comment` columns were added in MySQL 3.23.0. The `Null` and `Index_type` columns were added in MySQL 4.0.2.

You can use *`db_name.tbl_name`* as an alternative to the *`tbl_name`* `FROM` *`db_name`* syntax. These two statements are equivalent:

```
mysql> SHOW INDEX FROM mytable FROM mydb;
mysql> SHOW INDEX FROM mydb.mytable;
```

`SHOW KEYS` is a synonym for `SHOW INDEX`. You can also list a table's indexes with the **mysqlshow -k** ***db_name tbl_name*** command.

13.5.4.12. `SHOW INNODB STATUS` Syntax

```
SHOW INNODB STATUS
```

This statement shows extensive information about the state of the `InnoDB` storage engine.

13.5.4.13. `SHOW LOGS` Syntax

```
SHOW [BDB] LOGS
```

`SHOW LOGS` displays status information about existing log files. It was implemented in MySQL 3.23.29. Currently, it displays only information about Berkeley DB log files, so an alias for it (available as of MySQL 4.1.1) is `SHOW BDB LOGS`.

`SHOW LOGS` returns the following fields:

- `File`

  The full path to the log file.

- `Type`

  The log file type (`BDB` for Berkeley DB log files).

- `Status`

  The status of the log file (`FREE` if the file can be removed, or `IN USE` if the file is needed by the transaction subsystem)

13.5.4.14. SHOW PRIVILEGES Syntax

```
SHOW PRIVILEGES
```

`SHOW PRIVILEGES` shows the list of system privileges that the MySQL server supports. This statement is implemented as of MySQL 4.1.0. The output depends on the version of your server.

```
mysql> SHOW PRIVILEGES\G
*************************** 1. row ***************************
Privilege: Alter
  Context: Tables
  Comment: To alter the table
*************************** 2. row ***************************
Privilege: Alter routine
  Context: Functions,Procedures
  Comment: To alter or drop stored functions/procedures
*************************** 3. row ***************************
Privilege: Create
  Context: Databases,Tables,Indexes
  Comment: To create new databases and tables
*************************** 4. row ***************************
Privilege: Create routine
  Context: Functions,Procedures
  Comment: To use CREATE FUNCTION/PROCEDURE
*************************** 5. row ***************************
Privilege: Create temporary tables
  Context: Databases
  Comment: To use CREATE TEMPORARY TABLE
*************************** 6. row ***************************
Privilege: Create view
  Context: Tables
  Comment: To create new views
*************************** 7. row ***************************
Privilege: Delete
  Context: Tables
  Comment: To delete existing rows
*************************** 8. row ***************************
Privilege: Drop
  Context: Databases,Tables
  Comment: To drop databases, tables, and views
*************************** 9. row ***************************
Privilege: Execute
  Context: Functions,Procedures
  Comment: To execute stored routines
```

```
*************************** 10. row ***************************
Privilege: File
  Context: File access on server
  Comment: To read and write files on the server
*************************** 11. row ***************************
Privilege: Grant option
  Context: Databases,Tables,Functions,Procedures
  Comment: To give to other users those privileges you possess
*************************** 12. row ***************************
Privilege: Index
  Context: Tables
  Comment: To create or drop indexes
*************************** 13. row ***************************
Privilege: Insert
  Context: Tables
  Comment: To insert data into tables
*************************** 14. row ***************************
Privilege: Lock tables
  Context: Databases
  Comment: To use LOCK TABLES (together with SELECT privilege)
*************************** 15. row ***************************
Privilege: Process
  Context: Server Admin
  Comment: To view the plain text of currently executing queries
*************************** 16. row ***************************
Privilege: References
  Context: Databases,Tables
  Comment: To have references on tables
*************************** 17. row ***************************
Privilege: Reload
  Context: Server Admin
  Comment: To reload or refresh tables, logs and privileges
*************************** 18. row ***************************
Privilege: Replication client
  Context: Server Admin
  Comment: To ask where the slave or master servers are
*************************** 19. row ***************************
Privilege: Replication slave
  Context: Server Admin
  Comment: To read binary log events from the master
*************************** 20. row ***************************
Privilege: Select
  Context: Tables
  Comment: To retrieve rows from table
*************************** 21. row ***************************
Privilege: Show databases
  Context: Server Admin
  Comment: To see all databases with SHOW DATABASES
*************************** 22. row ***************************
Privilege: Show view
  Context: Tables
  Comment: To see views with SHOW CREATE VIEW
*************************** 23. row ***************************
Privilege: Shutdown
  Context: Server Admin
  Comment: To shut down the server
*************************** 24. row ***************************
Privilege: Super
  Context: Server Admin
  Comment: To use KILL thread, SET GLOBAL, CHANGE MASTER, etc.
*************************** 25. row ***************************
Privilege: Update
  Context: Tables
  Comment: To update existing rows
```

```
*************************** 26. row ***************************
Privilege: Usage
  Context: Server Admin
  Comment: No privileges - allow connect only
```

13.5.4.15. SHOW PROCESSLIST Syntax

```
SHOW [FULL] PROCESSLIST
```

`SHOW PROCESSLIST` shows you which threads are running. You can also get this information using the **mysqladmin processlist** statement. If you have the `SUPER` privilege, you can see all threads. Otherwise, you can see only your own threads (that is, threads associated with the MySQL account that you are using). See Section 13.5.5.3, “`KILL` Syntax”. If you don't use the `FULL` keyword, only the first 100 characters of each query are shown.

Starting from MySQL 4.0.12, the statement reports the hostname for TCP/IP connections in *`host_name`*`:`*`client_port`* format to make it easier to determine which client is doing what.

This statement is very useful if you get the "too many connections" error message and want to find out what is going on. MySQL reserves one extra connection to be used by accounts that have the `SUPER` privilege, to ensure that administrators should always be able to connect and check the system (assuming that you are not giving this privilege to all your users).

Some states commonly seen in the output from `SHOW PROCESSLIST`:

- `Checking table`

  The thread is performing (automatic) checking of the table.

- `Closing tables`

  Means that the thread is flushing the changed table data to disk and closing the used tables. This should be a fast operation. If not, then you should verify that you don't have a full disk and that the disk is not in very heavy use.

- `Connect Out`

  Slave connecting to master.

- `Copying to tmp table on disk`

  The temporary result set was larger than `tmp_table_size` and the thread is changing the temporary table from in-memory to disk-based format to save memory.

- `Creating tmp table`

  The thread is creating a temporary table to hold a part of the result for the query.

- `deleting from main table`

  The server is executing the first part of a multiple-table delete and deleting only from the first table.

- `deleting from reference tables`

  The server is executing the second part of a multiple-table delete and deleting the matched rows from the other tables.

- `Flushing tables`

  The thread is executing `FLUSH TABLES` and is waiting for all threads to close their tables.

- `Killed`

  Someone has sent a kill to the thread and it should abort next time it checks the kill flag. The flag is checked in each major loop in MySQL, but in some cases it might still take a short time for the thread to die. If the thread is locked by some other thread, the kill takes effect as soon as the other thread releases its lock.

- `Locked`

  The query is locked by another query.

- `Sending data`

  The thread is processing rows for a `SELECT` statement and also is sending data to the client.

- `Sorting for group`

  The thread is doing a sort to satisfy a `GROUP BY`.

- `Sorting for order`

  The thread is doing a sort to satisfy a `ORDER BY`.

- `Opening tables`

  The thread is trying to open a table. This is should be very fast procedure, unless something prevents opening. For example, an `ALTER TABLE` or a `LOCK TABLE` statement can prevent opening a table until the statement is finished.

- `Removing duplicates`

  The query was using `SELECT DISTINCT` in such a way that MySQL couldn't optimize away the distinct operation at an early stage. Because of this, MySQL requires an extra stage to remove all duplicated rows before sending the result to the client.

- `Reopen table`

  The thread got a lock for the table, but noticed after getting the lock that the underlying table structure changed. It has freed the lock, closed the table, and is trying to reopen it.

- `Repair by sorting`

  The repair code is using sorting to create indexes.

- `Repair with keycache`

  The repair code is using creating keys one by one through the key cache. This is much slower than `Repair by sorting`.

- `Searching rows for update`

  The thread is doing a first phase to find all matching rows before updating them. This has to be done if the `UPDATE` is changing the index that is used to find the involved rows.

- `Sleeping`

The thread is waiting for the client to send a new statement to it.

- `System lock`

  The thread is waiting to get an external system lock for the table. If you are not using multiple **mysqld** servers that are accessing the same tables, you can disable system locks with the `--skip-external-locking` option.

- `Upgrading lock`

  The `INSERT DELAYED` handler is trying to get a lock for the table to insert rows.

- `Updating`

  The thread is searching for rows to update and updating them.

- `User Lock`

  The thread is waiting on a `GET_LOCK()`.

- `Waiting for tables`

  The thread got a notification that the underlying structure for a table has changed and it needs to reopen the table to get the new structure. However, to be able to reopen the table, it must wait until all other threads have closed the table in question.

  This notification happens if another thread has used `FLUSH TABLES` or one of the following statements on the table in question: `FLUSH TABLES` *`tbl_name`*, `ALTER TABLE`, `RENAME TABLE`, `REPAIR TABLE`, `ANALYZE TABLE`, or `OPTIMIZE TABLE`.

- `waiting for handler insert`

  The `INSERT DELAYED` handler has processed all pending inserts and is waiting for new ones.

Most states correspond to very quick operations. If a thread stays in any of these states for many seconds, there might be a problem that needs to be investigated.

There are some other states that are not mentioned in the preceding list, but many of them are useful only for finding bugs in the server.

13.5.4.16. `SHOW STATUS` Syntax

```
SHOW STATUS [LIKE 'pattern']
```

`SHOW STATUS` provides server status information. This information also can be obtained using the **mysqladmin extended-status** command.

Partial output is shown here. The list of variables and their values may be different for your server. The meaning of each variable is given in See Section 5.2.4, “Server Status Variables”.

```
mysql> SHOW STATUS;
+--------------------------+------------+
| Variable_name            | Value      |
+--------------------------+------------+
| Aborted_clients          | 0          |
| Aborted_connects         | 0          |
| Bytes_received           | 155372598  |
```

```
| Bytes_sent                | 1176560426 |
| Connections               | 30023      |
| Created_tmp_disk_tables   | 0          |
| Created_tmp_tables        | 8340       |
| Created_tmp_files         | 60         |
...
| Open_tables               | 1          |
| Open_files                | 2          |
| Open_streams              | 0          |
| Opened_tables             | 44600      |
| Questions                 | 2026873    |
...
| Table_locks_immediate     | 1920382    |
| Table_locks_waited        | 0          |
| Threads_cached            | 0          |
| Threads_created           | 30022      |
| Threads_connected         | 1          |
| Threads_running           | 1          |
| Uptime                    | 80380      |
+---------------------------+------------+
```

With a `LIKE` clause, the statement displays only those variables that match the pattern:

```
mysql> SHOW STATUS LIKE 'Key%';
+--------------------+----------+
| Variable_name      | Value    |
+--------------------+----------+
| Key_blocks_used    | 14955    |
| Key_read_requests  | 96854827 |
| Key_reads          | 162040   |
| Key_write_requests | 7589728  |
| Key_writes         | 3813196  |
+--------------------+----------+
```

13.5.4.17. SHOW TABLE STATUS Syntax

```
SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']
```

`SHOW TABLE STATUS` works likes `SHOW TABLE`, but provides a lot of information about each table. You can also get this list using the **mysqlshow --status *db_name*** command. This statement was added in MySQL 3.23. As of MySQL 5.0.1, it also displays information about views.

`SHOW TABLE STATUS` returns the following fields:

- `Name`

  The name of the table.

- `Engine`

  The storage engine for the table. Before MySQL 4.1.2, this value is labeled as `Type`. See Chapter 14, *MySQL Storage Engines and Table Types*.

- `Version`

  The version number of the table's `.frm` file.

- `Row_format`

The row storage format (`Fixed`, `Dynamic`, `Compressed`, `Redundant`, `Compact`). Starting with MySQL/InnoDB 5.0.3, the format of InnoDB tables is reported as `Redundant` or `Compact`. Before 5.0.3, InnoDB tables are always in the `Redundant` format.

- `Rows`

  The number of rows. Some storage engines, such as `MyISAM` and `ISAM`, store the exact count.

  For other storage engines, such as `InnoDB`, this value is an approximation, and may vary from the actual value by as much as 40 to 50%. In such cases, use `SELECT COUNT(*)` to obtain an accurate count.

- `Avg_row_length`

  The average row length.

- `Data_length`

  The length of the data file.

- `Max_data_length`

  The maximum length of the data file. This is the total number of data bytes that can be stored in the table, given the data pointer size used.

- `Index_length`

  The length of the index file.

- `Data_free`

  The number of allocated but unused bytes.

- `Auto_increment`

  The next `AUTO_INCREMENT` value.

- `Create_time`

  When the table was created.

- `Update_time`

  When the data file was last updated.

- `Check_time`

  When the table was last checked.

- `Collation`

  The table's character set and collation. (New in 4.1.1)

- `Checksum`

  The live checksum value (if any). (New in 4.1.1)

- `Create_options`

Extra options used with `CREATE TABLE`.

- `Comment`

  The comment used when creating the table (or some information why MySQL couldn't access the table information).

In the table comment, `InnoDB` tables report the free space of the tablespace to which the table belongs. For a table located in the shared tablespace, this is the free space of the shared tablespace. If you are using multiple tablespaces and the table has its own tablespace, the freespace is for just that table.

For `MEMORY` (`HEAP`) tables, the `Data_length`, `Max_data_length`, and `Index_length` values approximate the actual amount of allocated memory. The allocation algorithm reserves memory in large amounts to reduce the number of allocation operations.

For views, all the fields displayed by `SHOW TABLE STATUS` are `NULL` except that `Name` indicates the view name and `Comment` says `view`.

13.5.4.18. SHOW TABLES Syntax

```
SHOW [FULL|OPEN] TABLES [FROM db_name] [LIKE 'pattern']
```

`SHOW TABLES` lists the non-`TEMPORARY` tables in a given database. You can also get this list using the **mysqlshow *db_name*** command.

Before MySQL 5.0.1, the output from `SHOW TABLES` contains a single column of table names. Beginning with MySQL 5.0.1, also lists the views in the database. As of MySQL 5.0.2, the `FULL` modifier is supported such that `SHOW FULL TABLES` displays a second output column. Values in the second column are `BASE TABLE` for a table and `VIEW` for a view.

Note: If you have no privileges for a table, the table does not show up in the output from `SHOW TABLES` or **mysqlshow db_name**.

`SHOW OPEN TABLES` lists the tables that are currently open in the table cache. See Section 7.4.8, “How MySQL Opens and Closes Tables”. The `Comment` field in the output tells how many times the table is `cached` and `in_use`. `OPEN` can be used from MySQL 3.23.33 on.

13.5.4.19. SHOW VARIABLES Syntax

```
SHOW [GLOBAL | SESSION] VARIABLES [LIKE 'pattern']
```

`SHOW VARIABLES` shows the values of some MySQL system variables. This information also can be obtained using the **mysqladmin variables** command.

The `GLOBAL` and `SESSION` options are new in MySQL 4.0.3. With `GLOBAL`, you get the values that are used for new connections to MySQL. With `SESSION`, you get the values that are in effect for the current connection. If you use neither option, the default is `SESSION`. `LOCAL` is a synonym for `SESSION`.

If the default values are unsuitable, you can set most of these variables using command-line options when **mysqld** starts or at runtime with the `SET` statement. See Section 5.2.1, “**mysqld** Command-Line Options” and Section 13.5.3, “`SET` Syntax”.

Partial output is shown here. The list of variables and their values may be different for your server. The meaning of each variable is given in See Section 5.2.3, “Server System Variables”. Information about

tuning them is provided in Section 7.5.2, “Tuning Server Parameters”.

```
mysql> SHOW VARIABLES;
+---------------------------------+-----------------------------+
| Variable_name                   | Value                       |
+---------------------------------+-----------------------------|
| back_log                        | 50                          |
| basedir                         | /usr/local/mysql            |
| bdb_cache_size                  | 8388572                     |
| bdb_log_buffer_size             | 32768                       |
| bdb_home                        | /usr/local/mysql            |
...
| max_connections                 | 100                         |
| max_connect_errors              | 10                          |
| max_delayed_threads             | 20                          |
| max_error_count                 | 64                          |
| max_heap_table_size             | 16777216                    |
| max_join_size                   | 4294967295                  |
| max_relay_log_size              | 0                           |
| max_sort_length                 | 1024                        |
...
| timezone                        | EEST                        |
| tmp_table_size                  | 33554432                    |
| tmpdir                          | /tmp/:/mnt/hd2/tmp/         |
| version                         | 4.0.4-beta                  |
| wait_timeout                    | 28800                       |
+---------------------------------+-----------------------------+
```

With a `LIKE` clause, the statement displays only those variables that match the pattern:

```
mysql> SHOW VARIABLES LIKE 'have%';
+--------------------+----------+
| Variable_name      | Value    |
+--------------------+----------+
| have_bdb           | YES      |
| have_innodb        | YES      |
| have_isam          | YES      |
| have_raid          | NO       |
| have_symlink       | DISABLED |
| have_openssl       | YES      |
| have_query_cache   | YES      |
+--------------------+----------+
```

13.5.4.20. `SHOW WARNINGS` Syntax

```
SHOW WARNINGS [LIMIT [offset,] row_count]
SHOW COUNT(*) WARNINGS
```

`SHOW WARNINGS` shows the error, warning, and note messages that resulted from the last statement that generated messages, or nothing if the last statement that used a table generated no messages. This statement is implemented as of MySQL 4.1.0. A related statement, `SHOW ERRORS`, shows only the errors. See Section 13.5.4.9, “`SHOW ERRORS` Syntax”.

The list of messages is reset for each new statement that uses a table.

The `SHOW COUNT(*) WARNINGS` statement displays the total number of errors, warnings, and notes. You can also retrieve this number from the `warning_count` variable:

```
SHOW COUNT(*) WARNINGS;
SELECT @@warning_count;
```

The value of warning_count might be greater than the number of messages displayed by SHOW WARNINGS if the max_error_count system variable is set low enough that not all messages are stored. An example shown later in this section demonstrates how this can happen.

The LIMIT clause has the same syntax as for the SELECT statement. See Section 13.1.7, "SELECT Syntax".

The MySQL server sends back the total number of errors, warnings, and notes resulting from the last statement. If you are using the C API, this value can be obtained by calling mysql_warning_count(). See Section 22.2.3.61, "mysql_warning_count()".

Note that the framework for warnings was added in MySQL 4.1.0, at which point many statements did not generate warnings. In 4.1.1, the situation is much improved, with warnings generated for statements such as LOAD DATA INFILE and DML statements such as INSERT, UPDATE, CREATE TABLE, and ALTER TABLE.

The following DROP TABLE statement results in a note:

```
mysql> DROP TABLE IF EXISTS no_such_table;
mysql> SHOW WARNINGS;
+-------+------+-------------------------------+
| Level | Code | Message                       |
+-------+------+-------------------------------+
| Note  | 1051 | Unknown table 'no_such_table' |
+-------+------+-------------------------------+
```

Here is a simple example that shows a syntax warning for CREATE TABLE and conversion warnings for INSERT:

```
mysql> CREATE TABLE t1 (a TINYINT NOT NULL, b CHAR(4)) TYPE=MyISAM;
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1287
Message: 'TYPE=storage_engine' is deprecated, use
         'ENGINE=storage_engine' instead
1 row in set (0.00 sec)

mysql> INSERT INTO t1 VALUES(10,'mysql'),(NULL,'test'),
    -> (300,'Open Source');
Query OK, 3 rows affected, 4 warnings (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 4

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1265
Message: Data truncated for column 'b' at row 1
*************************** 2. row ***************************
  Level: Warning
   Code: 1263
Message: Data truncated, NULL supplied to NOT NULL column 'a' at row 2
*************************** 3. row ***************************
  Level: Warning
   Code: 1264
Message: Data truncated, out of range for column 'a' at row 3
*************************** 4. row ***************************
  Level: Warning
```

```
   Code: 1265
Message: Data truncated for column 'b' at row 3
4 rows in set (0.00 sec)
```

The maximum number of error, warning, and note messages to store is controlled by the `max_error_count` system variable. By default, its value is 64. To change the number of messages you want stored, change the value of `max_error_count`. In the following example, the `ALTER TABLE` statement produces three warning messages, but only one is stored because `max_error_count` has been set to `1`:

```
mysql> SHOW VARIABLES LIKE 'max_error_count';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| max_error_count | 64    |
+-----------------+-------+
1 row in set (0.00 sec)

mysql> SET max_error_count=1;
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER TABLE t1 MODIFY b CHAR;
Query OK, 3 rows affected, 3 warnings (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 3

mysql> SELECT @@warning_count;
+-----------------+
| @@warning_count |
+-----------------+
|               3 |
+-----------------+
1 row in set (0.01 sec)

mysql> SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1263 | Data truncated for column 'b' at row 1 |
+---------+------+----------------------------------------+
1 row in set (0.00 sec)
```

To disable warnings, set `max_error_count` to `0`. In this case, `warning_count` still indicates how many warnings have occurred, but none of the messages are stored.

As of MySQL 4.1.11/5.0.3, you can set the `SQL_NOTES` session variable to 0 to cause `Note`-level warnings not to be recorded.

13.5.5. Other Administrative Statements

13.5.5.1. `CACHE INDEX` Syntax

```
CACHE INDEX
  tbl_index_list [, tbl_index_list] ...
  IN key_cache_name

tbl_index_list:
  tbl_name [[INDEX|KEY] (index_name[, index_name] ...)]
```

The CACHE INDEX statement assigns table indexes to a specific key cache. It is used only for MyISAM tables.

The following statement assigns indexes from the tables t1, t2, and t3 to the key cache named hot_cache:

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+---------+--------------------+----------+----------+
| Table   | Op                 | Msg_type | Msg_text |
+---------+--------------------+----------+----------+
| test.t1 | assign_to_keycache | status   | OK       |
| test.t2 | assign_to_keycache | status   | OK       |
| test.t3 | assign_to_keycache | status   | OK       |
+---------+--------------------+----------+----------+
```

The syntax of CACHE INDEX allows you to specify that only particular indexes from a table should be assigned to the cache. However, the current implementation assigns all the table's indexes to the cache, so there is no reason to specify anything other than the table name.

The key cache referred to in a CACHE INDEX statement can be created by setting its size with a parameter setting statement or in the server parameter settings. For example:

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

Key cache parameters can be accessed as members of a structured system variable. See Section 9.4.1, “Structured System Variables”.

A key cache must exist before you can assign indexes to it:

```
mysql> CACHE INDEX t1 IN non_existent_cache;
ERROR 1284 (HY000): Unknown key cache 'non_existent_cache'
```

By default, table indexes are assigned to the main (default) key cache created at the server startup. When a key cache is destroyed, all indexes assigned to it become assigned to the default key cache again.

Index assignment affects the server globally: If one client assigns an index to a given cache, this cache is used for all queries involving the index, no matter what client issues the queries.

CACHE INDEX was added in MySQL 4.1.1.

13.5.5.2. FLUSH Syntax

```
FLUSH [LOCAL | NO_WRITE_TO_BINLOG] flush_option [, flush_option] ...
```

You should use the FLUSH statement if you want to clear some of the internal caches MySQL uses. To execute FLUSH, you must have the RELOAD privilege.

flush_option can be any of the following:

- HOSTS

  Empties the host cache tables. You should flush the host tables if some of your hosts change IP number or if you get the error message Host ... is blocked. When more than max_connect_errors errors occur successively for a given host while connecting to the MySQL server, MySQL assumes that something is wrong and blocks the host from further connec-

tion requests. Flushing the host tables allows the host to attempt to connect again. See Section A.2.5, "`Host 'host_name' is blocked`". You can start **mysqld** with `--max_connect_errors=999999999` to avoid this error message.

- `DES_KEY_FILE`

  Reloads the DES keys from the file that was specified with the `--des-key-file` option at server startup time.

- `LOGS`

  Closes and reopens all log files. If you have specified an update log file or a binary log file without an extension, the extension number of the log file is incremented by one relative to the previous file. If you have used an extension in the file name, MySQL closes and reopens the update log or binary log file. See Section 5.9.3, "The Update Log". On Unix, this is the same thing as sending a `SIGHUP` signal to the **mysqld** server (except on some Mac OS X 10.3 versions where **mysqld** ignores `SIGHUP` and `SIGQUIT`).

- `PRIVILEGES`

  Reloads the privileges from the grant tables in the `mysql` database.

- `QUERY CACHE`

  Defragment the query cache to better utilize its memory. This statement does not remove any queries from the cache, unlike `RESET QUERY CACHE`.

- `STATUS`

  Resets most status variables to zero. This is something you should use only when debugging a query. See Section 1.4.1.3, "How to Report Bugs or Problems".

- `{TABLE | TABLES} [tbl_name [, tbl_name] ...]`

  When no tables are named, closes all open tables and forces all tables in use to be closed. This also flushes the query cache. With one or more table names, flushes only the given tables. `FLUSH TABLES` also removes all query results from the query cache, like the `RESET QUERY CACHE` statement.

- `TABLES WITH READ LOCK`

  Closes all open tables and locks all tables for all databases with a read lock until you execute `UNLOCK TABLES`. This is very convenient way to get backups if you have a filesystem such as Veritas that can take snapshots in time.

- `USER_RESOURCES`

  Resets all per-hour user resources to zero. This enables clients that have reached their hourly connection, query, or update limits to resume activity. `FLUSH USER_RESOURCES` does not apply to the limit on maximum simultaneous connections. See Section 13.5.1.3, "`GRANT` and `REVOKE` Syntax".

Before MySQL 4.1.1, `FLUSH` statements are not written to the binary log. As of MySQL 4.1.1, they are written to the binary log unless the optional `NO_WRITE_TO_BINLOG` keyword (or its alias `LOCAL`) is used. Exceptions are that `FLUSH LOGS`, `FLUSH MASTER`, `FLUSH SLAVE`, and `FLUSH TABLES WITH READ LOCK` are not logged in any case because they would cause problems if replicated to a slave.

You can also access some of these statements with the **mysqladmin** utility, using the `flush-hosts`, `flush-logs`, `flush-privileges`, `flush-status`, or `flush-tables` commands.

Take also a look at the `RESET` statement used with replication. See Section 13.5.5.5, "`RESET` Syntax".

13.5.5.3. `KILL` Syntax

```
KILL [CONNECTION | QUERY] thread_id
```

Each connection to **mysqld** runs in a separate thread. You can see which threads are running with the `SHOW PROCESSLIST` statement and kill a thread with the `KILL thread_id` statement.

As of MySQL 5.0.0, `KILL` allows the optional `CONNECTION` or `QUERY` modifiers:

- `KILL CONNECTION` is the same as `KILL` with no modifier: It terminates the connection associated with the given *`thread_id`*.
- `KILL QUERY` terminates the statement that the connection currently is executing, but leaves the connection intact.

If you have the `PROCESS` privilege, you can see all threads. If you have the `SUPER` privilege, you can kill all threads and statements. Otherwise, you can see and kill only your own threads and statements.

You can also use the **mysqladmin processlist** and **mysqladmin kill** commands to examine and kill threads.

Note: You currently cannot use `KILL` with the Embedded MySQL Server library, because the embedded server merely runs inside the threads of the host application, it does not create connection threads of its own.

When you do a `KILL`, a thread-specific kill flag is set for the thread. In most cases, it might take some time for the thread to die, because the kill flag is checked only at specific intervals:

- In `SELECT`, `ORDER BY` and `GROUP BY` loops, the flag is checked after reading a block of rows. If the kill flag is set, the statement is aborted.
- During `ALTER TABLE`, the kill flag is checked before each block of rows are read from the original table. If the kill flag was set, the statement is aborted and the temporary table is deleted.
- During `UPDATE` or `DELETE`, the kill flag is checked after each block read and after each updated or deleted row. If the kill flag is set, the statement is aborted. Note that if you are not using transactions, the changes are not rolled back.
- `GET_LOCK()` aborts and returns `NULL`.
- An `INSERT DELAYED` thread quickly flushes (inserts) all rows it has in memory and terminates.
- If the thread is in the table lock handler (state: `Locked`), the table lock is quickly aborted.
- If the thread is waiting for free disk space in a write call, the write is aborted with a "disk full" error message.
- Some threads might refuse to be killed. For example, `REPAIR TABLE`, `CHECK TABLE`, and `OPTIMIZE TABLE` cannot be killed before MySQL 4.1 and run to completion. This is changed: `REPAIR TABLE` and `OPTIMIZE TABLE` can be killed as of MySQL 4.1.0, as can `CHECK TABLE`

as of MySQL 4.1.3. However, killing a REPAIR TABLE or OPTIMIZE TABLE operation on a MyISAM table results in a table that IS corrupted and is unusable (reads and writes to it fail) until you optimize or repair it again.

13.5.5.4. LOAD INDEX INTO CACHE Syntax

```
LOAD INDEX INTO CACHE
  tbl_index_list [, tbl_index_list] ...

tbl_index_list:
  tbl_name
    [[INDEX|KEY] (index_name[, index_name] ...)]
    [IGNORE LEAVES]
```

The LOAD INDEX INTO CACHE statement preloads a table index into the key cache to which it has been assigned by an explicit CACHE INDEX statement, or into the default key cache otherwise. LOAD INDEX INTO CACHE is used only for MyISAM tables.

The IGNORE LEAVES modifier causes only blocks for the non-leaf nodes of the index to be preloaded.

The following statement preloads nodes (index blocks) of indexes of the tables t1 and t2:

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+---------+--------------+----------+----------+
| Table   | Op           | Msg_type | Msg_text |
+---------+--------------+----------+----------+
| test.t1 | preload_keys | status   | OK       |
| test.t2 | preload_keys | status   | OK       |
+---------+--------------+----------+----------+
```

This statement preloads all index blocks from t1. It preloads only blocks for the non-leaf nodes from t2.

The syntax of LOAD INDEX INTO CACHE allows you to specify that only particular indexes from a table should be preloaded. However, the current implementation preloads all the table's indexes into the cache, so there is no reason to specify anything other than the table name.

LOAD INDEX INTO CACHE was added in MySQL 4.1.1.

13.5.5.5. RESET Syntax

```
RESET reset_option [, reset_option] ...
```

The RESET statement is used to clear the state of various server operations. It also acts as a stronger version of the FLUSH statement. See Section 13.5.5.2, “FLUSH Syntax”.

To execute RESET, you must have the RELOAD privilege.

reset_option can be any of the following:

- MASTER

  Deletes all binary logs listed in the index file, resets the binary log index file to be empty, and creates a new binary log file. Previously named FLUSH MASTER. See Section 13.6.1, “SQL Statements for Controlling Master Servers”.

- `QUERY CACHE`

  Removes all query results from the query cache.

- `SLAVE`

  Makes the slave forget its replication position in the master binary logs. Also resets the relay log by deleting any existing relay log files and beginning a new one. Previously named `FLUSH SLAVE`. See Section 13.6.2, “SQL Statements for Controlling Slave Servers”.

13.6. Replication Statements

This section describes replication-related SQL statements. One group of statements is used for controlling master servers. The other is used for controlling slave servers.

13.6.1. SQL Statements for Controlling Master Servers

Replication can be controlled through the SQL interface. This section discusses statements for managing master replication servers. Section 13.6.2, “SQL Statements for Controlling Slave Servers” discusses statements for managing slave servers.

13.6.1.1. `PURGE MASTER LOGS` Syntax

```
PURGE {MASTER | BINARY} LOGS TO 'log_name'
PURGE {MASTER | BINARY} LOGS BEFORE 'date'
```

Deletes all the binary logs listed in the log index that are strictly prior to the specified log or date. The logs also are removed from the list recorded in the log index file, so that the given log becomes the first.

Example:

```
PURGE MASTER LOGS TO 'mysql-bin.010';
PURGE MASTER LOGS BEFORE '2003-04-02 22:46:26';
```

The `BEFORE` variant is available as of MySQL 4.1. Its date argument can be in `'YYYY-MM-DD hh:mm:ss'` format. `MASTER` and `BINARY` are synonyms, but `BINARY` can be used only as of MySQL 4.1.1.

If you have an active slave that currently is reading one of the logs you are trying to delete, this statement does nothing and fails with an error. However, if a slave is dormant and you happen to purge one of the logs it wants to read, the slave is unable to replicate once it comes up. The statement is safe to run while slaves are replicating. You do not need to stop them.

To purge logs, follow this procedure:

1. On each slave server, use `SHOW SLAVE STATUS` to check which log it is reading.
2. Obtain a listing of the logs on the master server with `SHOW MASTER LOGS`.
3. Determine the earliest log among all the slaves. This is the target log. If all the slaves are up to date, this is the last log on the list.
4. Make a backup of all the logs you are about to delete. (The step is optional, but a good idea.)

5. Purge all logs up to but not including the target log.

13.6.1.2. RESET MASTER Syntax

```
RESET MASTER
```

Deletes all binary logs listed in the index file, resets the binary log index file to be empty, and creates a new binary log file.

This statement was named `FLUSH MASTER` before MySQL 3.23.26.

13.6.1.3. SET SQL_LOG_BIN Syntax

```
SET SQL_LOG_BIN = {0|1}
```

Disables or enables binary logging for the current connection (`SQL_LOG_BIN` is a session variable) if the client connects using an account that has the `SUPER` privilege. The statement is refused with an error if the client does not have that privilege. (Before MySQL 4.1.2, the statement was simply ignored in that case.)

13.6.1.4. SHOW BINLOG EVENTS Syntax

```
SHOW BINLOG EVENTS
   [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
```

Shows the events in the binary log. If you do not specify `'log_name'`, the first binary log is displayed.

The `LIMIT` clause has the same syntax as for the `SELECT` statement. See Section 13.1.7, "`SELECT` Syntax".

This statement is available as of MySQL 4.0.

13.6.1.5. SHOW MASTER LOGS Syntax

```
SHOW MASTER LOGS
SHOW BINARY LOGS
```

Lists the binary log files on the server. This statement is used as part of the procedure described in Section 13.6.1.1, "`PURGE MASTER LOGS` Syntax" for determining which logs can be purged.

`SHOW MASTER LOGS` was added in MySQL 3.23.38. As of MySQL 4.1.1, you can also use `SHOW BINARY LOGS`, which is equivalent.

13.6.1.6. SHOW MASTER STATUS Syntax

```
SHOW MASTER STATUS
```

Provides status information on the binary log files of the master.

13.6.1.7. SHOW SLAVE HOSTS Syntax

```
SHOW SLAVE HOSTS
```

Displays a list of slaves currently registered with the master. Any slave not started with the `--report-host=slave_name` option is not visible in that list.

13.6.2. SQL Statements for Controlling Slave Servers

Replication can be controlled through the SQL interface. This section discusses statements for managing slave replication servers. Section 13.6.1, “SQL Statements for Controlling Master Servers” discusses statements for managing master servers.

13.6.2.1. `CHANGE MASTER TO` Syntax

```
CHANGE MASTER TO master_def [, master_def] ...

master_def:
      MASTER_HOST = 'host_name'
    | MASTER_USER = 'user_name'
    | MASTER_PASSWORD = 'password'
    | MASTER_PORT = port_num
    | MASTER_CONNECT_RETRY = count
    | MASTER_LOG_FILE = 'master_log_name'
    | MASTER_LOG_POS = master_log_pos
    | RELAY_LOG_FILE = 'relay_log_name'
    | RELAY_LOG_POS = relay_log_pos
    | MASTER_SSL = {0|1}
    | MASTER_SSL_CA = 'ca_file_name'
    | MASTER_SSL_CAPATH = 'ca_directory_name'
    | MASTER_SSL_CERT = 'cert_file_name'
    | MASTER_SSL_KEY = 'key_file_name'
    | MASTER_SSL_CIPHER = 'cipher_list'
```

Changes the parameters that the slave server uses for connecting to and communicating with the master server.

`MASTER_USER`, `MASTER_PASSWORD`, `MASTER_SSL`, `MASTER_SSL_CA`, `MASTER_SSL_CAPATH`, `MASTER_SSL_CERT`, `MASTER_SSL_KEY`, and `MASTER_SSL_CIPHER` provide information for the slave about how to connect to its master.

The relay log options (`RELAY_LOG_FILE` and `RELAY_LOG_POS`) are available beginning with MySQL 4.0.

The SSL options (`MASTER_SSL`, `MASTER_SSL_CA`, `MASTER_SSL_CAPATH`, `MASTER_SSL_CERT`, `MASTER_SSL_KEY`, and `MASTER_SSL_CIPHER`) are available beginning with MySQL 4.1.1. You can change these options even on slaves that are compiled without SSL support. They are saved to the `master.info` file, but are ignored until you use a server that has SSL support enabled.

If you don't specify a given parameter, it keeps its old value, except as indicated in the following discussion. For example, if the password to connect to your MySQL master has changed, you just need to issue these statements to tell the slave about the new password:

```
mysql> STOP SLAVE; -- if replication was running
mysql> CHANGE MASTER TO MASTER_PASSWORD='new3cret';
mysql> START SLAVE; -- if you want to restart replication
```

There is no need to specify the parameters that do not change (host, port, user, and so forth).

MASTER_HOST and MASTER_PORT are the hostname (or IP address) of the master host and its TCP/IP port. Note that if MASTER_HOST is equal to localhost, then, like in other parts of MySQL, the port may be ignored (if Unix socket files can be used, for example).

If you specify MASTER_HOST or MASTER_PORT, the slave assumes that the master server is different than before (even if you specify a host or port value that is the same as the current value.) In this case, the old values for the master binary log name and position are considered no longer applicable, so if you do not specify MASTER_LOG_FILE and MASTER_LOG_POS in the statement, MASTER_LOG_FILE='' and MASTER_LOG_POS=4 are silently appended to it.

MASTER_LOG_FILE and MASTER_LOG_POS are the coordinates at which the slave I/O thread should begin reading from the master the next time the thread starts. If you specify either of them, you can't specify RELAY_LOG_FILE or RELAY_LOG_POS. If neither of MASTER_LOG_FILE or MASTER_LOG_POS are specified, the slave uses the last coordinates of the *slave SQL thread* before CHANGE MASTER was issued. This ensures that replication has no discontinuity, even if the slave SQL thread was late compared to the slave I/O thread, when you just want to change, say, the password to use. This safe behavior was introduced starting from MySQL 4.0.17 and 4.1.1. (Before these versions, the coordinates used were the last coordinates of the slave I/O thread before CHANGE MASTER was issued. This caused the SQL thread to possibly lose some events from the master, thus breaking replication.)

CHANGE MASTER *deletes all relay log files* and starts a new one, unless you specify RELAY_LOG_FILE or RELAY_LOG_POS. In that case, relay logs are kept; as of MySQL 4.1.1 the relay_log_purge global variable is set silently to 0.

CHANGE MASTER TO updates the contents of the master.info and relay-log.info files.

CHANGE MASTER is useful for setting up a slave when you have the snapshot of the master and have recorded the log and the offset corresponding to it. After loading the snapshot into the slave, you can run CHANGE MASTER TO MASTER_LOG_FILE='*log_name_on_master*', MASTER_LOG_POS=*log_offset_on_master* on the slave.

Examples:

```
mysql> CHANGE MASTER TO
    ->     MASTER_HOST='master2.mycompany.com',
    ->     MASTER_USER='replication',
    ->     MASTER_PASSWORD='bigs3cret',
    ->     MASTER_PORT=3306,
    ->     MASTER_LOG_FILE='master2-bin.001',
    ->     MASTER_LOG_POS=4,
    ->     MASTER_CONNECT_RETRY=10;

mysql> CHANGE MASTER TO
    ->     RELAY_LOG_FILE='slave-relay-bin.006',
    ->     RELAY_LOG_POS=4025;
```

The first example changes the master and master's binary log coordinates. This is used when you want to set up the slave to replicate the master.

The second example shows an operation that is less frequently used. It is done when the slave has relay logs that you want it to execute again for some reason. To do this, the master need not be reachable. You just have to use CHANGE MASTER TO and start the SQL thread (START SLAVE SQL_THREAD).

You can even use the second operation in a non-replication setup with a standalone, non-slave server, to recover after a crash. Suppose that your server has crashed and you have restored a backup. You want to replay the server's own binary logs (not relay logs, but regular binary logs), supposedly named myhost-bin.*. First, make a backup copy of these binary logs in some safe place, in case you don't exactly follow the procedure below and accidentally have the server purge the binary logs. If using

MySQL 4.1.1 or newer, use `SET GLOBAL relay_log_purge=0` for additional safety. Then start the server without the `--log-bin` option. Before MySQL 4.0.19, start it with a new (different from before) server id; in newer versions there is no need, just use the `--replicate-same-server-id` option. Start it with `--relay-log=myhost-bin` (to make the server believe that these regular binary logs are relay logs) and with `--skip-slave-start`. After the server starts, issue these statements:

```
mysql> CHANGE MASTER TO
    ->     RELAY_LOG_FILE='myhost-bin.153',
    ->     RELAY_LOG_POS=410,
    ->     MASTER_HOST='some_dummy_string';
mysql> START SLAVE SQL_THREAD;
```

The server reads and executes its own binary logs, thus achieving crash recovery. Once the recovery is finished, run `STOP SLAVE`, shut down the server, delete `master.info` and `relay-log.info`, and restart the server with its original options.

For the moment, specifying `MASTER_HOST` (even with a dummy value) is required to make the server think it is a slave. In the future, we plan to add options to get rid of these small constraints.

13.6.2.2. `LOAD DATA FROM MASTER` Syntax

```
LOAD DATA FROM MASTER
```

Takes a snapshot of the master and copies it to the slave. It updates the values of `MASTER_LOG_FILE` and `MASTER_LOG_POS` so that the slave starts replicating from the correct position. Any table and database exclusion rules specified with the `--replicate-*-do-*` and `--replicate-*-ignore-*` options are honored. `--replicate-rewrite-db` is *not* taken into account (because one user could, with this option, set up a non-unique mapping such as `--replicate-rewrite-db=db1->db3` and `--replicate-rewrite-db=db2->db3`, which would confuse the slave when it loads the master's tables).

Use of this statement is subject to the following conditions:

- It works only with `MyISAM` tables.
- It acquires a global read lock on the master while taking the snapshot, which prevents updates on the master during the load operation.

In the future, it is planned to make this statement work with `InnoDB` tables and to remove the need for a global read lock by using non-blocking online backup.

If you are loading big tables, you might have to increase the values of `net_read_timeout` and `net_write_timeout` on both your master and slave servers. See Section 5.2.3, “Server System Variables”.

Note that `LOAD DATA FROM MASTER` does *not* copy any tables from the `mysql` database. This makes it easy to have different users and privileges on the master and the slave.

The `LOAD DATA FROM MASTER` statement requires the replication account that is used to connect to the master to have the `RELOAD` and `SUPER` privileges on the master and the `SELECT` privilege for all master tables you want to load. All master tables for which the user does not have the `SELECT` privilege are ignored by `LOAD DATA FROM MASTER`. This is because the master hides them from the user: `LOAD DATA FROM MASTER` calls `SHOW DATABASES` to know the master databases to load, but `SHOW DATABASES` returns only databases for which the user has some privilege. See Section 13.5.4.7, “`SHOW DATABASES` Syntax”. On the slave's side, the user that issues `LOAD DATA FROM MASTER`

should have grants to drop and create the databases and tables that are copied.

13.6.2.3. LOAD TABLE *tbl_name* FROM MASTER Syntax

```
LOAD TABLE tbl_name FROM MASTER
```

Transfers a copy of the table from master to the slave. This statement is implemented mainly for debugging of `LOAD DATA FROM MASTER`. It requires that the account used for connecting to the master server has the `RELOAD` and `SUPER` privileges on the master and the `SELECT` privilege on the master table to load. On the slave side, the user that issues `LOAD TABLE FROM MASTER` should have privileges to drop and create the table.

The conditions for `LOAD DATA FROM MASTER` apply here, too. For example, `LOAD TABLE FROM MASTER` works only for `MyISAM` tables. The timeout notes for `LOAD DATA FROM MASTER` apply as well.

13.6.2.4. MASTER_POS_WAIT() Syntax

```
SELECT MASTER_POS_WAIT('master_log_file', master_log_pos)
```

This is a function, not a statement. It is used to ensure that the slave has read and executed events up to a given position in the master's binary log. See Section 12.8.4, “Miscellaneous Functions” for a full description.

13.6.2.5. RESET SLAVE Syntax

```
RESET SLAVE
```

Makes the slave forget its replication position in the master's binary logs. This statement is meant to be used for a clean start: It deletes the `master.info` and `relay-log.info` files, all the relay logs, and starts a new relay log.

Note: All relay logs are deleted, even if they have not been totally executed by the slave SQL thread. (This is a condition likely to exist on a replication slave if you have issued a `STOP SLAVE` statement or if the slave is highly loaded.)

Connection information stored in the `master.info` file is immediately reset using any values specified in the corresponding startup options. This information includes values such as master host, master port, master user, and master password. If the slave SQL thread was in the middle of replicating temporary tables when it was stopped, and `RESET SLAVE` is issued, these replicated temporary tables are deleted on the slave.

This statement was named `FLUSH SLAVE` before MySQL 3.23.26.

13.6.2.6. SET GLOBAL SQL_SLAVE_SKIP_COUNTER Syntax

```
SET GLOBAL SQL_SLAVE_SKIP_COUNTER = n
```

Skip the next *n* events from the master. This is useful for recovering from replication stops caused by a statement.

This statement is valid only when the slave thread is not running. Otherwise, it produces an error.

Before MySQL 4.0, omit the `GLOBAL` keyword from the statement.

13.6.2.7. SHOW SLAVE STATUS Syntax

```
SHOW SLAVE STATUS
```

Provides status information on essential parameters of the slave threads. If you issue this statement using the **mysql** client, you can use a `\G` statement terminator rather than semicolon to get a more readable vertical layout:

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
       Slave_IO_State: Waiting for master to send event
          Master_Host: localhost
          Master_User: root
          Master_Port: 3306
        Connect_Retry: 3
      Master_Log_File: gbichot-bin.005
  Read_Master_Log_Pos: 79
       Relay_Log_File: gbichot-relay-bin.005
        Relay_Log_Pos: 548
Relay_Master_Log_File: gbichot-bin.005
     Slave_IO_Running: Yes
    Slave_SQL_Running: Yes
      Replicate_Do_DB:
  Replicate_Ignore_DB:
           Last_Errno: 0
           Last_Error:
         Skip_Counter: 0
  Exec_Master_Log_Pos: 79
      Relay_Log_Space: 552
      Until_Condition: None
       Until_Log_File:
        Until_Log_Pos: 0
   Master_SSL_Allowed: No
   Master_SSL_CA_File:
   Master_SSL_CA_Path:
      Master_SSL_Cert:
    Master_SSL_Cipher:
       Master_SSL_Key:
Seconds_Behind_Master: 8
```

Depending on your version of MySQL, you may not see all the fields just shown. In particular, several fields are present only as of MySQL 4.1.1.

`SHOW SLAVE STATUS` returns the following fields:

- `Slave_IO_State`

  A copy of the `State` field of the output of `SHOW PROCESSLIST` for the slave I/O thread. This tells you if the thread is trying to connect to the master, waiting for events from the master, reconnecting to the master, and so on. Possible states are listed in Section 6.3, “Replication Implementation Details”. Looking at this field is necessary because, for example, the thread can be running but unsuccessfully trying to connect to the master; only this field makes you aware of the connection problem. The state of the SQL thread is not copied because it is simpler. If it is running, there is no problem; if it is not, you can find the error in the `Last_Error` field (described below).

  This field is present beginning with MySQL 4.1.1.

- `Master_Host`

The current master host.

- `Master_User`

  The current user used to connect to the master.

- `Master_Port`

  The current master port.

- `Connect_Retry`

  The current value of the `--master-connect-retry` option.

- `Master_Log_File`

  The name of the master binary log file from which the I/O thread is currently reading.

- `Read_Master_Log_Pos`

  The position up to which the I/O thread has read in the current master binary log.

- `Relay_Log_File`

  The name of the relay log file from which the SQL thread is currently reading and executing.

- `Relay_Log_Pos`

  The position up to which the SQL thread has read and executed in the current relay log.

- `Relay_Master_Log_File`

  The name of the master binary log file that contains the last event executed by the SQL thread.

- `Slave_IO_Running`

  Whether or not the I/O thread is started.

- `Slave_SQL_Running`

  Whether or not the SQL thread is started.

- `Replicate_Do_DB, Replicate_Ignore_DB`

  The lists of databases that were specified with the `--replicate-do-db` and `--replicate-ignore-db` options, if any.

  These fields are present beginning with MySQL 4.1.1.

- `Replicate_Do_Table, Replicate_Ignore_Table, Replicate_Wild_Do_Table, Replicate_Wild_Ignore_Table`

  The lists of tables that were specified with the `--replicate-do-table`, `--replicate-ignore-table`, `--replicate-wild-do-table`, and `--replicate-wild-ignore_table` options, if any.

  These fields are present beginning with MySQL 4.1.1.

- `Last_Errno, Last_Error`

The error number and error message returned by the most recently executed query. An error number of 0 and message of the empty string mean ``no error.'' If the `Last_Error` value is not empty, it also appears as a message in the slave's error log.

For example:

```
Last_Errno: 1051
Last_Error: error 'Unknown table 'z'' on query 'drop table z'
```

The message indicates that the table `z` existed on the master and was dropped there, but it did not exist on the slave, so `DROP TABLE` failed on the slave. (This might occur, for example, if you forget to copy the table to the slave when setting up replication.)

- `Skip_Counter`

  The last used value for `SQL_SLAVE_SKIP_COUNTER`.

- `Exec_Master_Log_Pos`

  The position of the last event executed by the SQL thread from the master's binary log (`Relay_Master_Log_File`). (`Relay_Master_Log_File`, `Exec_Master_Log_Pos`) in the master's binary log corresponds to (`Relay_Log_File`, `Relay_Log_Pos`) in the relay log.

- `Relay_Log_Space`

  The total combined size of all existing relay logs.

- `Until_Condition, Until_Log_File, Until_Log_Pos`

  The values specified in the `UNTIL` clause of the `START SLAVE` statement.

  `Until_Condition` has these values:

  - `None` if no `UNTIL` clause was specified
  - `Master` if the slave is reading until a given position in the master's binary logs
  - `Relay` if the slave is reading until a given position in its relay logs

  `Until_Log_File` and `Until_Log_Pos` indicate the log filename and position values that define the point at which the SQL thread stops executing.

  These fields are present beginning with MySQL 4.1.1.

- `Master_SSL_Allowed, Master_SSL_CA_File, Master_SSL_CA_Path, Master_SSL_Cert, Master_SSL_Cipher, Master_SSL_Key`

  These fields show the SSL parameters used by the slave to connect to the master, if any.

  `Master_SSL_Allowed` has these values:

  - `Yes` if an SSL connection to the master is allowed
  - `No` if an SSL connection to the master is not allowed

- `Ignored` if an SSL connection is allowed but the slave server does not have SSL support enabled

The values of the other SSL-related fields correspond to the values of the `--master-ca`, `--master-capath`, `--master-cert`, `--master-cipher`, and `--master-key` options.

These fields are present beginning with MySQL 4.1.1.

- `Seconds_Behind_Master`

  This field is present beginning with MySQL 4.1.1. It's been experimental and has been changed in MySQL 4.1.9. The following applies to slaves running MySQL 4.1.9 or newer. This field is an indication of how ``late'' the slave is. When the slave SQL thread is actively running (processing updates), this field is the number of seconds that have elapsed since the timestamp of the last master's event executed by that thread. When that thread has caught up on the slave I/O thread and goes idle waiting from more events from the I/O thread this field is zero. To sum up, this field measures in seconds the time difference between the slave SQL thread and the slave I/O thread.

  If the network connection between master and slave is fast, the slave I/O thread is very close to the master, so this field is a good approximation of how late the slave SQL thread is compared to the master. If the network is slow, this is **not** a good approximation; the slave SQL thread may quite often be caught up with the slow-reading slave I/O thread, so `Seconds_Behind_Master` often shows a value of 0, even if the I/O thread is late compared to the master. In other words, **this column is useful only for fast networks**.

  This time difference computation works even though your master and slave don't have identical clocks (the clock difference is computed when the slave I/O thread starts, and assumed to remain constant from then on). `Seconds_Behind_Master` is `NULL` (which means ``unknown'') if the slave SQL thread is not running, or if the slave I/O thread is not running or not connected to master. For example if the slave I/O thread is sleeping for `master-connect-retry` seconds before reconnecting, `NULL` is shown, as the slave cannot know what the master is doing, and thus cannot reliably say how late it is.

  This field has one limitation. Indeed the timestamp is preserved through replication, which means that if your master M1 is itself a slave of M0, any event from M1's binlog which has its origin in replication of an event of M0's binlog, has the timestamp of that last event. This enables MySQL to replicate `TIMESTAMP` successfully. But the drawback for `Seconds_Behind_Master` is that if M1 also receives direct updates from clients, then the value randomly goes up and down, because sometimes the last M1's event is from M0 and sometimes it is from a direct update, and so is the last timestamp.

13.6.2.8. START SLAVE Syntax

```
START SLAVE [thread_type [, thread_type] ... ]
START SLAVE [SQL_THREAD] UNTIL
    MASTER_LOG_FILE = 'log_name', MASTER_LOG_POS = log_pos
START SLAVE [SQL_THREAD] UNTIL
    RELAY_LOG_FILE = 'log_name', RELAY_LOG_POS = log_pos

thread_type: IO_THREAD | SQL_THREAD
```

`START SLAVE` with no options starts both of the slave threads. The I/O thread reads queries from the master server and stores them in the relay log. The SQL thread reads the relay log and executes the queries. `START SLAVE` requires the `SUPER` privilege.

If `START SLAVE` succeeds in starting the slave threads, it returns without any error. However, even in

that case, it might be that the slave threads start and then later stop (for example, because they don't manage to connect to the master or read its binary logs, or some other problem). `START SLAVE` does not warn you about this. You must check your slave's error log for error messages generated by the slave threads, or check that they are running fine with `SHOW SLAVE STATUS`.

As of MySQL 4.0.2, you can add `IO_THREAD` and `SQL_THREAD` options to the statement to name which of the threads to start.

As of MySQL 4.1.1, an `UNTIL` clause may be added to specify that the slave should start and run until the SQL thread reaches a given point in the master binary logs or in the slave relay logs. When the SQL thread reaches that point, it stops. If the `SQL_THREAD` option is specified in the statement, it starts only the SQL thread. Otherwise, it starts both slave threads. If the SQL thread is running, the `UNTIL` clause is ignored and a warning is issued.

With an `UNTIL` clause, you must specify both a log filename and position. Do not mix master and relay log options.

Any `UNTIL` condition is reset by a subsequent `STOP SLAVE` statement, a `START SLAVE` statement that includes no `UNTIL` clause, or a server restart.

The `UNTIL` clause can be useful for debugging replication, or to cause replication to proceed until just before the point where you want to avoid having the slave replicate a statement. For example, if an unwise `DROP TABLE` statement was executed on the master, you can use `UNTIL` to tell the slave to execute up to that point but no farther. To find what the event is, use **mysqlbinlog** with the master logs or slave relay logs, or by using a `SHOW BINLOG EVENTS` statement.

If you are using `UNTIL` to have the slave process replicated queries in sections, it is recommended that you start the slave with the `--skip-slave-start` option to prevent the SQL thread from running when the slave server starts. It is probably best to use this option in an option file rather than on the command line, so that an unexpected server restart does not cause it to be forgotten.

The `SHOW SLAVE STATUS` statement includes output fields that display the current values of the `UNTIL` condition.

This statement is called `SLAVE START` before MySQL 4.0.5. For the moment, `SLAVE START` is still accepted for backward compatibility, but is deprecated.

13.6.2.9. STOP SLAVE Syntax

```
STOP SLAVE [thread_type [, thread_type] ... ]

thread_type: IO_THREAD | SQL_THREAD
```

Stops the slave threads. `STOP SLAVE` requires the `SUPER` privilege.

Like `START SLAVE`, as of MySQL 4.0.2, this statement may be used with the `IO_THREAD` and `SQL_THREAD` options to name the thread or threads to stop.

This statement is called `SLAVE STOP` before MySQL 4.0.5. For the moment, `SLAVE STOP` is still accepted for backward compatibility, but is deprecated.

13.7. SQL Syntax for Prepared Statements

Support for server-side prepared statements was added in MySQL 4.1. This support takes advantage of the efficient client/server binary protocol, provided that you use an appropriate client programming interface. Candidate interfaces include the MySQL C API client library (for C programs), MySQL Connector/J (for Java programs), and MySQL Connector/NET. For example, the C API provides a set of

function calls that make up its prepared statement API. See Section 22.2.4, “C API Prepared Statements”. Other language interfaces can provide support for prepared statements that use the binary protocol by linking in the C client library. (The mysqli extension in PHP 5.0 does this, for example.)

Beginning with MySQL 4.1.3, an alternative interface to prepared statements is available: SQL syntax for prepared statements. This interface is not as efficient as using the binary protocol through a prepared statement API, but requires no programming because it is available directly at the SQL level:

- You can use it when no programming interface is available to you.
- You can use it from any program that allows you to send SQL statements to the server to be executed, such as the **mysql** client program.
- You can use it even if the client is using an old version of the client library. The only requirement is that you be able to connect to a server that is recent enough to support SQL syntax for prepared statements.

SQL syntax for prepared statements is intended to be used for situations such as these:

- You may want to test how prepared statements work in your application before doing the application coding. Or perhaps an application has a problem executing prepared statements and you want to determine what the problem is interactively.
- You want to create a test case that describes a problem you are having with prepared statements, so that you can file a bug report.
- You need to use prepared statements but do not have access to a programming API that supports them.

SQL syntax for prepared statements is based on three SQL statements:

```
PREPARE stmt_name FROM preparable_stmt;

EXECUTE stmt_name [USING @var_name [, @var_name] ...];

{DEALLOCATE | DROP} PREPARE stmt_name;
```

The `PREPARE` statement prepares a statement and assigns it a name, *`stmt_name`*, by which to refer to the statement later. Statement names are not case sensitive. *`preparable_stmt`* is either a string literal or a user variable that contains the text of the statement. The text must represent a single SQL statement, not multiple statements. Within the statement, '?' characters can be used as parameter markers to indicate where data values are to be bound to the query later when you execute it. The '?' characters should not be enclosed within quotes, even if you intend to bind them to string values.

If a prepared statement exists with the same name, it is deallocated implicitly before the new statement is prepared. This means that if the new statement contains an error and cannot be prepared, an error is returned and no statement with the given name exists.

The scope of a prepared statement is the client session within which it is created. Other clients cannot see it.

After preparing a statement, you execute it with an `EXECUTE` statement that refers to the prepared statement name. If the prepared statement contains any parameter markers, you must supply a `USING` clause that lists user variables containing the values to be bound to the parameters. Parameter values can be supplied only by user variables, and the `USING` clause must name exactly as many variables as the num-

ber of parameter markers in the statement.

You can execute a given prepared statement multiple times, passing it different variables or setting the variables to different values before each execution.

To deallocate a prepared statement, use the `DEALLOCATE PREPARE` statement. Attempting to execute a prepared statement after deallocating it results in an error.

If you terminate a client session without deallocating a previously prepared statement, the server deallocates it automatically.

The following statements can be used as prepared statements: `CREATE TABLE`, `DELETE`, `DO`, `INSERT`, `REPLACE`, `SELECT`, `SET`, `UPDATE`, and most `SHOW`, statements. Other statements are not yet supported.

The following examples show two equivalent ways of preparing a statement that computes the hypotenuse of a triangle given the lengths of the two sides.

The first example shows how to create a prepared statement by using a string literal to supply the text of the statement:

```
mysql> PREPARE stmt1 FROM 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> SET @a = 3;
mysql> SET @b = 4;
mysql> EXECUTE stmt1 USING @a, @b;
+------------+
| hypotenuse |
+------------+
|          5 |
+------------+
mysql> DEALLOCATE PREPARE stmt1;
```

The second example is similar, but supplies the text of the statement with a user variable:

```
mysql> SET @s = 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> PREPARE stmt2 FROM @s;
mysql> SET @a = 6;
mysql> SET @b = 8;
mysql> EXECUTE stmt2 USING @a, @b;
+------------+
| hypotenuse |
+------------+
|         10 |
+------------+
mysql> DEALLOCATE PREPARE stmt2;
```

SQL syntax for prepared statements cannot be used in nested fashion. That is, a statement passed to `PREPARE` cannot itself be a `PREPARE`, `EXECUTE`, or `DEALLOCATE PREPARE` statement.

Also, SQL syntax for prepared statements is distinct from using prepared statement API calls. For example, you cannot use the `mysql_stmt_prepare()` C API function to prepare a `PREPARE`, `EXECUTE`, or `DEALLOCATE PREPARE` statement.

Chapter 14. MySQL Storage Engines and Table Types

MySQL supports several storage engines that act as handlers for different table types. MySQL storage engines include both those that handle transaction-safe tables and those that handle non-transaction-safe tables:

- The original storage engine was `ISAM`, which managed non-transactional tables. This engine has been replaced by `MyISAM` and should no longer be used. It is deprecated in MySQL 4.1, and is removed in MySQL 5.0.
- In MySQL 3.23.0, the `MyISAM` and `HEAP` storage engines were introduced. `MyISAM` is an improved replacement for `ISAM`. The `HEAP` storage engine provides in-memory tables. The `MERGE` storage engine was added in MySQL 3.23.25. It allows a collection of identical `MyISAM` tables to be handled as a single table. All three of these storage engines handle non-transactional tables, and all are included in MySQL by default. Note that the `HEAP` storage engine has been renamed the `MEMORY` engine.
- The `InnoDB` and `BDB` storage engines that handle transaction-safe tables were introduced in later versions of MySQL 3.23. Both are available in source distributions as of MySQL 3.23.34a. `BDB` is included in MySQL-Max binary distributions on those operating systems that support it. `InnoDB` also is included in MySQL-Max binary distributions for MySQL 3.23. Beginning with MySQL 4.0, `InnoDB` is included by default in all MySQL binary distributions. In source distributions, you can enable or disable either engine by configuring MySQL as you like.
- The `EXAMPLE` storage engine was added in MySQL 4.1.3. It is a ``stub'' engine that does nothing. You can create tables with this engine, but no data can be stored into them or retrieved from them. The purpose of this engine is to serve as an example in the MySQL source code that illustrates how to begin writing new storage engines. As such, it is primarily of interest to developers.
- `NDB Cluster` is the storage engine used by MySQL Cluster to implement tables that are partitioned over many computers. It is available in source code distributions as of MySQL 4.1.2 and binary distributions as of MySQL-Max 4.1.3.
- The `ARCHIVE` storage engine was added in MySQL 4.1.3. It is used for storing large amounts of data without indexes in a very small footprint.
- The `CSV` storage engine was added in MySQL 4.1.4. This engine stores data in text files using comma-separated-values format.
- The `FEDERATED` storage engine was added in MySQL 5.0.3. This engine stores data in a remote database. In this release, it works with MySQL only, using the MySQL C Client API. Future releases will be able to connect to other data sources using other driver or client connection methods.

This chapter describes each of the MySQL storage engines except for `InnoDB` and `NDB Cluster`, which are covered in Chapter 15, *The `InnoDB` Storage Engine* and Chapter 16, *MySQL Cluster*.

When you create a new table, you can tell MySQL what type of table to create by adding an `ENGINE` or `TYPE` table option to the `CREATE TABLE` statement:

```
CREATE TABLE t (i INT) ENGINE = INNODB;
CREATE TABLE t (i INT) TYPE = MEMORY;
```

ENGINE is the preferred term, but cannot be used before MySQL 4.0.18. TYPE is available beginning with MySQL 3.23.0, the first version of MySQL for which multiple storage engines were available.

If you omit the ENGINE or TYPE option, the default storage engine is used. By default this is MyISAM. You can change it by using the --default-storage-engine or --default-table-type server startup option, or by setting the storage_engine or table_type system variable.

When MySQL is installed on Windows using the MySQL Configuration Wizard, the InnoDB storage engine is the default instead of MyISAM. See Section 2.3.5.1, “Introduction”.

To convert a table from one type to another, use an ALTER TABLE statement that indicates the new type:

```
ALTER TABLE t ENGINE = MYISAM;
ALTER TABLE t TYPE = BDB;
```

See Section 13.2.6, “CREATE TABLE Syntax” and Section 13.2.2, “ALTER TABLE Syntax”.

If you try to use a storage engine that is not compiled in or that is compiled in but deactivated, MySQL instead creates a table of type MyISAM. This behavior is convenient when you want to copy tables between MySQL servers that support different storage engines. (For example, in a replication setup, perhaps your master server supports transactional storage engines for increased safety, but the slave servers use only non-transactional storage engines for greater speed.)

This automatic substitution of the MyISAM table type when an unavailable type is specified can be confusing for new MySQL users. In MySQL 4.1 and up, a warning is generated when a table type is automatically changed.

MySQL always creates an .frm file to hold the table and column definitions. The table's index and data may be stored in one or more other files, depending on the table type. The server creates the .frm file above the storage engine level. Individual storage engines create any additional files required for the tables that they manage.

A database may contain tables of different types.

Transaction-safe tables (TSTs) have several advantages over non-transaction-safe tables (NTSTs):

- Safer. Even if MySQL crashes or you get hardware problems, you can get your data back, either by automatic recovery or from a backup plus the transaction log.
- You can combine many statements and accept them all at the same time with the COMMIT statement (if autocommit is disabled).
- You can execute ROLLBACK to ignore your changes (if autocommit is disabled).
- If an update fails, all of your changes are reverted. (With non-transaction-safe tables, all changes that have taken place are permanent.)
- Transaction-safe storage engines can provide better concurrency for tables that get many updates concurrently with reads.

Note that to use the InnoDB storage engine in MySQL 3.23, you must configure at least the innodb_data_file_path startup option. In 4.0 and up, InnoDB uses default configuration values if you specify none. See Section 15.4, “InnoDB Configuration”.

Non-transaction-safe tables have several advantages of their own, all of which occur because there is no

transaction overhead:

- Much faster
- Lower disk space requirements
- Less memory required to perform updates

You can combine transaction-safe and non-transaction-safe tables in the same statements to get the best of both worlds. However, within a transaction with autocommit disabled, changes to non-transaction-safe tables still are committed immediately and cannot be rolled back.

14.1. The MyISAM Storage Engine

MyISAM is the default storage engine as of MySQL 3.23. It is based on the ISAM code but has many useful extensions.

Each MyISAM table is stored on disk in three files. The files have names that begin with the table name and have an extension to indicate the file type. An .frm file stores the table definition. The data file has an .MYD (MYData) extension. The index file has an .MYI (MYIndex) extension,

To specify explicitly that you want a MyISAM table, indicate that with an ENGINE or TYPE table option:

```
CREATE TABLE t (i INT) ENGINE = MYISAM;
CREATE TABLE t (i INT) TYPE = MYISAM;
```

Normally, the ENGINE or TYPE option is unnecessary; MyISAM is the default storage engine unless the default has been changed.

You can check or repair MyISAM tables with the **myisamchk** utility. See Section 5.7.3.7, “Using **myisamchk** for Crash Recovery”. You can compress MyISAM tables with **myisampack** to take up much less space. See Section 8.2, “**myisampack**, the MySQL Compressed Read-only Table Generator”.

The following characteristics of the MyISAM storage engine are improvements over the older ISAM engine:

- All data values are stored with the low byte first. This makes the data machine and operating system independent. The only requirement for binary portability is that the machine uses two's-complement signed integers (as every machine for the last 20 years has) and IEEE floating-point format (also totally dominant among mainstream machines). The only area of machines that may not support binary compatibility are embedded systems, which sometimes have peculiar processors.

  There is no big speed penalty for storing data low byte first; the bytes in a table row normally are unaligned and it doesn't take that much more power to read an unaligned byte in order than in reverse order. Also, the code in the server that fetches column values is not time critical compared to other code.

- Large files (up to 63-bit file length) are supported on filesystems and operating systems that support large files.
- Dynamic-sized rows are much less fragmented when mixing deletes with updates and inserts. This is done by automatically combining adjacent deleted blocks and by extending blocks if the next block is deleted.

- The maximum number of indexes per table is 64 (32 before MySQL 4.1.2). This can be changed by recompiling. The maximum number of columns per index is 16.
- The maximum key length is 1000 bytes (500 before MySQL 4.1.2). This can be changed by recompiling. For the case of a key longer than 250 bytes, a larger key block size than the default of 1024 bytes is used.
- `BLOB` and `TEXT` columns can be indexed.
- `NULL` values are allowed in indexed columns. This takes 0-1 bytes per key.
- All numeric key values are stored with the high byte first to allow better index compression.
- Index files are usually much smaller with `MyISAM` than with `ISAM`. This means that `MyISAM` normally uses less system resources than `ISAM`, but needs more CPU time when inserting data into a compressed index.
- When records are inserted in sorted order (as when you are using an `AUTO_INCREMENT` column), the index tree is split so that the high node only contains one key. This improves space utilization in the index tree.
- Internal handling of one `AUTO_INCREMENT` column per table. `MyISAM` automatically updates this column for `INSERT/UPDATE`. This makes `AUTO_INCREMENT` columns faster (at least 10%). Values at the top of the sequence are not reused after being deleted as they are with `ISAM`. (When an `AUTO_INCREMENT` column is defined as the last column of a multiple-column index, reuse of values deleted from the top of a sequence does occur.) The `AUTO_INCREMENT` value can be reset with `ALTER TABLE` or **myisamchk**.
- If a table doesn't have free blocks in the middle of the data file, you can `INSERT` new rows into it at the same time that other threads are reading from the table. (These are known as concurrent inserts.) A free block can occur as a result of deleting rows or an update of a dynamic length row with more data than its current contents. When all free blocks are used up (filled in), future inserts become concurrent again.
- You can put the data file and index file on different directories to get more speed with the `DATA DIRECTORY` and `INDEX DIRECTORY` table options to `CREATE TABLE`. See Section 13.2.6, “`CREATE TABLE` Syntax”.
- As of MySQL 4.1, each character column can have a different character set.
- There is a flag in the `MyISAM` index file that indicates whether the table was closed correctly. If **mysqld** is started with the `--myisam-recover` option, `MyISAM` tables are automatically checked when opened and repaired if the table wasn't closed properly.
- **myisamchk** marks tables as checked if you run it with the `--update-state` option. **myisamchk --fast** checks only those tables that don't have this mark.
- **myisamchk --analyze** stores statistics for key parts, not only for whole keys as in `ISAM`.
- **myisampack** can pack `BLOB` and `VARCHAR` columns; **pack_isam** cannot.

`MyISAM` also supports the following features, which MySQL will be able to use in the near future:

- Support for a true `VARCHAR` type; a `VARCHAR` column starts with a length stored in two bytes.
- Tables with `VARCHAR` may have fixed or dynamic record length.

- VARCHAR and CHAR columns may be up to 64KB.
- A hashed computed index can be used for UNIQUE. This allows you to have UNIQUE on any combination of columns in a table. (You can't search on a UNIQUE computed index, however.)

14.1.1. MyISAM Startup Options

The following options to **mysqld** can be used to change the behavior of MyISAM tables:

- --myisam-recover=*mode*

  Set the mode for automatic recovery of crashed MyISAM tables.

- --delay-key-write=ALL

  Don't flush key buffers between writes for any MyISAM table.

  Note: If you do this, you should not use MyISAM tables from another program (such as from another MySQL server or with **myisamchk**) when the table is in use. Doing so leads to index corruption.

  Using --external-locking does not help for tables that use --delay-key-write.

See Section 5.2.1, "**mysqld** Command-Line Options".

The following system variables affect the behavior of MyISAM tables:

- bulk_insert_buffer_size

  The size of the tree cache used in bulk insert optimization. **Note**: This is a limit *per thread*!

- myisam_max_extra_sort_file_size

  Used to help MySQL to decide when to use the slow but safe key cache index creation method. **Note**: This parameter is given in megabytes before MySQL 4.0.3, and in bytes as of 4.0.3.

- myisam_max_sort_file_size

  Don't use the fast sort index method to create an index if the temporary file would become larger than this. **Note**: This parameter is given in megabytes before MySQL 4.0.3, and in bytes as of 4.0.3.

- myisam_sort_buffer_size

  Set the size of the buffer used when recovering tables.

See Section 5.2.3, "Server System Variables".

Automatic recovery is activated if you start **mysqld** with the --myisam-recover option. In this case, when the server opens a MyISAM table, it checks whether the table is marked as crashed or whether the open count variable for the table is not 0 and you are running the server with --skip-external-locking. If either of these conditions is true, the following happens:

- The table is checked for errors.

- If the server finds an error, it tries to do a fast table repair (with sorting and without re-creating the data file).
- If the repair fails because of an error in the data file (for example, a duplicate-key error), the server tries again, this time re-creating the data file.
- If the repair still fails, the server tries once more with the old repair option method (write row by row without sorting). This method should be able to repair any type of error and has low disk space requirements.

If the recovery wouldn't be able to recover all rows from a previous completed statement and you didn't specify `FORCE` in the value of the `--myisam-recover` option, automatic repair aborts with an error message in the error log:

```
Error: Couldn't repair table: test.g00pages
```

If you specify `FORCE`, a warning like this is written instead:

```
Warning: Found 344 of 354 rows when repairing ./test/g00pages
```

Note that if the automatic recovery value includes `BACKUP`, the recovery process creates files with names of the form *`tbl_name-datetime`*`.BAK`. You should have a **cron** script that automatically moves these files from the database directories to backup media.

14.1.2. Space Needed for Keys

`MyISAM` tables use B-tree indexes. You can roughly calculate the size for the index file as `(key_length+4)/0.67`, summed over all keys. This is for the worst case when all keys are inserted in sorted order and the table doesn't have any compressed keys.

String indexes are space compressed. If the first index part is a string, it is also prefix compressed. Space compression makes the index file smaller than the worst-case figure if the string column has a lot of trailing space or is a `VARCHAR` column that is not always used to the full length. Prefix compression is used on keys that start with a string. Prefix compression helps if there are many strings with an identical prefix.

In `MyISAM` tables, you can also prefix compress numbers by specifying `PACK_KEYS=1` when you create the table. This helps when you have many integer keys that have an identical prefix when the numbers are stored high-byte first.

14.1.3. `MyISAM` Table Storage Formats

`MyISAM` supports three different storage formats. Two of them (fixed and dynamic format) are chosen automatically depending on the type of columns you are using. The third, compressed format, can be created only with the **myisampack** utility.

When you `CREATE` or `ALTER` a table that has no `BLOB` or `TEXT` columns, you can force the table format to `FIXED` or `DYNAMIC` with the `ROW_FORMAT` table option. This causes `CHAR` and `VARCHAR` columns to become `CHAR` for `FIXED` format or `VARCHAR` for `DYNAMIC` format.

In the future, you will be able to compress or decompress tables by specifying `ROW_FORMAT={COMPRESSED | DEFAULT}` to `ALTER TABLE`. See Section 13.2.6, “`CREATE TABLE` Syntax”.

14.1.3.1. Static (Fixed-Length) Table Characteristics

Static format is the default for `MyISAM` tables. It is used when the table contains no variable-length columns (`VARCHAR`, `BLOB`, or `TEXT`). Each row is stored using a fixed number of bytes.

Of the three `MyISAM` storage formats, static format is the simplest and most secure (least subject to corruption). It is also the fastest of the on-disk formats. The speed comes from the easy way that rows in the data file can be found on disk: When looking up a row based on a row number in the index, multiply the row number by the row length. Also, when scanning a table, it is very easy to read a constant number of records with each disk read operation.

The security is evidenced if your computer crashes while the MySQL server is writing to a fixed-format `MyISAM` file. In this case, **myisamchk** can easily determine where each row starts and ends, so it can usually reclaim all records except the partially written one. Note that `MyISAM` table indexes can always be reconstructed based on the data rows.

General characteristics of static format tables:

- All `CHAR`, `NUMERIC`, and `DECIMAL` columns are space-padded to the column width.
- Very quick.
- Easy to cache.
- Easy to reconstruct after a crash, because records are located in fixed positions.
- Reorganization is unnecessary unless you delete a huge number of records and want to return free disk space to the operating system. To do this, use `OPTIMIZE TABLE` or **myisamchk -r**.
- Usually require more disk space than for dynamic-format tables.

14.1.3.2. Dynamic Table Characteristics

Dynamic storage format is used if a `MyISAM` table contains any variable-length columns (`VARCHAR`, `BLOB`, or `TEXT`), or if the table was created with the `ROW_FORMAT=DYNAMIC` option.

This format is a little more complex because each row has a header that indicates how long it is. One record can also end up at more than one location when it is made longer as a result of an update.

You can use `OPTIMIZE TABLE` or **myisamchk** to defragment a table. If you have fixed-length columns that you access or change frequently in a table that also contains some variable-length columns, it might be a good idea to move the variable-length columns to other tables just to avoid fragmentation.

General characteristics of dynamic-format tables:

- All string columns are dynamic except those with a length less than four.
- Each record is preceded by a bitmap that indicates which columns contain the empty string (for string columns) or zero (for numeric columns). Note that this does not include columns that contain `NULL` values. If a string column has a length of zero after trailing space removal, or a numeric column has a value of zero, it is marked in the bitmap and not saved to disk. Non-empty strings are saved as a length byte plus the string contents.
- Much less disk space usually is required than for fixed-length tables.

- Each record uses only as much space as is required. However, if a record becomes larger, it is split into as many pieces as are required, resulting in record fragmentation. For example, if you update a row with information that extends the row length, the row becomes fragmented. In this case, you may have to run `OPTIMIZE TABLE` or **myisamchk -r** from time to time to get better performance. Use **myisamchk -ei** to obtain table statistics.

- More difficult than static-format tables to reconstruct after a crash, because a record may be fragmented into many pieces and a link (fragment) may be missing.

- The expected row length for dynamic-sized records is calculated using the following expression:

  ```
  3
  + (number of columns + 7) / 8
  + (number of char columns)
  + (packed size of numeric columns)
  + (length of strings)
  + (number of NULL columns + 7) / 8
  ```

  There is a penalty of 6 bytes for each link. A dynamic record is linked whenever an update causes an enlargement of the record. Each new link is at least 20 bytes, so the next enlargement probably goes in the same link. If not, there is another link. You may check how many links there are with **myisamchk -ed**. All links may be removed with **myisamchk -r**.

14.1.3.3. Compressed Table Characteristics

Compressed storage format is a read-only format that is generated with the **myisampack** tool.

All MySQL distributions as of version 3.23.19 include **myisampack** by default. (This version is when MySQL was placed under the GPL.) For earlier versions, **myisampack** was included only with licenses or support agreements, but the server still can read tables that were compressed with **myisampack**. Compressed tables can be uncompressed with **myisamchk**. (For the `ISAM` storage engine, compressed tables can be created with **pack_isam** and uncompressed with **isamchk**.)

Compressed tables have the following characteristics:

- Compressed tables take very little disk space. This minimizes disk usage, which is very nice when using slow disks (such as CD-ROMs).

- Each record is compressed separately, so there is very little access overhead. The header for a record is fixed (1-3 bytes) depending on the biggest record in the table. Each column is compressed differently. There is usually a different Huffman tree for each column. Some of the compression types are:

  - Suffix space compression.

  - Prefix space compression.

  - Numbers with a value of zero are stored using one bit.

  - If values in an integer column have a small range, the column is stored using the smallest possible type. For example, a `BIGINT` column (eight bytes) can be stored as a `TINYINT` column (one byte) if all its values are in the range from `-128` to `127`.

  - If a column has only a small set of possible values, the column type is converted to `ENUM`.

  - A column may use a combination of the preceding compressions.

- Can handle fixed-length or dynamic-length records.

14.1.4. MyISAM Table Problems

The file format that MySQL uses to store data has been extensively tested, but there are always circumstances that may cause database tables to become corrupted.

14.1.4.1. Corrupted MyISAM Tables

Even though the MyISAM table format is very reliable (all changes to a table made by an SQL statement are written before the statement returns), you can still get corrupted tables if some of the following things happen:

- The **mysqld** process is killed in the middle of a write.
- Unexpected computer shutdown occurs (for example, the computer is turned off).
- Hardware errors.
- You are using an external program (such as **myisamchk**) on a table that is being modified by the server at the same time.
- A software bug in the MySQL or MyISAM code.

Typical symptoms for a corrupt table are:

- You get the following error while selecting data from the table:

  ```
  Incorrect key file for table: '...'. Try to repair it
  ```

- Queries don't find rows in the table or return incomplete data.

You can check whether a MyISAM table is okay with the CHECK TABLE statement. You can repair a corrupted MyISAM table with REPAIR TABLE. When **mysqld** is not running, you can also check or repair a table with the **myisamchk** command. See Section 13.5.2.3, “CHECK TABLE Syntax”, Section 13.5.2.6, “REPAIR TABLE Syntax”, and Section 5.7.3.1, “**myisamchk** Invocation Syntax”.

If your tables become corrupted frequently, you should try to determine why this is happening. The most important thing to know is whether the table became corrupted as a result of a server crash. You can verify this easily by looking for a recent restarted mysqld message in the error log. If there is such a message, it is likely that table corruption is a result of the server dying. Otherwise, corruption may have occurred during normal operation, which is a bug. You should try to create a reproducible test case that demonstrates the problem. See Section A.4.2, “What to Do If MySQL Keeps Crashing” and Section E.1.6, “Making a Test Case If You Experience Table Corruption”.

14.1.4.2. Problems from Tables Not Being Closed Properly

Each MyISAM index (.MYI) file has a counter in the header that can be used to check whether a table has been closed properly. If you get the following warning from CHECK TABLE or **myisamchk**, it means that this counter has gone out of sync:

```
clients are using or haven't closed the table properly
```

This warning doesn't necessarily mean that the table is corrupted, but you should at least check the table to verify that it's okay.

The counter works as follows:

- The first time a table is updated in MySQL, a counter in the header of the index files is incremented.
- The counter is not changed during further updates.
- When the last instance of a table is closed (because of a `FLUSH TABLES` operation or because there isn't room in the table cache), the counter is decremented if the table has been updated at any point.
- When you repair the table or check the table and it is found to be okay, the counter is reset to zero.
- To avoid problems with interaction with other processes that might check the table, the counter is not decremented on close if it was zero.

In other words, the counter can go out of sync only under these conditions:

- The `MyISAM` tables are copied without a preceding `LOCK TABLES` and `FLUSH TABLES`.
- MySQL has crashed between an update and the final close. (Note that the table may still be okay, because MySQL always issues writes for everything between each statement.)
- A table was modified by **myisamchk --recover** or **myisamchk --update-state** at the same time that it was in use by **mysqld**.
- Many **mysqld** servers are using the table and one server performed a `REPAIR TABLE` or `CHECK TABLE` on the table while it was in use by another server. In this setup, it is safe to use `CHECK TABLE`, although you might get the warning from other servers. However, `REPAIR TABLE` should be avoided because when one server replaces the data file with a new one, this is not signaled to the other servers.

  In general, it is a bad idea to share a data directory among multiple servers. See Section 5.10, “Running Multiple MySQL Servers on the Same Machine” for additional discussion.

14.2. The `MERGE` Storage Engine

The `MERGE` storage engine was introduced in MySQL 3.23.25. It is also known as the `MRG_MyISAM` engine.

A `MERGE` table is a collection of identical `MyISAM` tables that can be used as one. ``Identical'' means that all tables have identical column and index information. You can't merge tables in which the columns are listed in a different order, don't have exactly the same columns, or have the indexes in different order. However, any or all of the tables can be compressed with **myisampack**. See Section 8.2, “**myisampack**, the MySQL Compressed Read-only Table Generator”. Differences in table options such as `AVG_ROW_LENGTH`, `MAX_ROWS`, or `PACK_KEYS` do not matter.

When you create a `MERGE` table, MySQL creates two files on disk. The files have names that begin with the table name and have an extension to indicate the file type. An `.frm` file stores the table definition, and an `.MRG` file contains the names of the tables that should be used as one. (Originally, all used tables had to be in the same database as the `MERGE` table itself. This restriction has been lifted as of MySQL

4.1.1.)

You can use SELECT, DELETE, UPDATE, and (as of MySQL 4.0) INSERT on the collection of tables. For the moment, you must have SELECT, UPDATE, and DELETE privileges on the tables that you map to a MERGE table.

If you DROP the MERGE table, you are dropping only the MERGE specification. The underlying tables are not affected.

When you create a MERGE table, you must specify a UNION=(list-of-tables) clause that indicates which tables you want to use as one. You can optionally specify an INSERT_METHOD option if you want inserts for the MERGE table to happen in the first or last table of the UNION list. If you don't specify any INSERT_METHOD option or specify it with a value of NO, attempts to insert records into the MERGE table result in an error.

The following example shows how to create a MERGE table:

```
mysql> CREATE TABLE t1 (
    ->    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->    message CHAR(20));
mysql> CREATE TABLE t2 (
    ->    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->    message CHAR(20));
mysql> INSERT INTO t1 (message) VALUES ('Testing'),('table'),('t1');
mysql> INSERT INTO t2 (message) VALUES ('Testing'),('table'),('t2');
mysql> CREATE TABLE total (
    ->    a INT NOT NULL AUTO_INCREMENT,
    ->    message CHAR(20), INDEX(a))
    ->    TYPE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;
```

Note that the a column is indexed in the MERGE table, but is not declared as a PRIMARY KEY as it is in the underlying MyISAM tables. This is necessary because a MERGE table cannot enforce uniqueness over the set of underlying tables.

After creating the MERGE table, you can do things like this:

```
mysql> SELECT * FROM total;
+---+---------+
| a | message |
+---+---------+
| 1 | Testing |
| 2 | table   |
| 3 | t1      |
| 1 | Testing |
| 2 | table   |
| 3 | t2      |
+---+---------+
```

Note that you can also manipulate the .MRG file directly from outside of the MySQL server:

```
shell> cd /mysql-data-directory/current-database
shell> ls -1 t1 t2 > total.MRG
shell> mysqladmin flush-tables
```

To remap a MERGE table to a different collection of MyISAM tables, you can do one of the following:

- DROP the table and re-create it.
- Use ALTER TABLE *tbl_name* UNION=(...) to change the list of underlying tables.

- Change the `.MRG` file and issue a `FLUSH TABLE` statement for the `MERGE` table and all underlying tables to force the storage engine to read the new definition file.

`MERGE` tables can help you solve the following problems:

- Easily manage a set of log tables. For example, you can put data from different months into separate tables, compress some of them with **myisampack**, and then create a `MERGE` table to use them as one.
- Obtain more speed. You can split a big read-only table based on some criteria, and then put individual tables on different disks. A `MERGE` table on this could be much faster than using the big table.
- Do more efficient searches. If you know exactly what you are looking for, you can search in just one of the split tables for some queries and use a `MERGE` table for others. You can even have many different `MERGE` tables that use overlapping sets of tables.
- Do more efficient repairs. It's easier to repair the individual tables that are mapped to a `MERGE` table than to repair a single really big table.
- Instantly map many tables as one. A `MERGE` table need not maintain an index of its own because it uses the indexes of the individual tables. As a result, `MERGE` table collections are *very* fast to create or remap. (Note that you must still specify the index definitions when you create a `MERGE` table, even though no indexes are created.)
- If you have a set of tables that you join as a big table on demand or batch, you should instead create a `MERGE` table on them on demand. This is much faster and saves a lot of disk space.
- Exceed the file size limit for the operating system. Each `MyISAM` table is bound by this limit, but a collection of `MyISAM` tables is not.
- You can create an alias or synonym for a `MyISAM` table by defining a `MERGE` table that maps to that single table. There should be no really notable performance impact of doing this (only a couple of indirect calls and `memcpy()` calls for each read).

The disadvantages of `MERGE` tables are:

- You can use only identical `MyISAM` tables for a `MERGE` table.
- `MERGE` tables use more file descriptors. If 10 clients are using a `MERGE` table that maps to 10 tables, the server uses (10*10) + 10 file descriptors. (10 data file descriptors for each of the 10 clients, and 10 index file descriptors shared among the clients.)
- Key reads are slower. When you read a key, the `MERGE` storage engine needs to issue a read on all underlying tables to check which one most closely matches the given key. If you then do a ``read-next,'' the `MERGE` storage engine needs to search the read buffers to find the next key. Only when one key buffer is used up, the storage engine needs to read the next key block. This makes `MERGE` keys much slower on `eq_ref` searches, but not much slower on `ref` searches. See Section 7.2.1, “`EXPLAIN` Syntax (Get Information About a `SELECT`)” for more information about `eq_ref` and `ref`.

14.2.1. `MERGE` Table Problems

The following are the known problems with `MERGE` tables:

- If you use ALTER TABLE to change a MERGE table to another table type, the mapping to the underlying tables is lost. Instead, the rows from the underlying MyISAM tables are copied into the altered table, which then is assigned the new type.
- Before MySQL 4.1.1, all underlying tables and the MERGE table itself had to be in the same database.
- REPLACE doesn't work.
- You can't use DROP TABLE, ALTER TABLE, DELETE FROM without a WHERE clause, REPAIR TABLE, TRUNCATE TABLE, OPTIMIZE TABLE, or ANALYZE TABLE on any of the tables that are mapped into a MERGE table that is ``open." If you do this, the MERGE table may still refer to the original table and you get unexpected results. The easiest way to work around this deficiency is to issue a FLUSH TABLES statement to ensure that no MERGE tables remain ``open."
- A MERGE table cannot maintain UNIQUE constraints over the whole table. When you perform an INSERT, the data goes into the first or last MyISAM table (depending on the value of the INSERT_METHOD option). MySQL ensures that unique key values remain unique within that MyISAM table, but not across all the tables in the collection.
- Before MySQL 3.23.49, DELETE FROM merge_table used without a WHERE clause only clears the mapping for the table. That is, it incorrectly empties the .MRG file rather than deleting records from the mapped tables.
- Using RENAME TABLE on an active MERGE table may corrupt the table. This is fixed in MySQL 4.1.x.
- When you create a MERGE table, there is no check whether the underlying tables exist and have identical structure. When the MERGE table is used, MySQL does a quick check that the record length for all mapped tables is equal, but this is not foolproof. If you create a MERGE table from dissimilar MyISAM tables, you are very likely to run into strange problems.
- Index order in the MERGE table and its underlying tables should be the same. If you use ALTER TABLE to add a UNIQUE index to a table used in a MERGE table, and then use ALTER TABLE to add a non-unique index on the MERGE table, the index order is different for the tables if there was an old non-unique index in the underlying table. (This is because ALTER TABLE puts UNIQUE indexes before non-unique indexes to be able to detect duplicate keys as early as possible.) Consequently, queries may return unexpected results.
- DROP TABLE on a table that is in use by a MERGE table does not work on Windows because the MERGE storage engine does the table mapping hidden from the upper layer of MySQL. Because Windows doesn't allow you to delete files that are open, you first must flush all MERGE tables (with FLUSH TABLES) or drop the MERGE table before dropping the table.

14.3. The MEMORY (HEAP) Storage Engine

The MEMORY storage engine creates tables with contents that are stored in memory. Before MySQL 4.1, MEMORY tables are called HEAP tables. As of 4.1, HEAP is a synonym for MEMORY, and MEMORY is the preferred term.

Each MEMORY table is associated with one disk file. The filename begins with the table name and has an extension of .frm to indicate that it stores the table definition.

To specify explicitly that you want a MEMORY table, indicate that with an ENGINE or TYPE table option:

```
CREATE TABLE t (i INT) ENGINE = MEMORY;
```

```
CREATE TABLE t (i INT) TYPE = HEAP;
```

MEMORY tables are stored in memory and use hash indexes by default. This makes them very fast, and very useful for creating temporary tables. However, when the server shuts down, all data stored in MEMORY tables is lost. The tables continue to exist because their definitions are stored in the .frm files on disk, but their contents are empty when the server restarts.

Here is an example that shows how you might create, use, and remove a MEMORY table:

```
mysql> CREATE TABLE test TYPE=MEMORY
    ->     SELECT ip,SUM(downloads) AS down
    ->     FROM log_table GROUP BY ip;
mysql> SELECT COUNT(ip),AVG(down) FROM test;
mysql> DROP TABLE test;
```

MEMORY tables have the following characteristics:

- Space for MEMORY tables is allocated in small blocks. The tables use 100% dynamic hashing (on inserting). No overflow areas and no extra key space are needed. There is no extra space needed for free lists. Deleted rows are put in a linked list and are reused when you insert new data into the table. MEMORY tables also don't have problems with deletes plus inserts, which is common with hashed tables.
- MEMORY tables allow up to 32 indexes per table, 16 columns per index, and a maximum key length of 500 bytes.
- Before MySQL 4.1, the MEMORY storage engine implements only hash indexes. From MySQL 4.1 on, hash indexes are still the default, but you can specify explicitly that a MEMORY table index should be HASH or BTREE by adding a USING clause:

  ```
  CREATE TABLE lookup
      (id INT, INDEX USING HASH (id))
      ENGINE = MEMORY;
  CREATE TABLE lookup
      (id INT, INDEX USING BTREE (id))
      ENGINE = MEMORY;
  ```

  General characteristics of B-tree and hash indexes are described in Section 7.4.5, “How MySQL Uses Indexes”.
- You can have non-unique keys in a MEMORY table. (This is an uncommon feature for implementations of hash indexes.)
- If you have a hash index on a MEMORY table that has a high degree of key duplication (many index entries containing the same value), updates to the table that affect key values and all deletes are significantly slower. The degree of slowdown is proportional to the degree of duplication (or, inversely proportional to the index cardinality). You can use a BTREE index to avoid this problem.
- MEMORY tables use a fixed record length format.
- MEMORY doesn't support BLOB or TEXT columns.
- MEMORY doesn't support AUTO_INCREMENT columns before MySQL 4.1.0.
- Prior to MySQL 4.0.2, MEMORY doesn't support indexes on columns that can contain NULL values.

- MEMORY tables are shared between all clients (just like any other non-TEMPORARY table).
- MEMORY table contents are stored in memory, which is a property that MEMORY tables share with internal tables that the server creates on the fly while processing queries. However, the two types of tables differ in that MEMORY tables are not subject to storage conversion, whereas internal tables are:
  - If an internal table becomes too large, the server automatically converts it to an on-disk table. The size limit is determined by the value of the tmp_table_size system variable.
  - MEMORY tables are never converted to disk tables. To ensure that you don't accidentally do anything foolish, you can set the max_heap_table_size system variable to impose a maximum size on MEMORY tables. For individual tables, you can also specify a MAX_ROWS table option in the CREATE TABLE statement.
- The server needs enough extra memory to maintain all MEMORY tables that are in use at the same time.
- To free memory used by a MEMORY table if you no longer require its contents, you should execute DELETE or TRUNCATE TABLE, or else remove the table with DROP TABLE.
- If you want to populate a MEMORY table when the MySQL server starts, you can use the --init-file option. For example, you can put statements such as INSERT INTO ... SELECT or LOAD DATA INFILE into the file to load the table from some persistent data source. See Section 5.2.1, "**mysqld** Command-Line Options".
- If you are using replication, the master server's MEMORY tables become empty when it is shut down and restarted. However, a slave is not aware that these tables have become empty, so it returns out-of-date content if you select data from them. Beginning with MySQL 4.0.18, when a MEMORY table is used on the master for the first time since the master's startup, a DELETE FROM statement is written to the master's binary log automatically, thus synchronizing the slave to the master again. Note that even with this strategy, the slave still has out-of-date data in the table during the interval between the master's restart and its first use of the table. But if you use the --init-file option to populate the MEMORY table on the master at startup, it ensures that the failing time interval is zero.
- The memory needed for one row in a MEMORY table is calculated using the following expression:

  ```
  SUM_OVER_ALL_BTREE_KEYS(max_length_of_key + sizeof(char*) * 4)
  + SUM_OVER_ALL_HASH_KEYS(sizeof(char*) * 2)
  + ALIGN(length_of_row+1, sizeof(char*))
  ```

  ALIGN() represents a round-up factor to cause the row length to be an exact multiple of the char pointer size. sizeof(char*) is 4 on 32-bit machines and 8 on 64-bit machines.

14.4. The BDB (BerkeleyDB) Storage Engine

Sleepycat Software has provided MySQL with the Berkeley DB transactional storage engine. This storage engine typically is called BDB for short. Support for the BDB storage engine is included in MySQL source distributions starting from version 3.23.34a and is activated in MySQL-Max binary distributions.

BDB tables may have a greater chance of surviving crashes and are also capable of COMMIT and ROLLBACK operations on transactions. The MySQL source distribution comes with a BDB distribution that has a couple of small patches to make it work more smoothly with MySQL. You can't use a non-patched BDB version with MySQL.

We at MySQL AB are working in close cooperation with Sleepycat to keep the quality of the MySQL/BDB interface high. (Even though Berkeley DB is in itself very tested and reliable, the MySQL interface

is still considered gamma quality. We are improving and optimizing it.)

When it comes to support for any problems involving BDB tables, we are committed to helping our users locate the problem and create a reproducible test case. Any such test case is forwarded to Sleepycat, which in turn helps us find and fix the problem. As this is a two-stage operation, any problems with BDB tables may take a little longer for us to fix than for other storage engines. However, we anticipate no significant difficulties with this procedure because the Berkeley DB code itself is used in many applications other than MySQL.

For general information about Berkeley DB, please visit the Sleepycat Web site, http://www.sleepycat.com/.

14.4.1. Operating Systems Supported by BDB

Currently, we know that the BDB storage engine works with the following operating systems:

- Linux 2.x Intel
- Sun Solaris (SPARC and x86)
- FreeBSD 4.x/5.x (x86, sparc64)
- IBM AIX 4.3.x
- SCO OpenServer
- SCO UnixWare 7.1.x

BDB does not work with the following operating systems:

- Linux 2.x Alpha
- Linux 2.x AMD64
- Linux 2.x IA-64
- Linux 2.x s390
- Mac OS X

Note: The preceding lists are not complete. We update them as we receive more information.

If you build MySQL from source with support for BDB tables, but the following error occurs when you start **mysqld**, it means BDB is not supported for your architecture:

```
bdb: architecture lacks fast mutexes: applications cannot be threaded
Can't init databases
```

In this case, you must rebuild MySQL without BDB table support or start the server with the `--skip-bdb` option.

14.4.2. Installing BDB

If you have downloaded a binary version of MySQL that includes support for Berkeley DB, simply fol-

low the usual binary distribution installation instructions. (MySQL-Max distributions include BDB support.)

If you build MySQL from source, you can enable BDB support by running **configure** with the `--with-berkeley-db` option in addition to any other options that you normally use. Download a distribution for MySQL 3.23.34 or newer, change location into its top-level directory, and run this command:

```
shell> ./configure --with-berkeley-db [other-options]
```

For more information, see Section 2.7, “Installing MySQL on Other Unix-Like Systems”, Section 5.1.2, “The **mysqld-max** Extended MySQL Server”, and See Section 2.8, “MySQL Installation Using a Source Distribution”.

14.4.3. BDB Startup Options

The following options to **mysqld** can be used to change the behavior of the BDB storage engine:

- `--bdb-home=path`

  The base directory for BDB tables. This should be the same directory you use for `--datadir`.

- `--bdb-lock-detect=method`

  The BDB lock detection method. The option value should be DEFAULT, OLDEST, RANDOM, or YOUNGEST.

- `--bdb-logdir=path`

  The BDB log file directory.

- `--bdb-no-recover`

  Don't start Berkeley DB in recover mode.

- `--bdb-no-sync`

  Don't synchronously flush the BDB logs.

- `--bdb-shared-data`

  Start Berkeley DB in multi-process mode. (Don't use DB_PRIVATE when initializing Berkeley DB.)

- `--bdb-tmpdir=path`

  The BDB temporary file directory.

- `--skip-bdb`

  Disable the BDB storage engine.

See Section 5.2.1, “**mysqld** Command-Line Options”.

The following system variable affects the behavior of BDB tables:

- `bdb_max_lock`

  The maximum number of locks you can have active on a `BDB` table.

See Section 5.2.3, “Server System Variables”.

If you use the `--skip-bdb` option, MySQL does not initialize the Berkeley DB library and this saves a lot of memory. However, if you use this option, you cannot use `BDB` tables. If you try to create a `BDB` table, MySQL creates a `MyISAM` table instead.

Normally, you should start **mysqld** without the `--bdb-no-recover` option if you intend to use `BDB` tables. However, this may give you problems when you try to start **mysqld** if the `BDB` log files are corrupted. See Section 2.9.2.3, “Starting and Troubleshooting the MySQL Server”.

With the `bdb_max_lock` variable, you can specify the maximum number of locks that can be active on a `BDB` table. The default is 10,000. You should increase this if errors such as the following occur when you perform long transactions or when **mysqld** has to examine many rows to execute a query:

```
bdb: Lock table is out of available locks
Got error 12 from ...
```

You may also want to change the `binlog_cache_size` and `max_binlog_cache_size` variables if you are using large multiple-statement transactions. See Section 5.9.4, “The Binary Log”.

14.4.4. Characteristics of BDB Tables

Each `BDB` table is stored on disk in two files. The files have names that begin with the table name and have an extension to indicate the file type. An `.frm` file stores the table definition, and a `.db` file contains the table data and indexes.

To specify explicitly that you want a `BDB` table, indicate that with an `ENGINE` or `TYPE` table option:

```
CREATE TABLE t (i INT) ENGINE = BDB;
CREATE TABLE t (i INT) TYPE = BDB;
```

`BerkeleyDB` is a synonym for `BDB` in the `ENGINE` or `TYPE` option.

The `BDB` storage engine provides transactional tables. The way you use these tables depends on the autocommit mode:

- If you are running with autocommit enabled (which is the default), changes to `BDB` tables are committed immediately and cannot be rolled back.
- If you are running with autocommit disabled, changes do not become permanent until you execute a `COMMIT` statement. Instead of committing, you can execute `ROLLBACK` to forget the changes.

  You can start a transaction with the `BEGIN WORK` statement to suspend autocommit, or with `SET AUTOCOMMIT=0` to disable autocommit explicitly.

See Section 13.4.1, “`START TRANSACTION`, `COMMIT`, and `ROLLBACK` Syntax”.

The `BDB` storage engine has the following characteristics:

- BDB tables can have up to 31 indexes per table, 16 columns per index, and a maximum key size of 1024 bytes (500 bytes before MySQL 4.0).
- MySQL requires a PRIMARY KEY in each BDB table so that each row can be uniquely identified. If you don't create one explicitly, MySQL creates and maintains a hidden PRIMARY KEY for you. The hidden key has a length of five bytes and is incremented for each insert attempt.
- The PRIMARY KEY is faster than any other index, because the PRIMARY KEY is stored together with the row data. The other indexes are stored as the key data + the PRIMARY KEY, so it's important to keep the PRIMARY KEY as short as possible to save disk space and get better speed.

  This behavior is similar to that of InnoDB, where shorter primary keys save space not only in the primary index but in secondary indexes as well.
- If all columns you access in a BDB table are part of the same index or part of the primary key, MySQL can execute the query without having to access the actual row. In a MyISAM table, this can be done only if the columns are part of the same index.
- Sequential scanning is slower than for MyISAM tables because the data in BDB tables is stored in B-trees and not in a separate data file.
- Key values are not prefix- or suffix-compressed like key values in MyISAM tables. In other words, key information takes a little more space in BDB tables compared to MyISAM tables.
- There are often holes in the BDB table to allow you to insert new rows in the middle of the index tree. This makes BDB tables somewhat larger than MyISAM tables.
- SELECT COUNT(*) FROM *tbl_name* is slow for BDB tables, because no row count is maintained in the table.
- The optimizer needs to know the approximate number of rows in the table. MySQL solves this by counting inserts and maintaining this in a separate segment in each BDB table. If you don't issue a lot of DELETE or ROLLBACK statements, this number should be accurate enough for the MySQL optimizer. However, MySQL stores the number only on close, so it may be incorrect if the server terminates unexpectedly. It should not be fatal even if this number is not 100% correct. You can update the row count by using ANALYZE TABLE or OPTIMIZE TABLE. See Section 13.5.2.1, “ANALYZE TABLE Syntax” and Section 13.5.2.5, “OPTIMIZE TABLE Syntax”.
- Internal locking in BDB tables is done at the page level.
- LOCK TABLES works on BDB tables as with other tables. If you don't use LOCK TABLE, MySQL issues an internal multiple-write lock on the table (a lock that doesn't block other writers) to ensure that the table is properly locked if another thread issues a table lock.
- To be able to roll back transactions, the BDB storage engine maintains log files. For maximum performance, you can use the --bdb-logdir option to place the BDB logs on a different disk than the one where your databases are located.
- MySQL performs a checkpoint each time a new BDB log file is started, and removes any BDB log files that are not needed for current transactions. You can also use FLUSH LOGS at any time to checkpoint the Berkeley DB tables.

  For disaster recovery, you should use table backups plus MySQL's binary log. See Section 5.7.1, “Database Backups”.

  Warning: If you delete old log files that are still in use, BDB is not able to do recovery at all and you may lose data if something goes wrong.
- Applications must always be prepared to handle cases where any change of a BDB table may cause

an automatic rollback and any read may fail with a deadlock error.

- If you get full disk with a BDB table, you get an error (probably error 28) and the transaction should roll back. This contrasts with MyISAM and ISAM tables, for which **mysqld** waits for enough free disk before continuing.

14.4.5. Things We Need to Fix for BDB

- It's very slow to open many BDB tables at the same time. If you are going to use BDB tables, you should not have a very large table cache (for example, with a size larger than 256) and you should use the --no-auto-rehash option when you use the **mysql** client. We plan to partly fix this in 4.0.
- SHOW TABLE STATUS doesn't yet provide very much information for BDB tables.
- Optimize performance.
- Change to not use page locks at all for table scanning operations.

14.4.6. Restrictions on BDB Tables

The following list indicates restrictions that you must observe when using BDB tables:

- Each BDB table stores in the .db file the path to the file as it was created. This was done to be able to detect locks in a multi-user environment that supports symlinks. However, the consequence is that BDB table files cannot be moved from one database directory to another.
- When making backups of BDB tables, you must either use **mysqldump** or else make a backup that includes the files for each BDB table (the .frm and .db files) as well as the BDB log files. The BDB storage engine stores unfinished transactions in its log files and requires them to be present when **mysqld** starts. The BDB logs are the files in the data directory with names of the form log.XXXXXXXXXX (ten digits).
- If a column that allows NULL values has a unique index, only a single NULL value is allowed. This differs from other storage engines.

14.4.7. Errors That May Occur When Using BDB Tables

- If the following error occurs when you start **mysqld**, it means that the new BDB version doesn't support the old log file format:

  ```
  bdb:  Ignoring log file: .../log.XXXXXXXXXX:
  unsupported log version #
  ```

  In this case, you must delete all BDB logs from your data directory (the files with names that have the format log.XXXXXXXXXX) and restart **mysqld**. We also recommend that you then use **mysqldump --opt** to dump your BDB tables, drop the tables, and restore them from the dump file.
- If autocommit mode is disabled and you drop a BDB table that is referenced in another transaction, you may get error messages of the following form in your MySQL error log:

```
001119 23:43:56  bdb:  Missing log fileid entry
001119 23:43:56  bdb:  txn_abort: Log undo failed for LSN:
                       1 3644744: Invalid
```

This is not fatal, but until the problem is fixed, we recommend that you not drop BDB tables except while autocommit mode is enabled. (The fix is not trivial.)

14.5. The EXAMPLE Storage Engine

The EXAMPLE storage engine was added in MySQL 4.1.3. It is a ``stub'' engine that does nothing. Its purpose is to serve as an example in the MySQL source code that illustrates how to begin writing new storage engines. As such, it is primarily of interest to developers.

To examine the source for the EXAMPLE engine, look in the sql/examples directory of a source distribution for MySQL 4.1.3 or newer.

To enable this storage engine, use the --with-example-storage-engine option to **configure** when you build MySQL.

When you create an EXAMPLE table, the server creates a table definition file in the database directory. The file begins with the table name and has an .frm extension. No other files are created. No data can be stored into the table or retrieved from it.

```
mysql> CREATE TABLE test (i INT) ENGINE = EXAMPLE;
Query OK, 0 rows affected (0.78 sec)

mysql> INSERT INTO test VALUES(1),(2),(3);
ERROR 1031 (HY000): Table storage engine for 'test' doesn't have this option

mysql> SELECT * FROM test;
Empty set (0.31 sec)
```

The EXAMPLE storage engine does not support indexing.

14.6. The FEDERATED Storage Engine

The FEDERATED storage engine was added in MySQL 5.0.3. It is a storage engine that accesses data in tables of remote databases rather than in local tables.

To examine the source for the FEDERATED engine, look in the sql directory of a source distribution for MySQL 5.0.3 or newer.

14.6.1. Installing the FEDERATED Storage Engine

To enable this storage engine, use the --with-federated-storage-engine option to **configure** when you build MySQL.

14.6.2. Description of the FEDERATED Storage Engine

When you create a FEDERATED table, the server creates a table definition file in the database directory. The file begins with the table name and has an .frm extension. No other files are created, because the actual data is in a remote database. This differs from the way that storage engines for local tables work.

For local database tables, data files are local. For example, if you create a MyISAM table named users, the MyISAM handler creates a data file named users.MYD. A handler for local tables reads, inserts, deletes, and updates data in local data files, and records are stored in a format particular to the handler. To read records, the handler must parse data into columns. To write records, column values must be converted to the row format used by the handler and written to the local data file.

With the MySQL FEDERATED storage engine, there are no local data files for a table (for example, there is no .MYD file). Instead, a remote database stores the data that normally would be in the table. This necessitates the use of the MySQL client API to read, delete, update, and insert data. Data retrieval is initiated via a SELECT * FROM *tbl_name* SQL statement. To read the result, rows are fetched one at a time by using the mysql_fetch_row() C API function, and then converted from the columns in the SELECT result set to the format that the FEDERATED handler expects.

The basic flow is as follows:

1. SQL calls issues locally
2. MySQL handler API (data in handler format)
3. MySQL client API (data converted to SQL calls)
4. Remote database -> MySQL client API
5. Convert result sets (if any) to handler format
6. Handler API -> Result rows or rows-affected count to local

14.6.3. How to use FEDERATED Tables

The procedure for using FEDERATED tables is very simple. Normally, you have two servers running, either both on the same host or on different hosts. (It is also possible for a FEDERATED table to use another table that is managed by the same server, though there is little point in doing so.)

First, you must have a table on the remote server that you want to access with the FEDERATED table. Suppose that the remote table is in the federated database and is defined like this:

```
CREATE TABLE test_table (
    id     int(20) NOT NULL auto_increment,
    name   varchar(32) NOT NULL default '',
    other  int(20) NOT NULL default '0',
    PRIMARY KEY  (id),
    KEY name (name),
    KEY other_key (other)
)
ENGINE=MyISAM
DEFAULT CHARSET=latin1 ;
```

The ENGINE table option could name any storage engine; the table need not be a MyISAM table.

Next, create a FEDERATED table for accessing the remote table. The server where you create the FEDERATED table is the ``client server.'' On this server, create the table as follows:

```
CREATE TABLE federated_table (
    id     int(20) NOT NULL auto_increment,
    name   varchar(32) NOT NULL default '',
    other  int(20) NOT NULL default '0',
    PRIMARY KEY  (id),
```

```
    KEY name (name),
    KEY other_key (other)
)
ENGINE=FEDERATED
DEFAULT CHARSET=latin1
COMMENT='mysql://root@remote_host:9306/federated/test_table';
```

The structure of this table must be exactly the same as the remote table, except that the `ENGINE` table option should be `FEDERATED` and the `COMMENT` table option is a connection string that indicates to the `FEDERATED` engine how to connect to the remote server.

The `FEDERATED` engine creates only the `test_table.frm` file in the `federated` database.

The remote host information indicates the remote server to which your ``client'' server connects, and the database and table information indicates which remote table to use as the ``data file.'' In the example, the remote server is indicated to be running as `remote_host` on port 9306, so you want to start that server so that it is indeed listening to port 9306.

The general form of the connection string in the `COMMENT` option is as follows:

```
scheme://user_name[:password]@host_name[:port_num]:/db_name/tbl_name
```

Only `mysql` is supported as the *`scheme`* at this point, and the password and port number are optional.

Here are some example connection strings:

```
COMMENT='mysql://username:password@hostname:port/database/tablename'
COMMENT='mysql://username@hostname/database/tablename'
COMMENT='mysql://username:password@hostname/database/tablename'
```

The use of `COMMENT` for specifying the connection string is non-optimal and will likely change in MySQL 5.1. Keep this in mind when you use `FEDERATED` tables, because it means you'll need to make some modifications when that happens.

Also, because a password is stored in the connection string as plain text, it can be seen by any user who can use `SHOW CREATE TABLE` or `SHOW TABLE STATUS` for the `FEDERATED` table.

14.6.4. Limitations of the FEDERATED Storage Engine

What the `FEDERATED` storage engine does and doesn't support:

- In the first version, the remote server must be a MySQL server. Support by `FEDERATED` for other database engines may be be added in the future.
- The remote table that a `FEDERATED` table points to *must* exist before you try to access the table through the `FEDERATED` table.
- It is possible for one `FEDERATED` table to point to another, but you must be careful not to create a loop. You know and have heard the screeching of audio feedback? You know what you see visually when you place two mirrors in front of each other, how the reflection continues for eternity? Well, need we say more?!
- There is no support for transactions.
- There is no way for the `FEDERATED` engine to know if the remote table has changed. The reason for this is that this table has to work like a data file that would never be written to by anything other than

the database. The integrity of the data in the local table could be breached if there was any change to the remote database.

- The FEDERATED storage engine supports SELECT, INSERT, UPDATE, DELETE, and indexes. It does not support ALTER TABLE, DROP TABLE, or any other Data Definition Language statements. The first implementation does not use Prepared statements. It remains to be seen whether the limited subset of the client API for the server supports this capability.
- The implementation uses SELECT, INSERT, UPDATE, DELETE and not HANDLER.
- FEDERATED tables do not work with the query cache.

Some of these limitations may be lifted in future versions of the FEDERATED handler.

14.7. The ARCHIVE Storage Engine

The ARCHIVE storage engine was added in MySQL 4.1.3. It is used for storing large amounts of data without indexes in a very small footprint.

To enable this storage engine, use the --with-archive-storage-engine option to **configure** when you build MySQL.

When you create an ARCHIVE table, the server creates a table definition file in the database directory. The file begins with the table name and has an .frm extension. The storage engine creates other files, all having names beginning with the table name. The data and metadata files have extensions of .ARZ and .ARM. An .ARN file may appear during optimization operations.

The ARCHIVE engine supports only INSERT and SELECT. (No deletes, replaces, or updates.) A SELECT performs a complete table scan. Records are compressed as they are inserted. Use of OPTIMIZE TABLE can analyze the table and pack it into a smaller format.

The ARCHIVE engine uses row-level locking.

14.8. The CSV Storage Engine

The CSV storage engine was added in MySQL 4.1.4. This engine stores data in text files using comma-separated-values format.

To enable this storage engine, use the --with-csv-storage-engine option to **configure** when you build MySQL.

When you create a CSV table, the server creates a table definition file in the database directory. The file begins with the table name and has an .frm extension. The storage engine also creates a data file. Its name begins with the table name and has a .CSV extension. The data file is a plain text file. When you store data into the table, the storage engine saves it into the data file in CSV format.

```
mysql> CREATE TABLE test(i INT, c CHAR(10)) ENGINE = CSV;
Query OK, 0 rows affected (0.12 sec)

mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM test;
+------+------------+
| i    | c          |
+------+------------+
```

```
|    1 | record one |
|    2 | record two |
+------+------------+
2 rows in set (0.00 sec)
```

If you examine the `test.CSV` file in the database directory after executing the preceding statements, its contents look like this:

```
"1","record one"
"2","record two"
```

The `CSV` storage engine does not support indexing.

14.9. The `ISAM` Storage Engine

The original storage engine in MySQL was the `ISAM` engine. It was the only storage engine available until MySQL 3.23, when the improved `MyISAM` engine was introduced as the default. `ISAM` is deprecated. As of MySQL 4.1, it's included in the source but not enabled in binary distributions. It is not available in MySQL 5.0. Embedded MySQL server versions do not support `ISAM` tables by default.

Due to the deprecated status of `ISAM`, and because `MyISAM` is an improvement over `ISAM`, you are advised to convert any remaining `ISAM` tables to `MyISAM` as soon as possible. To convert an `ISAM` table to a `MyISAM` table, use an `ALTER TABLE` statement:

```
mysql> ALTER TABLE tbl_name TYPE = MYISAM;
```

For more information about `MyISAM`, see Section 14.1, “The `MyISAM` Storage Engine”.

Each `ISAM` table is stored on disk in three files. The files have names that begin with the table name and have an extension to indicate the file type. An `.frm` file stores the table definition. The data file has an `.ISD` extension. The index file has an `.ISM` extension.

`ISAM` uses B-tree indexes.

You can check or repair `ISAM` tables with the **isamchk** utility. See Section 5.7.3.7, “Using **myisamchk** for Crash Recovery”.

`ISAM` has the following properties:

- Compressed and fixed-length keys
- Fixed and dynamic record length
- 16 indexes per table, with 16 key parts per key
- Maximum key length 256 bytes (default)
- Data values are stored in machine format; this is fast, but machine/OS dependent

Many of the properties of `MyISAM` tables are also true for `ISAM` tables. However, there are also many differences. The following list describes some of the ways that `ISAM` is distinct from `MyISAM`:

- Not binary portable across OS/platforms.

- Can't handle tables larger than 4GB.
- Only supports prefix compression on strings.
- Smaller (more restrictive) key limits.
- Dynamic tables become more fragmented.
- Doesn't support `MERGE` tables.
- Tables are checked and repaired with **isamchk** rather than with **myisamchk**.
- Tables are compressed with **pack_isam** rather than with **myisampack**.
- Cannot be used with the `BACKUP TABLE` or `RESTORE TABLE` backup-related statements.
- Cannot be used with the `CHECK TABLE`, `REPAIR TABLE`, `OPTIMIZE TABLE`, or `ANALYZE TABLE` table-maintenance statements.
- No support for full-text searching or spatial data types.
- No support for multiple character sets per table.
- Indexes cannot be assigned to specific key caches.

Chapter 15. The InnoDB Storage Engine

15.1. InnoDB Overview

InnoDB provides MySQL with a transaction-safe (ACID compliant) storage engine with commit, rollback, and crash recovery capabilities. InnoDB does locking on the row level and also provides an Oracle-style consistent non-locking read in SELECT statements. These features increase multi-user concurrency and performance. There is no need for lock escalation in InnoDB because row-level locks in InnoDB fit in very little space. InnoDB also supports FOREIGN KEY constraints. In SQL queries you can freely mix InnoDB type tables with other table types of MySQL, even within the same query.

InnoDB has been designed for maximum performance when processing large data volumes. Its CPU efficiency is probably not matched by any other disk-based relational database engine.

Fully integrated with MySQL Server, the InnoDB storage engine maintains its own buffer pool for caching data and indexes in main memory. InnoDB stores its tables and indexes in a tablespace, which may consist of several files (or raw disk partitions). This is different from, for example, MyISAM tables where each table is stored using separate files. InnoDB tables can be of any size even on operating systems where file size is limited to 2GB.

InnoDB is included in binary distributions by default as of MySQL 4.0. For information about InnoDB support in MySQL 3.23, see Section 15.3, "InnoDB in MySQL 3.23". Starting from MySQL 4.1.5, the new Windows installer makes InnoDB the MySQL default table type on Windows.

InnoDB is used in production at numerous large database sites requiring high performance. The famous Internet news site Slashdot.org runs on InnoDB. Mytrix, Inc. stores over 1TB of data in InnoDB, and another site handles an average load of 800 inserts/updates per second in InnoDB.

InnoDB is published under the same GNU GPL License Version 2 (of June 1991) as MySQL. For more information on MySQL licensing, see http://www.mysql.com/company/legal/licensing/.

15.2. InnoDB Contact Information

Contact information for Innobase Oy, producer of the InnoDB engine:

```
Web site: http://www.innodb.com/
Email: <sales@innodb.com>
Phone: +358-9-6969 3250 (office)
       +358-40-5617367 (mobile)

Innobase Oy Inc.
World Trade Center Helsinki
Aleksanterinkatu 17
P.O.Box 800
00101 Helsinki
Finland
```

15.3. InnoDB in MySQL 3.23

Beginning with MySQL 4.0, InnoDB is enabled by default, so the following information applies only to MySQL 3.23.

InnoDB tables are included in the MySQL source distribution starting from 3.23.34a and are activated in the MySQL-Max binaries of the 3.23 series. For Windows, the MySQL-Max binaries are included in the standard distribution.

If you have downloaded a binary version of MySQL that includes support for InnoDB, simply follow the instructions of the MySQL manual for installing a binary version of MySQL. If you have MySQL 3.23 installed, the simplest way to install MySQL-Max is to replace the executable **mysqld** server with the corresponding executable from the MySQL-Max distribution. MySQL and MySQL-Max differ only in the server executable. See Section 2.7, “Installing MySQL on Other Unix-Like Systems” and Section 5.1.2, “The **mysqld-max** Extended MySQL Server”.

To compile the MySQL source code with InnoDB support, download MySQL 3.23.34a or newer from http://www.mysql.com/ and configure MySQL with the `--with-innodb` option. See Section 2.8, “MySQL Installation Using a Source Distribution”.

To use InnoDB tables with MySQL 3.23, you must specify configuration parameters in the `[mysqld]` section of the `my.cnf` option file. On Windows, you can use `my.ini` instead. If you do not configure InnoDB in the option file, InnoDB does not start. (From MySQL 4.0 on, InnoDB uses default parameters if you do not specify any. However, to get best performance, it is still recommended that you use parameters appropriate for your system, as discussed in Section 15.4, “InnoDB Configuration”.)

In MySQL 3.23, you must specify at the minimum an `innodb_data_file_path` value to configure the InnoDB data files. For example, to configure InnoDB to use a single 500MB data file, place the following setting in the `[mysqld]` section of your option file:

```
[mysqld]
innodb_data_file_path=ibdata1:500M
```

InnoDB creates the `ibdata1` file in the MySQL data directory by default. To specify the location explicitly, specify an `innodb_data_home_dir` setting. See Section 15.4, “InnoDB Configuration”.

15.4. InnoDB Configuration

To enable InnoDB tables in MySQL 3.23, see Section 15.3, “InnoDB in MySQL 3.23”.

From MySQL 4.0 on, the InnoDB storage engine is enabled by default. If you don't want to use InnoDB tables, you can add the `skip-innodb` option to your MySQL option file.

Two important disk-based resources managed by the InnoDB storage engine are its tablespace data files and its log files.

If you specify no InnoDB configuration options, MySQL 4.0 and above creates an auto-extending 10MB data file named `ibdata1` and two 5MB log files named `ib_logfile0` and `ib_logfile1` in the MySQL data directory. (In MySQL 4.0.0 and 4.0.1, the data file is 64MB and not auto-extending.) In MySQL 3.23, InnoDB does not start if you provide no configuration options.

Note: To get good performance, you should explicitly provide InnoDB parameters as discussed in the following examples. Naturally, you should edit the settings to suit your hardware and requirements.

To set up the InnoDB tablespace files, use the `innodb_data_file_path` option in the `[mysqld]` section of the `my.cnf` option file. On Windows, you can use `my.ini` instead. The value of `innodb_data_file_path` should be a list of one or more data file specifications. If you name more than one data file, separate them by semicolon (`;`) characters:

```
innodb_data_file_path=datafile_spec1[;datafile_spec2]...
```

For example, a setting that explicitly creates a tablespace having the same characteristics as the MySQL 4.0 default is as follows:

```
[mysqld]
innodb_data_file_path=ibdata1:10M:autoextend
```

This setting configures a single 10MB data file named ibdata1 that is auto-extending. No location for the file is given, so the default is the MySQL data directory.

Sizes are specified using M or G suffix letters to indicate units of MB or GB.

A tablespace containing a fixed-size 50MB data file named ibdata1 and a 50MB auto-extending file named ibdata2 in the data directory can be configured like this:

```
[mysqld]
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

The full syntax for a data file specification includes the filename, its size, and several optional attributes:

file_name:*file_size*[:autoextend[:max:*max_file_size*]]

The autoextend attribute and those following can be used only for the last data file in the innodb_data_file_path line. autoextend is available starting from MySQL 3.23.50 and 4.0.2.

If you specify the autoextend option for the last data file, InnoDB extends the data file if it runs out of free space in the tablespace. The increment is 8MB at a time.

If the disk becomes full, you might want to add another data file on another disk. Instructions for reconfiguring an existing tablespace are given in Section 15.8, “Adding and Removing InnoDB Data and Log Files”.

InnoDB is not aware of the maximum file size, so be cautious on filesystems where the maximum file size is 2GB. To specify a maximum size for an auto-extending data file, use the max attribute. The following configuration allows ibdata1 to grow up to a limit of 500MB:

```
[mysqld]
innodb_data_file_path=ibdata1:10M:autoextend:max:500M
```

InnoDB creates tablespace files in the MySQL data directory by default. To specify a location explicitly, use the innodb_data_home_dir option. For example, to use two files named ibdata1 and ibdata2 but create them in the /ibdata directory, configure InnoDB like this:

```
[mysqld]
innodb_data_home_dir = /ibdata
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

Note: InnoDB does not create directories, so make sure that the /ibdata directory exists before you start the server. This is also true of any log file directories that you configure. Use the Unix or DOS mkdir command to create any necessary directories.

InnoDB forms the directory path for each data file by textually concatenating the value of innodb_data_home_dir to the data file name, adding a slash or backslash between if needed. If the innodb_data_home_dir option is not mentioned in my.cnf at all, the default value is the ``dot'' directory ./, which means the MySQL data directory.

If you specify innodb_data_home_dir as an empty string, you can specify absolute paths for the data files listed in the innodb_data_file_path value. The following example is equivalent to the preceding one:

```
[mysqld]
innodb_data_home_dir =
innodb_data_file_path=/ibdata/ibdata1:50M;/ibdata/ibdata2:50M:autoextend
```

A simple my.cnf example. Suppose that you have a computer with 128MB RAM and one hard disk. The following example shows possible configuration parameters in my.cnf or my.ini for InnoDB. The example assumes the use of MySQL-Max 3.23.50 or later or MySQL 4.0.2 or later because it uses the autoextend attribute.

This example suits most users, both on Unix and Windows, who do not want to distribute InnoDB data files and log files on several disks. It creates an auto-extending data file ibdata1 and two InnoDB log files ib_logfile0 and ib_logfile1 in the MySQL data directory. Also, the small archived InnoDB log file ib_arch_log_0000000000 that InnoDB creates automatically ends up in the data directory.

```
[mysqld]
# You can write your other MySQL server options here
# ...
# Data files must be able to hold your data and indexes.
# Make sure that you have enough free disk space.
innodb_data_file_path = ibdata1:10M:autoextend
#
# Set buffer pool size to 50-80% of your computer's memory
set-variable = innodb_buffer_pool_size=70M
set-variable = innodb_additional_mem_pool_size=10M
#
# Set the log file size to about 25% of the buffer pool size
set-variable = innodb_log_file_size=20M
set-variable = innodb_log_buffer_size=8M
#
innodb_flush_log_at_trx_commit=1
```

Make sure that the MySQL server has the proper access rights to create files in the data directory. More generally, the server must have access rights in any directory where it needs to create data files or log files.

Note that data files must be less than 2GB in some filesystems. The combined size of the log files must be less than 4GB. The combined size of data files must be at least 10MB.

When you create an InnoDB tablespace for the first time, it is best that you start the MySQL server from the command prompt. InnoDB then prints the information about the database creation to the screen, so you can see what is happening. For example, on Windows, if **mysqld-max** is located in C:\mysql\bin, you can start it like this:

```
C:\> C:\mysql\bin\mysqld-max --console
```

If you do not send server output to the screen, check the server's error log to see what InnoDB prints during the startup process.

See Section 15.6, “Creating the InnoDB Tablespace” for an example of what the information displayed by InnoDB should look like.

Where to specify options on Windows? The rules for option files on Windows are as follows:

- Only one of my.cnf or my.ini should be created.
- The my.cnf file should be placed in the root directory of the C: drive.
- The my.ini file should be placed in the WINDIR directory; for example, C:\WINDOWS or C:\WINNT. You can use the SET command at the command prompt in a console window to print the value of WINDIR:

  ```
  C:\> SET WINDIR
  windir=C:\WINNT
  ```

- If your PC uses a boot loader where the C: drive is not the boot drive, your only option is to use the my.ini file.
- If you installed MySQL 4.1.5 or higher using the installation and configuration wizards, the my.ini file is located in your MySQL installation directory. See Section 2.3.5.14, "The Location of the my.ini File".

Where to specify options on Unix? On Unix, **mysqld** reads options from the following files, if they exist, in the following order:

- /etc/my.cnf

  Global options.

- *$MYSQL_HOME*/my.cnf

  Server-specific options.

- defaults-extra-file

  The file specified with the --defaults-extra-file option.

- ~/.my.cnf

  User-specific options.

MYSQL_HOME represents an environment variable, which contains a path to the directory containing the server-specific my.cnf file.

If you want to make sure that **mysqld** reads options only from a specific file, you can use the --defaults-option as the first option on the command line when starting the server:

```
mysqld --defaults-file=your_path_to_my_cnf
```

An advanced my.cnf example. Suppose that you have a Linux computer with 2GB RAM and three 60GB hard disks (at directory paths /, /dr2 and /dr3). The following example shows possible configuration parameters in my.cnf for InnoDB.

```
[mysqld]
# You can write your other MySQL server options here
# ...
innodb_data_home_dir =
#
# Data files must be able to hold your data and indexes
innodb_data_file_path = /ibdata/ibdata1:2000M;/dr2/ibdata/ibdata2:2000M:autoextend
```

```
#
# Set buffer pool size to 50-80% of your computer's memory,
# but make sure on Linux x86 total memory usage is < 2GB
set-variable = innodb_buffer_pool_size=1G
set-variable = innodb_additional_mem_pool_size=20M
innodb_log_group_home_dir = /dr3/iblogs
#
# innodb_log_arch_dir must be the same as innodb_log_group_home_dir
# (starting from 4.0.6, you can omit it)
innodb_log_arch_dir = /dr3/iblogs
set-variable = innodb_log_files_in_group=2
#
# Set the log file size to about 25% of the buffer pool size
set-variable = innodb_log_file_size=250M
set-variable = innodb_log_buffer_size=8M
#
innodb_flush_log_at_trx_commit=1
set-variable = innodb_lock_wait_timeout=50
#
# Uncomment the next lines if you want to use them
#innodb_flush_method=fdatasync
#set-variable = innodb_thread_concurrency=5
```

Note that the example places the two data files on different disks. InnoDB fills the tablespace beginning with the first data file. In some cases, it improves the performance of the database if all data is not placed on the same physical disk. Putting log files on a different disk from data is very often beneficial for performance. You can also use raw disk partitions (raw devices) as InnoDB data files, which may speed up I/O. See Section 15.15.2, “Using Raw Devices for the Tablespace”.

Warning: On 32-bit GNU/Linux x86, you must be careful not to set memory usage too high. glibc may allow the process heap to grow over thread stacks, which crashes your server. It is a risk if the value of the following expression is close to or exceeds 2GB:

```
innodb_buffer_pool_size
+ key_buffer_size
+ max_connections*(sort_buffer_size+read_buffer_size+binlog_cache_size)
+ max_connections*2MB
```

Each thread uses a stack (often 2MB, but only 256KB in MySQL AB binaries) and in the worst case also uses sort_buffer_size + read_buffer_size additional memory.

Starting from MySQL 4.1, by compiling MySQL yourself, you can use up to 64GB of physical memory in 32-bit Windows. See the description for innodb_buffer_pool_awe_mem_mb in Section 15.5, “InnoDB Startup Options”.

How to tune other mysqld server parameters? The following values are typical and suit most users:

```
[mysqld]
skip-external-locking
set-variable = max_connections=200
set-variable = read_buffer_size=1M
set-variable = sort_buffer_size=1M
#
# Set key_buffer to 5 - 50% of your RAM depending on how much
# you use MyISAM tables, but keep key_buffer_size + InnoDB
# buffer pool size < 80% of your RAM
set-variable = key_buffer_size=...
```

15.5. InnoDB Startup Options

This section describes the InnoDB-related server options. In MySQL 4.0 and up, all of them can be specified in `--opt_name=value` form on the command line or in option files. Before MySQL 4.0, numeric options should be specified using `--set-variable=opt_name=value` or `-O opt_name=value` syntax.

- `innodb_additional_mem_pool_size`

  The size of a memory pool InnoDB uses to store data dictionary information and other internal data structures. The more tables you have in your application, the more memory you need to allocate here. If InnoDB runs out of memory in this pool, it starts to allocate memory from the operating system, and writes warning messages to the MySQL error log. The default value is 1MB.

- `innodb_autoextend_increment`

  The increment size (in megabytes) for extending the size of an autoextending tablespace when it becomes full. The default value is 8. This option is available starting from MySQL 4.0.24 and 4.1.5. As of MySQL 4.0.24 and 4.1.6, it can be changed at runtime as a global system variable.

- `innodb_buffer_pool_awe_mem_mb`

  The size of the buffer pool (in MB), if it is placed in the AWE memory of 32-bit Windows. Available from MySQL 4.1.0 and relevant only in 32-bit Windows. If your 32-bit Windows operating system supports more than 4GB memory, so-called ``Address Windowing Extensions," you can allocate the InnoDB buffer pool into the AWE physical memory using this parameter. The maximum possible value for this is 64000. If this parameter is specified, `innodb_buffer_pool_size` is the window in the 32-bit address space of **mysqld** where InnoDB maps that AWE memory. A good value for `innodb_buffer_pool_size` is 500MB.

- `innodb_buffer_pool_size`

  The size of the memory buffer InnoDB uses to cache data and indexes of its tables. The larger you set this value, the less disk I/O is needed to access data in tables. On a dedicated database server, you may set this to up to 80% of the machine physical memory size. However, do not set it too large because competition for the physical memory might cause paging in the operating system.

- `innodb_checksums`

  InnoDB uses checksum validation on all pages read from the disk to ensure extra fault tolerance against broken hardware or data files. However, under some rare circumstances (such as when running benchmarks) this ``extra safety" feature is unneeded. In such cases, this option (which is enabled by default) can be turned off with `--skip-innodb-checksums`. This option was added in MySQL 5.0.3.

- `innodb_data_file_path`

  The paths to individual data files and their sizes. The full directory path to each data file is acquired by concatenating `innodb_data_home_dir` to each path specified here. The file sizes are specified in megabytes or gigabytes (1024MB) by appending `M` or `G` to the size value. The sum of the sizes of the files must be at least 10MB. On some operating systems, files must be less than 2GB. If you do not specify `innodb_data_file_path`, the default behavior starting from 4.0 is to create a single 10MB auto-extending data file named `ibdata1`. Starting from 3.23.44, you can set the file size bigger than 4GB on those operating systems that support big files. You can also use raw disk partitions as data files. See Section 15.15.2, “Using Raw Devices for the Tablespace”.

- `innodb_data_home_dir`

The common part of the directory path for all InnoDB data files. If you do not set this value, the default is the MySQL data directory. You can specify this also as an empty string, in which case you can use absolute file paths in innodb_data_file_path.

- innodb_doublewrite

  By default, InnoDB stores all data twice, first to the doublewrite buffer, and then to the actual data files. This option can be used to disable this functionality. Like innodb_checksums, this option is enabled by default; it can be turned off with --skip-innodb-doublewrite for benchmarks or cases when top performance is needed rather than concern for data integrity or possible failures. This option was added in MySQL 5.0.3.

- innodb_fast_shutdown

  By default, InnoDB does a full purge and an insert buffer merge before a shutdown. These operations can take minutes, or even hours in extreme cases. If you set this parameter to 1, InnoDB skips these operations at shutdown. This option is available starting from MySQL 3.23.44 and 4.0.1. Its default value is 1 starting from 3.23.50.

- innodb_file_io_threads

  The number of file I/O threads in InnoDB. Normally this should be left at the default value of 4, but disk I/O on Windows may benefit from a larger number. On Unix, increasing the number has no effect; InnoDB always uses the default value. This option is available as of MySQL 3.23.37.

- innodb_file_per_table

  NOTE: CRITICAL BUG in 4.1 if you specify innodb_file_per_table in my.cnf! If you shut down **mysqld**, then records may disappear from the secondary indexes of a table. See (Bug #7496 [http://bugs.mysql.com/7496]) for more information and workarounds. This is fixed in 4.1.9, but another bug (Bug #8021 [http://bugs.mysql.com/8021]) bit the Windows version in 4.1.9, and in the Windows version of 4.1.9 you must put the line innodb_flush_method=unbuffered to your my.cnf or my.ini to get **mysqld** to work.

  This option causes InnoDB to create each new table using its own .ibd file for storing data and indexes, rather than in the shared tablespace. See Section 15.7.6, “Using Per-Table Tablespaces”. This option is available as of MySQL 4.1.1.

- innodb_flush_log_at_trx_commit

  When innodb_flush_log_at_trx_commit is set to 0, once per second the log buffer is written out to the log file, and the flush to disk operation is performed on the log file, but nothing is done at a transaction commit. When this value is 1 (the default), at each transaction commit the log buffer is written out to the log file, and the flush to disk operation is performed on the log file. When set to 2, at each commit the log buffer is written out to the file, but the flush to disk operation is not performed on it. However, the flushing on the log file takes place once per second also in the case of 2. We must note that the once-per-second flushing is not 100% guaranteed to happen every second, due to process scheduling issues. You can achieve better performance by setting the value different from 1, but then you can lose at most one second worth of transactions in a crash. If you set the value to 0, then any **mysqld** process crash can erase the last second of transactions. If you set the value to 2, then only an operating system crash or a power outage can erase the last second of transactions. Note that many operating systems and some disk hardware fool in the flush-to-disk operation. They may tell to **mysqld** that the flush has taken place, though it has not. Then the durability of transactions is not guaranteed even with the setting 1, and in the worst case a power outage can even corrupt the InnoDB database. Using a battery-backed disk cache in the SCSI disk controller or in the disk itself speeds up file flushes, and makes the operation safer. You can also try using the Unix command **hdparm** to disable the caching of disk writes in hardware caches, or use some other command specific to the hardware vendor. The default value of this option is 1 (prior to MySQL 4.0.13, the default is

0).

- innodb_flush_method

  This option is relevant only on Unix systems. If set to fdatasync, InnoDB uses fsync() to flush both the data and log files. If set to O_DSYNC, InnoDB uses O_SYNC to open and flush the log files, but uses fsync() to flush the data files. If O_DIRECT is specified (available on some GNU/Linux versions starting from MySQL 4.0.14), InnoDB uses O_DIRECT to open the data files, and uses fsync() to flush both the data and log files. Note that InnoDB does not use fdatasync or O_DSYNC by default because there have been problems with them on many Unix flavors. This option is available as of MySQL 3.23.40.

- innodb_force_recovery

  Warning: This option should be defined only in an emergency situation when you want to dump your tables from a corrupt database! Possible values are from 1 to 6. The meanings of these values are described in Section 15.9.1, “Forcing Recovery”. As a safety measure, InnoDB prevents a user from modifying data when this option is greater than 0. This option is available starting from MySQL 3.23.44.

- innodb_lock_wait_timeout

  The timeout in seconds an InnoDB transaction may wait for a lock before being rolled back. InnoDB automatically detects transaction deadlocks in its own lock table and rolls back the transaction. Beginning with MySQL 4.0.20 and 4.1.2, InnoDB notices locks set using the LOCK TABLES statement. Before that, if you use the LOCK TABLES statement, or other transaction-safe storage engines than InnoDB in the same transaction, a deadlock may arise that InnoDB cannot notice. In cases like this, the timeout is useful to resolve the situation. The default is 50 seconds.

- innodb_locks_unsafe_for_binlog

  This option turns off next-key locking in InnoDB searches and index scans. Default value for this option is false.

  Normally InnoDB uses an algorithm called ``next-key locking." InnoDB does the row-level locking in such a way that when it searches or scans an index of a table, it sets shared or exclusive locks on the index records it encounters. Thus the row-level locks are actually index record locks. The locks InnoDB sets on index records also affect the ``gap" before that index record. If a user has a shared or exclusive lock on record R in an index, another user cannot insert a new index record immediately before R in the index order. This option causes InnoDB not to use next-key locking in searches or index scans. Next-key locking is still used to ensure foreign key constraints and duplicate key checking. Note that using this option may cause phantom problems: Suppose that you want to read and lock all children from the child table with an identifier value larger than 100, with the intent of updating some column in the selected rows later:

  ```
  SELECT * FROM child WHERE id > 100 FOR UPDATE;
  ```

  Suppose that there is an index on the id column. The query scans that index starting from the first record where id is bigger than 100. If the locks set on the index records do not lock out inserts made in the gaps, a new row is meanwhile inserted to the table. If you execute the same SELECT within the same transaction, you see a new row in the result set returned by the query. This also means, that if new items are added to the database, InnoDB does not guarantee serializability instead conflict serializability is still guaranteed. Therefore, if this option is used InnoDB guarantees at most isolation level READ COMMITTED. This option is available as of MySQL 4.1.4.

  Starting from MySQL 5.0.2 this option is even more unsafe. InnoDB in an UPDATE or a DELETE only locks rows that it updates or deletes. This greatly reduces the probability of deadlocks but they

can happen. Note that this option still does not allow e.g. UPDATE to overtake other UPDATE even the case when both updates different rows. Consider following example:

```
CREATE TABLE A(A INT NOT NULL, B INT);
INSERT INTO A VALUES (1,2),(2,3),(3,2),(4,3),(5,2);
COMMIT;
```

If one connection executes a query:

```
SET AUTOCOMMIT = 0;
UPDATE A SET B = 5 WHERE B = 3;
```

and the other connection executes after the first one a query:

```
SET AUTOCOMMIT = 0;
UPDATE A SET B = 4 WHERE B = 2;
```

Then query two has to wait for a commit or rollback of query one, because query one has an exclusive lock to a row (2,3), and query two while scanning rows also tries to take an exclusive lock to the row (2,3) which it cannot have. This is because query two first takes an exclusive lock to a row and then checks does this row belong to the result set and if not then releases the unnecessary lock when option innodb_locks_unsafe_for_binlog is used.

Therefore, query one is executed as follows:

```
x-lock(1,2)
unlock(1,2)
x-lock(2,3)
update(2,3) to (2,5)
x-lock(3,2)
unlock(3,2)
x-lock(4,3)
update(4,3) to (4,5)
x-lock(5,2)
unlock(5,2)
```

and then query two is executed as follows:

```
x-lock(1,2)
update(1,2) to (1,4)
x-lock(2,3) - wait for query one to commit or rollback
```

- innodb_log_arch_dir

  The directory where fully written log files would be archived if we used log archiving. The value of this parameter should currently be set the same as innodb_log_group_home_dir. Starting from MySQL 4.0.6, you may omit this option.

- innodb_log_archive

  This value should currently be set to 0. Because recovery from a backup is done by MySQL using its own log files, there is currently no need to archive InnoDB log files. The default for this option is 0.

- innodb_log_buffer_size

  The size of the buffer that InnoDB uses to write to the log files on disk. Sensible values range from 1MB to 8MB. The default is 1MB. A large log buffer allows large transactions to run without a need to write the log to disk before the transactions commit. Thus, if you have big transactions, making the log buffer larger saves disk I/O.

- innodb_log_file_size

  The size of each log file in a log group. The combined size of log files must be less than 4GB on 32-bit computers. The default is 5MB. Sensible values range from 1MB to 1/*N*-th of the size of the buffer pool, below, where *N* is the number of log files in the group. The larger the value, the less checkpoint flush activity is needed in the buffer pool, saving disk I/O. But larger log files also mean that recovery is slower in case of a crash.

- innodb_log_files_in_group

  The number of log files in the log group. InnoDB writes to the files in a circular fashion. The default is 2 (recommended).

- innodb_log_group_home_dir

  The directory path to the InnoDB log files. It must have the same value as innodb_log_arch_dir. If you do not specify any InnoDB log parameters, the default is to create two 5MB files names ib_logfile0 and ib_logfile1 in the MySQL data directory.

- innodb_max_dirty_pages_pct

  This is an integer in the range from 0 to 100. The default is 90. The main thread in InnoDB tries to flush pages from the buffer pool so that at most this many percent of pages may not yet flushed been flushed at any particular time. Available starting from 4.0.13 and 4.1.1. If you have the SUPER privilege, this percentage can be changed while the server is running:

  ```
  SET GLOBAL innodb_max_dirty_pages_pct = value;
  ```

- innodb_max_purge_lag

  This option controls how to delay INSERT, UPDATE and DELETE operations when the purge operations are lagging. The default value of this parameter is zero, meaning that there are no delays. When the value is greater than zero, InnoDB may delay new row operations, as described in Section 15.13, “Implementation of Multi-Versioning”. This option can be changed at runtime as a global system variable. innodb_max_purge_lag is available as of MySQL 4.0.22 and 4.1.6.

- innodb_mirrored_log_groups

  The number of identical copies of log groups we keep for the database. Currently this should be set to 1.

- innodb_open_files

  This option is relevant only if you use multiple tablespaces in InnoDB. It specifies the maximum number of .ibd files that InnoDB can keep open at one time. The minimum value is 10. The default is 300. This option is available as of MySQL 4.1.1.

  The file descriptors used for .ibd files are for InnoDB only. They are independent of those specified by the --open-files-limit server option, and do not affect the operation of the table cache.

- innodb_table_locks

  Starting from MySQL 4.0.20, and 4.1.2, InnoDB honors LOCK TABLES; MySQL does not return from LOCK TABLE .. WRITE until all other threads have released all their locks to the table. In MySQL 4.0.19 and before, InnoDB ignored table locks, which allowed one to more easily simulate transactions with a combination of MyISAM and InnoDB tables. The default value is 1, which means that LOCK TABLES causes also InnoDB internally to take a table lock. In applications using AUTOCOMMIT=1, InnoDB's internal table locks can cause deadlocks. You can set innodb_table_locks=0 in my.cnf to remove that problem.

- innodb_thread_concurrency

  InnoDB tries to keep the number of operating system threads concurrently inside InnoDB less than or equal to the limit given by this parameter. The default value is 8. If you have low performance and SHOW INNODB STATUS reveals many threads waiting for semaphores, you may have thread thrashing and should try setting this parameter lower or higher. If you have a computer with many processors and disks, you can try setting the value higher to better utilize the resources of you computer. A recommended value is the sum of the number of processors and disks your system has. A value of 500 or greater disables the concurrency checking. This option is available starting from MySQL 3.23.44 and 4.0.1.

- innodb_status_file

  This option causes InnoDB to create a file <*datadir*>/innodb_status.<*pid*> for periodical SHOW INNODB STATUS output. This option is available as of MySQL 4.0.21.

15.6. Creating the InnoDB Tablespace

Suppose that you have installed MySQL and have edited your option file so that it contains the necessary InnoDB configuration parameters. Before starting MySQL, you should verify that the directories you have specified for InnoDB data files and log files exist and that the MySQL server has access rights to those directories. InnoDB cannot create directories, only files. Check also that you have enough disk space for the data and log files.

It is best to run the MySQL server **mysqld** from the command prompt when you create an InnoDB database, not from the **mysqld_safe** wrapper or as a Windows service. When you run from a command prompt you see what **mysqld** prints and what is happening. On Unix, just invoke **mysqld**. On Windows, use the --console option.

When you start the MySQL server after initially configuring InnoDB in your option file, InnoDB creates your data files and log files. InnoDB prints something like the following:

```
InnoDB: The first specified datafile /home/heikki/data/ibdata1
did not exist:
InnoDB: a new database to be created!
InnoDB: Setting file /home/heikki/data/ibdata1 size to 134217728
InnoDB: Database physically writes the file full: wait...
InnoDB: datafile /home/heikki/data/ibdata2 did not exist:
new to be created
InnoDB: Setting file /home/heikki/data/ibdata2 size to 262144000
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file /home/heikki/data/logs/ib_logfile0 did not exist:
new to be created
InnoDB: Setting log file /home/heikki/data/logs/ib_logfile0 size
to 5242880
```

```
InnoDB: Log file /home/heikki/data/logs/ib_logfile1 did not exist:
new to be created
InnoDB: Setting log file /home/heikki/data/logs/ib_logfile1 size
to 5242880
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: Creating foreign key constraint system tables
InnoDB: Foreign key constraint system tables created
InnoDB: Started
mysqld: ready for connections
```

A new InnoDB database has been created. You can connect to the MySQL server with the usual MySQL client programs like **mysql**. When you shut down the MySQL server with **mysqladmin shutdown**, the output is like the following:

```
010321 18:33:34  mysqld: Normal shutdown
010321 18:33:34  mysqld: Shutdown Complete
InnoDB: Starting shutdown...
InnoDB: Shutdown completed
```

You can look at the data file and log directories and you see the files created. The log directory also contains a small file named ib_arch_log_0000000000. That file resulted from the database creation, after which InnoDB switched off log archiving. When MySQL is started again, the data files and log files have been created, so the output is much briefer:

```
InnoDB: Started
mysqld: ready for connections
```

Starting from MySQL 4.1.1, you can add the option innodb_file_per_table to my.cnf, and make InnoDB to store each table into its own .ibd file in a database directory of MySQL. See Section 15.7.6, “Using Per-Table Tablespaces”.

15.6.1. Dealing with InnoDB Initialization Problems

If InnoDB prints an operating system error in a file operation, usually the problem is one of the following:

- You did not create the InnoDB data file directory or the InnoDB log directory.
- **mysqld** does not have access rights to create files in those directories.
- **mysqld** cannot not read the proper my.cnf or my.ini option file, and consequently does not see the options you specified.
- The disk is full or a disk quota is exceeded.
- You have created a subdirectory whose name is equal to a data file you specified.
- There is a syntax error in innodb_data_home_dir or innodb_data_file_path.

If something goes wrong when InnoDB attempts to initialize its tablespace or its log files, you should delete all files created by InnoDB. This means all ibdata files and all ib_logfiles. In case you created some InnoDB tables, delete the corresponding .frm files for these tables (and any .ibd files if you are using multiple tablespaces) from the MySQL database directories as well. Then you can try the InnoDB database creation again. It is best to start the MySQL server from a command prompt so

that you see what is happening.

15.7. Creating InnoDB Tables

Suppose that you have started the MySQL client with the command `mysql test`. To create an InnoDB table, you must specify and `ENGINE = InnoDB` or `TYPE = InnoDB` option in the table creation SQL statement:

```
CREATE TABLE customers (a INT, b CHAR (20), INDEX (a)) ENGINE=InnoDB;
CREATE TABLE customers (a INT, b CHAR (20), INDEX (a)) TYPE=InnoDB;
```

The SQL statement creates a table and an index on column `a` in the InnoDB tablespace that consists of the data files you specified in `my.cnf`. In addition, MySQL creates a file `customers.frm` in the `test` directory under the MySQL database directory. Internally, InnoDB adds to its own data dictionary an entry for table `'test/customers'`. This means you can create a table of the same name `customers` in some other database, and the table names do not collide inside InnoDB.

You can query the amount of free space in the InnoDB tablespace by issuing a `SHOW TABLE STATUS` statement for any InnoDB table. The amount of free space in the tablespace appears in the `Comment` section in the output of `SHOW TABLE STATUS`. An example:

```
SHOW TABLE STATUS FROM test LIKE 'customers'
```

Note that the statistics `SHOW` gives about InnoDB tables are only approximate. They are used in SQL optimization. Table and index reserved sizes in bytes are accurate, though.

15.7.1. How to Use Transactions in InnoDB with Different APIs

By default, each client that connects to the MySQL server begins with autocommit mode enabled, which automatically commits every SQL statement you run. To use multiple-statement transactions, you can switch autocommit off with the SQL statement `SET AUTOCOMMIT = 0` and use `COMMIT` and `ROLLBACK` to commit or roll back your transaction. If you want to leave autocommit on, you can enclose your transactions between `START TRANSACTION` and `COMMIT` or `ROLLBACK`. Before MySQL 4.0.11, you have to use the keyword `BEGIN` instead of `START TRANSACTION`. The following example shows two transactions. The first is committed and the second is rolled back.

```
shell> mysql test
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5 to server version: 3.23.50-log
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> CREATE TABLE CUSTOMER (A INT, B CHAR (20), INDEX (A))
    -> TYPE=InnoDB;
Query OK, 0 rows affected (0.00 sec)
mysql> BEGIN;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO CUSTOMER VALUES (10, 'Heikki');
Query OK, 1 row affected (0.00 sec)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
mysql> SET AUTOCOMMIT=0;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO CUSTOMER VALUES (15, 'John');
Query OK, 1 row affected (0.00 sec)
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM CUSTOMER;
```

```
+------+--------+
| A    | B      |
+------+--------+
|   10 | Heikki |
+------+--------+
1 row in set (0.00 sec)
mysql>
```

In APIs like PHP, Perl DBI/DBD, JDBC, ODBC, or the standard C call interface of MySQL, you can send transaction control statements such as COMMIT to the MySQL server as strings just like any other SQL statements such as SELECT or INSERT. Some APIs also offer separate special transaction commit and rollback functions or methods.

15.7.2. Converting MyISAM Tables to InnoDB

Important: You should not convert MySQL system tables in the mysql database (such as user or host) to the InnoDB type. The system tables must always be of the MyISAM type.

If you want all your (non-system) tables to be created as InnoDB tables, you can, starting from the MySQL 3.23.43, add the line default-table-type=innodb to the [mysqld] section of your my.cnf or my.ini file.

InnoDB does not have a special optimization for separate index creation the way the MyISAM storage engine does. Therefore, it does not pay to export and import the table and create indexes afterward. The fastest way to alter a table to InnoDB is to do the inserts directly to an InnoDB table. That is, use ALTER TABLE ... TYPE=INNODB, or create an empty InnoDB table with identical definitions and insert the rows with INSERT INTO ... SELECT * FROM ....

If you have UNIQUE constraints on secondary keys, starting from MySQL 3.23.52, you can speed up a table import by turning off the uniqueness checks temporarily during the import session: SET UNIQUE_CHECKS=0; For big tables, this saves a lot of disk I/O because InnoDB can then use its insert buffer to write secondary index records in a batch.

To get better control over the insertion process, it might be good to insert big tables in pieces:

```
INSERT INTO newtable SELECT * FROM oldtable
   WHERE yourkey > something AND yourkey <= somethingelse;
```

After all records have been inserted, you can rename the tables.

During the conversion of big tables, you should increase the size of the InnoDB buffer pool to reduce disk I/O. Do not use more than 80% of the physical memory, though. You can also increase the sizes of the InnoDB log files and the log files.

Make sure that you do not fill up the tablespace: InnoDB tables require a lot more disk space than MyISAM tables. If an ALTER TABLE runs out of space, it starts a rollback, and that can take hours if it is disk-bound. For inserts, InnoDB uses the insert buffer to merge secondary index records to indexes in batches. That saves a lot of disk I/O. In rollback, no such mechanism is used, and the rollback can take 30 times longer than the insertion.

In the case of a runaway rollback, if you do not have valuable data in your database, it may be advisable to kill the database process rather than wait for millions of disk I/O operations to complete. For the complete procedure, see Section 15.9.1, “Forcing Recovery”.

15.7.3. How an AUTO_INCREMENT Column Works in In-

noDB

If you specify an AUTO_INCREMENT column for a table, the InnoDB table handle in the data dictionary contains a special counter called the auto-increment counter that is used in assigning new values for the column. The auto-increment counter is stored only in main memory, not on disk.

InnoDB uses the following algorithm to initialize the auto-increment counter for a table T that contains an AUTO_INCREMENT column named ai_col: After a server startup, when a user first does an insert to a table T, InnoDB executes the equivalent of this statement:

```
SELECT MAX(ai_col) FROM T FOR UPDATE;
```

The value retrieved by the statement is incremented by one and assigned to the column and the auto-increment counter of the table. If the table is empty, the value 1 is assigned. If the auto-increment counter is not initialized and the user invokes a SHOW TABLE STATUS statement that displays output for the table T, the counter is initialized (but not incremented) and stored for use by later inserts. Note that in this initialization we do a normal exclusive-locking read on the table and the lock lasts to the end of the transaction.

InnoDB follows the same procedure for initializing the auto-increment counter for a freshly created table.

Note that if the user specifies NULL or 0 for the AUTO_INCREMENT column in an INSERT, InnoDB treats the row as if the value had not been specified and generates a new value for it.

After the auto-increment counter has been initialized, if a user inserts a row that explicitly specifies the column value, and the value is bigger than the current counter value, the counter is set to the specified column value. If the user does not explicitly specify a value, InnoDB increments the counter by one and assigns the new value to the column.

When accessing the auto-increment counter, InnoDB uses a special table level AUTO-INC lock that it keeps to the end of the current SQL statement, not to the end of the transaction. The special lock release strategy was introduced to improve concurrency for inserts into a table containing an AUTO_INCREMENT column. Two transactions cannot have the AUTO-INC lock on the same table simultaneously.

Note that you may see gaps in the sequence of values assigned to the AUTO_INCREMENT column if you roll back transactions that have gotten numbers from the counter.

The behavior of the auto-increment mechanism is not defined if a user assigns a negative value to the column or if the value becomes bigger than the maximum integer that can be stored in the specified integer type.

15.7.4. FOREIGN KEY Constraints

Starting from MySQL 3.23.44, InnoDB features foreign key constraints.

The syntax of a foreign key constraint definition in InnoDB looks like this:

```
[CONSTRAINT symbol] FOREIGN KEY [id] (index_col_name, ...)
    REFERENCES tbl_name (index_col_name, ...)
    [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT}]
    [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT}]
```

Foreign keys definitions are subject to the following conditions:

- Both tables must be InnoDB type.
- In the referencing table, there must be an index where the foreign key columns are listed as the *first* columns in the same order.
- In the referenced table, there must be an index where the referenced columns are listed as the *first* columns in the same order.
- Index prefixes on foreign key columns are not supported. One consequence of this is that BLOB and TEXT columns cannot be included in a foreign key, because indexes on those columns must always include a prefix length.

InnoDB needs indexes on foreign keys and referenced keys so that foreign key checks can be fast and not require a table scan. Starting with MySQL 4.1.2, these indexes are created automatically. In older versions, the indexes must be created explicitly or the creation of foreign key constraints fails.

Corresponding columns in the foreign key and the referenced key must have similar internal data types inside InnoDB so that they can be compared without a type conversion. The **size and the signedness of integer types has to be the same**. The length of string types need not be the same. If you specify a SET NULL action, make sure that you have **not declared the columns in the child table** as NOT NULL.

If MySQL reports an error number 1005 from a CREATE TABLE statement, and the error message string refers to errno 150, this means that the table creation failed because a foreign key constraint was not correctly formed. Similarly, if an ALTER TABLE fails and it refers to errno 150, that means a foreign key definition would be incorrectly formed for the altered table. Starting from MySQL 4.0.13, you can use SHOW INNODB STATUS to display a detailed explanation of the latest InnoDB foreign key error in the server.

Starting from MySQL 3.23.50, InnoDB does not check foreign key constraints on those foreign key or referenced key values that contain a NULL column.

A deviation from SQL standards: If in the parent table there are several rows that have the same referenced key value, then InnoDB acts in foreign key checks as if the other parent rows with the same key value do not exist. For example, if you have defined a RESTRICT type constraint, and there is a child row with several parent rows, InnoDB does not allow the deletion of any of those parent rows.

Starting from MySQL 3.23.50, you can also associate the ON DELETE CASCADE or ON DELETE SET NULL clause with the foreign key constraint. Corresponding ON UPDATE options are available starting from 4.0.8. If ON DELETE CASCADE is specified, and a row in the parent table is deleted, InnoDB automatically deletes also all those rows in the child table whose foreign key values are equal to the referenced key value in the parent row. If ON DELETE SET NULL is specified, the child rows are automatically updated so that the columns in the foreign key are set to the SQL NULL value. SET DEFAULT is parsed but ignored by MySQL. InnoDB rejects table definitions containing ON DELETE SET DEFAULT clauses.

InnoDB performs cascading operations through a depth-first algorithm, based on records in the indexes corresponding to the foreign key constraints.

A deviation from SQL standards: If ON UPDATE CASCADE or ON UPDATE SET NULL recurses to update the *same table* it has previously updated during the cascade, it acts like RESTRICT. This means that you cannot use self-referential ON UPDATE CASCADE or ON UPDATE SET NULL operations. This is to prevent infinite loops resulting from cascaded updates. A self-referential ON DELETE SET NULL, on the other hand, is possible from 4.0.13. A self-referential ON DELETE CASCADE has been possible since ON DELETE was implemented. Since 4.0.21, cascading operations may not be nested more than 15 levels.

A deviation from SQL standards: Like MySQL in general, in an SQL statement that inserts, deletes, or updates many rows, InnoDB checks UNIQUE and FOREIGN KEY constraints row-by-row. Accord-

ing to the SQL standard, the default behavior should be that constraints are only checked after the WHOLE SQL statement has been processed.

A simple example that relates parent and child tables through a single-column foreign key:

```
CREATE TABLE parent(id INT NOT NULL,
                     PRIMARY KEY (id)
) TYPE=INNODB;
CREATE TABLE child(id INT, parent_id INT,
                   INDEX par_ind (parent_id),
                   FOREIGN KEY (parent_id) REFERENCES parent(id)
                     ON DELETE CASCADE
) TYPE=INNODB;
```

A more complex example in which a product_order table has foreign keys for two other tables. One foreign key references a two-column index in the product table. The other references a single-column index in the customer table:

```
CREATE TABLE product (category INT NOT NULL, id INT NOT NULL,
                      price DECIMAL,
                      PRIMARY KEY(category, id)) TYPE=INNODB;
CREATE TABLE customer (id INT NOT NULL,
                      PRIMARY KEY (id)) TYPE=INNODB;
CREATE TABLE product_order (no INT NOT NULL AUTO_INCREMENT,
                      product_category INT NOT NULL,
                      product_id INT NOT NULL,
                      customer_id INT NOT NULL,
                      PRIMARY KEY(no),
                      INDEX (product_category, product_id),
                      FOREIGN KEY (product_category, product_id)
                        REFERENCES product(category, id)
                        ON UPDATE CASCADE ON DELETE RESTRICT,
                      INDEX (customer_id),
                      FOREIGN KEY (customer_id)
                        REFERENCES customer(id)) TYPE=INNODB;
```

Starting from MySQL 3.23.50, InnoDB allows you to add a new foreign key constraint to a table by using ALTER TABLE:

```
ALTER TABLE yourtablename
    ADD [CONSTRAINT symbol] FOREIGN KEY [id] (index_col_name, ...)
    REFERENCES tbl_name (index_col_name, ...)
    [ON DELETE {RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT}]
    [ON UPDATE {RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT}]
```

Remember to create the required indexes first. You can also add a self-referential foreign key constraint to a table using ALTER TABLE.

Starting from MySQL 4.0.13, InnoDB supports the use of ALTER TABLE to drop foreign keys:

```
ALTER TABLE yourtablename DROP FOREIGN KEY fk_symbol;
```

If the FOREIGN KEY clause included a CONSTRAINT name when you created the foreign key, you can refer to that name to drop the foreign key. (A constraint name can be given as of MySQL 4.0.18.) Otherwise, the fk_symbol value is internally generated by InnoDB when the foreign key is created. To find out the symbol when you want to drop a foreign key, use the SHOW CREATE TABLE statement. An example:

```
mysql> SHOW CREATE TABLE ibtest11c\G
```

```
*************************** 1. row ***************************
       Table: ibtest11c
Create Table: CREATE TABLE `ibtest11c` (
  `A` int(11) NOT NULL auto_increment,
  `D` int(11) NOT NULL default '0',
  `B` varchar(200) NOT NULL default '',
  `C` varchar(175) default NULL,
  PRIMARY KEY  (`A`,`D`,`B`),
  KEY `B` (`B`,`C`),
  KEY `C` (`C`),
  CONSTRAINT `0_38775` FOREIGN KEY (`A`, `D`)
REFERENCES `ibtest11a` (`A`, `D`)
ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `0_38776` FOREIGN KEY (`B`, `C`)
REFERENCES `ibtest11a` (`B`, `C`)
ON DELETE CASCADE ON UPDATE CASCADE
) TYPE=InnoDB CHARSET=latin1
1 row in set (0.01 sec)

mysql> ALTER TABLE ibtest11c DROP FOREIGN KEY 0_38775;
```

Starting from MySQL 3.23.50, the InnoDB parser allows you to use backticks around table and column names in a FOREIGN KEY ... REFERENCES ... clause. Starting from MySQL 4.0.5, the InnoDB parser also takes into account the lower_case_table_names system variable setting.

Before MySQL 3.23.50, ALTER TABLE or CREATE INDEX should not be used in connection with tables that have foreign key constraints or that are referenced in foreign key constraints: Any ALTER TABLE removes all foreign key constraints defined for the table. You should not use ALTER TABLE with the referenced table, either. Instead, use DROP TABLE and CREATE TABLE to modify the schema. When MySQL does an ALTER TABLE it may internally use RENAME TABLE, and that confuses the foreign key constraints that refer to the table. In MySQL, a CREATE INDEX statement is processed as an ALTER TABLE, so the same considerations apply.

Starting from MySQL 3.23.50, InnoDB returns the foreign key definitions of a table as part of the output of the SHOW CREATE TABLE statement:

```
SHOW CREATE TABLE tbl_name;
```

From this version, **mysqldump** also produces correct definitions of tables to the dump file, and does not forget about the foreign keys.

You can display the foreign key constraints for a table like this:

```
SHOW TABLE STATUS FROM db_name LIKE 'tbl_name'
```

The foreign key constraints are listed in the Comment column of the output.

When performing foreign key checks, InnoDB sets shared row level locks on child or parent records it has to look at. InnoDB checks foreign key constraints immediately; the check is not deferred to transaction commit.

To make it easier to reload dump files for tables that have foreign key relationships, **mysqldump** automatically includes a statement in the dump output to set FOREIGN_KEY_CHECKS to 0 as of MySQL 4.1.1. This avoids problems with tables having to be reloaded in a particular order when the dump is reloaded. For earlier versions, you can disable the variable manually within **mysql** when loading the dump file like this:

```
mysql> SET FOREIGN_KEY_CHECKS = 0;
```

```
mysql> SOURCE dump_file_name
mysql> SET FOREIGN_KEY_CHECKS = 1;
```

This allows you to import the tables in any order if the dump file contains tables that are not correctly ordered for foreign keys. It also speeds up the import operation. FOREIGN_KEY_CHECKS is available starting from MySQL 3.23.52 and 4.0.3.

Setting FOREIGN_KEY_CHECKS to 0 can also be useful for ignoring foreign key constraints during LOAD DATA operations.

InnoDB does not allow you to drop a table that is referenced by a FOREIGN KEY constraint, unless you do SET FOREIGN_KEY_CHECKS=0. When you drop a table, the constraints that were defined in its create statement are also dropped.

If you re-create a table that was dropped, it must have a definition that conforms to the foreign key constraints referencing it. It must have the right column names and types, and it must have indexes on the referenced keys, as stated earlier. If these are not satisfied, MySQL returns error number 1005 and refers to errno 150 in the error message string.

15.7.5. InnoDB and MySQL Replication

MySQL replication works for InnoDB tables as it does for MyISAM tables. It is also possible to use replication in a way where the table type on the slave is not the same as the original table type on the master. For example, you can replicate modifications to an InnoDB table on the master to a MyISAM table on the slave.

To set up a new slave for a master, you have to make a copy of the InnoDB tablespace and the log files, as well as the .frm files of the InnoDB tables, and move the copies to the slave. For the proper procedure to do this, see Section 15.10, “Moving an InnoDB Database to Another Machine”.

If you can shut down the master or an existing slave, you can take a cold backup of the InnoDB tablespace and log files and use that to set up a slave. To make a new slave without taking down any server you can also use the non-free (commercial) InnoDB Hot Backup tool [http://www.innodb.com/order.html].

There are minor limitations in InnoDB replication:

- LOAD TABLE FROM MASTER does not work for InnoDB type tables. There are workarounds: 1) dump the table on the master and import the dump file into the slave, or 2) use ALTER TABLE *tbl_name* TYPE=MyISAM on the master before setting up replication with LOAD TABLE *tbl_name* FROM MASTER, and then use ALTER TABLE to alter the master table back to the InnoDB type afterward.
- Before MySQL 4.0.6, SLAVE STOP did not respect the boundary of a multiple-statement transaction. An incomplete transaction would be rolled back, and the next SLAVE START would only execute the remaining part of the half transaction. That would cause replication to fail.
- Before MySQL 4.0.6, a slave crash in the middle of a multiple-statement transaction would cause the same problem as SLAVE STOP.
- Before MySQL 4.0.11, replication of the SET FOREIGN_KEY_CHECKS=0 statement does not work properly.

Most of these limitations can be eliminated by using more recent server versions for which the limitations do not apply.

Transactions that fail on the master do not affect replication at all. MySQL replication is based on the binary log where MySQL writes SQL statements that modify data. A slave reads the binary log of the master and executes the same SQL statements. However, statements that occur within a transaction are not written to the binary log until the transaction commits, at which point all statements in the transaction are written at once. If a statement fails, for example, because of a foreign key violation, or if a transaction is rolled back, no SQL statements are written to the binary log, and the transaction is not executed on the slave at all.

15.7.6. Using Per-Table Tablespaces

NOTE: CRITICAL BUG in 4.1 if you specify `innodb_file_per_table` in `my.cnf`! If you shut down **mysqld**, then records may disappear from the secondary indexes of a table. See (Bug #7496 [http://bugs.mysql.com/7496]) for more information and workarounds. This is fixed in 4.1.9, but another bug (Bug #8021 [http://bugs.mysql.com/8021]) bit the Windows version in 4.1.9, and in the Windows version of 4.1.9 you must put the line `innodb_flush_method=unbuffered` to your `my.cnf` or `my.ini` to get **mysqld** to work.

Starting from MySQL 4.1.1, you can store each `InnoDB` table and its indexes into its own file. This feature is called ``multiple tablespaces'' because in effect each table has its own tablespace.

Using multiple tablespaces can be beneficial to users who want to move specific tables to separate physical disks or who wish to restore backups of single tables quickly without interrupting the use of the remaining InnoDB tables.

If you need to downgrade to 4.0, you have to take table dumps and re-create the whole `InnoDB` tablespace. If you have not created new `InnoDB` tables under MySQL 4.1.1 or later, and need to downgrade quickly, you can also do a direct downgrade to the MySQL 4.0.18 or later in the 4.0 series. Before doing the direct downgrade to 4.0.x, you have to end all client connections to the **mysqld** server that is to be downgraded, and let it run the purge and insert buffer merge operations to completion, so that `SHOW INNODB STATUS` shows the main thread in the state `waiting for server activity`. Then you can shut down **mysqld** and start 4.0.18 or later in the 4.0 series.

You can enable multiple tablespaces by adding a line to the `[mysqld]` section of `my.cnf`:

```
[mysqld]
innodb_file_per_table
```

After restarting the server, `InnoDB` stores each newly created table into its own file *`tbl_name`*`.ibd` in the database directory where the table belongs. This is similar to what the `MyISAM` storage engine does, but `MyISAM` divides the table into a data file *`tbl_name`*`.MYD` and the index file *`tbl_name`*`.MYI`. For `InnoDB`, the data and the indexes are stored together in the `.ibd` file. The *`tbl_name`*`.frm` file is still created as usual.

If you remove the `innodb_file_per_table` line from `my.cnf` and restart the server, `InnoDB` creates tables inside the shared tablespace files again.

`innodb_file_per_table` affects only table creation. If you start the server with this option, new tables are created using `.ibd` files, but you can still access tables that exist in the shared tablespace. If you remove the option, new tables are created in the shared tablespace, but you can still access any tables that were created using multiple tablespaces.

`InnoDB` always needs the shared tablespace. The `.ibd` files are not sufficient for `InnoDB` to operate. The shared tablespace consists of the familiar `ibdata` files where `InnoDB` puts its internal data dictionary and undo logs.

You cannot freely move `.ibd` files around between database directories the way you can with `MyISAM` table files. This is because the table definition is stored in the `InnoDB` shared tablespace, and also

because InnoDB must preserve the consistency of transaction IDs and log sequence numbers.

Within a given MySQL installation, you can move an .ibd file and the associated table from one database to another with the familiar RENAME TABLE statement:

```
RENAME TABLE old_db_name.tbl_name TO new_db_name.tbl_name;
```

If you have a ``clean'' backup of an .ibd file, you can restore it to the MySQL installation from which it originated as follows:

1. Issue this ALTER TABLE statement:

   ```
   ALTER TABLE tbl_name DISCARD TABLESPACE;
   ```

   Caution: This deletes the current .ibd file.

2. Put the backup .ibd file back in the proper database directory.

3. Issue this ALTER TABLE statement:

   ```
   ALTER TABLE tbl_name IMPORT TABLESPACE;
   ```

In this context, a ``clean'' .ibd file backup means:

- There are no uncommitted modifications by transactions in the .ibd file.
- There are no unmerged insert buffer entries in the .ibd file.
- Purge has removed all delete-marked index records from the .ibd file.
- **mysqld** has flushed all modified pages of the .ibd file from the buffer pool to the file.

You can make such a clean backup .ibd file with the following method:

1. Stop all activity from the **mysqld** server and commit all transactions.

2. Wait until SHOW INNODB STATUS shows that there are no active transactions in the database, and the main thread status of InnoDB is Waiting for server activity. Then you can make a copy of the .ibd file.

Another method for making a clean copy of an .ibd file is to use the commercial InnoDB Hot Backup tool:

1. Use InnoDB Hot Backup to back up the InnoDB installation.

2. Start a second **mysqld** server on the backup and let it clean up the .ibd files in the backup.

It is in the TODO to also allow moving clean .ibd files to another MySQL installation. This requires resetting of transaction IDs and log sequence numbers in the .ibd file.

15.8. Adding and Removing InnoDB Data and Log Files

This section describes what you can do when your InnoDB tablespace runs out of room or when you want to change the size of the log files.

From MySQL 3.23.50 and 4.0.2, the easiest way to increase the size of the InnoDB tablespace is to configure it from the beginning to be auto-extending. Specify the autoextend attribute for the last data file in the tablespace definition. Then InnoDB increases the size of that file automatically in 8MB increments when it runs out of space. Starting with MySQL 4.0.24 and 4.1.5, the increment size can be configured with the option innodb_autoextend_increment, in megabytes. The default value is 8.

Alternatively, you can increase the size of your tablespace by adding another data file. To do this, you have to shut down the MySQL server, edit the my.cnf file to add a new data file to the end of innodb_data_file_path, and start the server again.

If your last data file was defined with the keyword autoextend, the procedure to edit my.cnf must take into account the size to which the last data file has grown. You have to look at the size of the data file, round the size downward to the closest multiple of 1024 * 1024 bytes (= 1MB), and specify the rounded size explicitly in innodb_data_file_path. Then you can add another data file. Remember that only the last data file in the innodb_data_file_path can be specified as auto-extending.

As an example, assume that the tablespace has just one auto-extending data file ibdata1:

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:10M:autoextend
```

Suppose that this data file, over time, has grown to 988MB. Below is the configuration line after adding another auto-extending data file.

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:988M;/disk2/ibdata2:50M:autoextend
```

When you add a new file to the tablespace, make sure that it does not exist. InnoDB creates and initializes the file when you restart the server.

Currently, you cannot remove a data file from the tablespace. To decrease the size of your tablespace, use this procedure:

1. Use **mysqldump** to dump all your InnoDB tables.
2. Stop the server.
3. Remove all the existing tablespace files.
4. Configure a new tablespace.
5. Restart the server.
6. Import the dump files.

If you want to change the number or the size of your InnoDB log files, you have to stop the MySQL server and make sure that it shuts down without errors. Then copy the old log files into a safe place just

in case something went wrong in the shutdown and you need them to recover the tablespace. Delete the old log files from the log file directory, edit my.cnf to change the log file configuration, and start the MySQL server again. **mysqld** sees that no log files exist at startup and tells you that it is creating new ones.

15.9. Backing Up and Recovering an InnoDB Database

The key to safe database management is taking regular backups.

InnoDB Hot Backup is an online backup tool you can use to backup your InnoDB database while it is running. InnoDB Hot Backup does not require you to shut down your database and it does not set any locks or disturb your normal database processing. InnoDB Hot Backup is a non-free (commercial) additional tool whose annual license fee is 390 euros per computer where the MySQL server is run. See the InnoDB Hot Backup home page [http://www.innodb.com/order.html] for detailed information and screenshots.

If you are able to shut down your MySQL server, you can make a ``binary'' backup that consists of all files used by InnoDB to manage its tables. Use the following procedure:

1. Shut down your MySQL server and make sure that it shuts down without errors.
2. Copy all your data files (ibdata files, .ibd files) into a safe place.
3. Copy all your ib_logfiles to a safe place.
4. Copy your my.cnf configuration file or files to a safe place.
5. Copy all the .frm files for your InnoDB tables to a safe place.

Replication works with InnoDB type tables, so you can use MySQL replication capabilities to keep a copy of your database at database sites requiring high availability.

In addition to taking binary backups as just described, you should also regularly take dumps of your tables with **mysqldump**. The reason for this is that a binary file might be corrupted without you noticing it. Dumped tables are stored into text files that are human-readable, so spotting table corruption becomes easier. Also, since the format is simpler, the chance for serious data corruption is smaller. **mysqldump** also has a --single-transaction option that you can use to take a consistent snapshot without locking out other clients.

To be able to recover your InnoDB database to the present from the binary backup described above, you have to run your MySQL server with binary logging turned on. Then you can apply the binary log to the backup database to achieve point-in-time recovery:

```
mysqlbinlog yourhostname-bin.123 | mysql
```

To recover from a crash of your MySQL server process, the only thing you have to do is to restart it. InnoDB automatically checks the logs and performs a roll-forward of the database to the present. InnoDB automatically rolls back uncommitted transactions that were present at the time of the crash. During recovery, **mysqld** displays output something like this:

```
InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 13674004
```

```
InnoDB: Doing recovery: scanned up to log sequence number 0 13739520
InnoDB: Doing recovery: scanned up to log sequence number 0 13805056
InnoDB: Doing recovery: scanned up to log sequence number 0 13870592
InnoDB: Doing recovery: scanned up to log sequence number 0 13936128
...
InnoDB: Doing recovery: scanned up to log sequence number 0 20555264
InnoDB: Doing recovery: scanned up to log sequence number 0 20620800
InnoDB: Doing recovery: scanned up to log sequence number 0 20664692
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 16745
InnoDB: Rolling back of trx no 16745 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections
```

If your database gets corrupted or your disk fails, you have to do the recovery from a backup. In the case of corruption, you should first find a backup that is not corrupted. After restoring the base backup, do the recovery from the binary log files.

In some cases of database corruption it is enough just to dump, drop, and re-create one or a few corrupt tables. You can use the `CHECK TABLE` SQL statement to check whether a table is corrupt, although `CHECK TABLE` naturally cannot detect every possible kind of corruption. You can use `innodb_tablespace_monitor` to check the integrity of the file space management inside the tablespace files.

In some cases, apparent database page corruption is actually due to the operating system corrupting its own file cache, and the data on disk may be okay. It is best first to try restarting your computer. It may eliminate errors that appeared to be database page corruption.

15.9.1. Forcing Recovery

If there is database page corruption, you may want to dump your tables from the database with `SELECT INTO OUTFILE`, and usually most of the data is intact and correct. But the corruption may cause `SELECT * FROM` *`tbl_name`* or `InnoDB` background operations to crash or assert, or even the `InnoDB` roll-forward recovery to crash. Starting from MySQL 3.23.44, there is an `InnoDB` variable that you can use to force the `InnoDB` storage engine to start up, and you can also prevent background operations from running, so that you are able to dump your tables. For example, you can add the following line to the `[mysqld]` section of your option file before restarting the server:

```
[mysqld]
innodb_force_recovery = 4
```

Before MySQL 4.0, use this syntax instead:

```
[mysqld]
set-variable = innodb_force_recovery=4
```

The allowable non-zero values for `innodb_force_recovery` follow. A larger number includes all precautions of lower numbers. If you are able to dump your tables with an option value of at most 4, then you are relatively safe that only some data on corrupt individual pages is lost. A value of 6 is more dramatic, because database pages are left in an obsolete state, which in turn may introduce more corruption into B-trees and other database structures.

- 1 (SRV_FORCE_IGNORE_CORRUPT)

  Let the server run even if it detects a corrupt page; try to make SELECT * FROM *tbl_name* jump over corrupt index records and pages, which helps in dumping tables.

- 2 (SRV_FORCE_NO_BACKGROUND)

  Prevent the main thread from running. If a crash would occur in the purge operation, this prevents it.

- 3 (SRV_FORCE_NO_TRX_UNDO)

  Do not run transaction rollbacks after recovery.

- 4 (SRV_FORCE_NO_IBUF_MERGE)

  Prevent also insert buffer merge operations. If they would cause a crash, better not do them; do not calculate table statistics.

- 5 (SRV_FORCE_NO_UNDO_LOG_SCAN)

  Do not look at undo logs when starting the database: InnoDB treats even incomplete transactions as committed.

- 6 (SRV_FORCE_NO_LOG_REDO)

  Do not do the log roll-forward in connection with recovery.

The database must not otherwise be used with any of these options enabled! As a safety measure, InnoDB prevents users from doing INSERT, UPDATE, or DELETE when innodb_force_recovery is set to a value greater than 0.

Starting from MySQL 3.23.53 and 4.0.4, you are allowed to DROP or CREATE a table even if forced recovery is used. If you know that a certain table is causing a crash in rollback, you can drop it. You can use this also to stop a runaway rollback caused by a failing mass import or ALTER TABLE. You can kill the **mysqld** process and set innodb_force_recovery to 3 to bring your database up without the rollback. Then DROP the table that is causing the runaway rollback.

15.9.2. Checkpoints

InnoDB implements a checkpoint mechanism called a ``fuzzy checkpoint." InnoDB flushes modified database pages from the buffer pool in small batches. There is no need to flush the buffer pool in one single batch, which would in practice stop processing of user SQL statements for a while.

In crash recovery, InnoDB looks for a checkpoint label written to the log files. It knows that all modifications to the database before the label are present in the disk image of the database. Then InnoDB scans the log files forward from the place of the checkpoint, applying the logged modifications to the database.

InnoDB writes to the log files in a circular fashion. All committed modifications that make the database pages in the buffer pool different from the images on disk must be available in the log files in case InnoDB has to do a recovery. This means that when InnoDB starts to reuse a log file in the circular fashion, it has to make sure that the database page images on disk contain the modifications logged in the log file InnoDB is going to reuse. In other words, InnoDB has to make a checkpoint and often this involves flushing of modified database pages to disk.

The preceding description explains why making your log files very big may save disk I/O in checkpointing. It can make sense to set the total size of the log files as big as the buffer pool or even bigger. The

drawback of big log files is that crash recovery can take longer because there is more logged information to apply to the database.

15.10. Moving an InnoDB Database to Another Machine

On Windows, InnoDB internally always stores database and table names in lowercase. To move databases in a binary format from Unix to Windows or from Windows to Unix, you should have all table and database names in lowercase. A convenient way to accomplish this on Unix is to add the following line to the [mysqld] section of your my.cnf before you start creating your databases and tables:

```
[mysqld]
set-variable = lower_case_table_names=1
```

On Windows, lower_case_table_names is set to 1 by default.

Like MyISAM data files, InnoDB data and log files are binary-compatible on all platforms if the floating-point number format on the machines is the same. You can move an InnoDB database simply by copying all the relevant files, which were listed in Section 15.9, “Backing Up and Recovering an InnoDB Database”. If the floating-point formats on the machines are different but you have not used FLOAT or DOUBLE data types in your tables, then the procedure is the same: Just copy the relevant files. If the formats are different and your tables contain floating-point data, you have to use **mysqldump** to dump your tables on one machine and then import the dump files on the other machine.

A performance tip is to switch off autocommit mode when you import data into your database, assuming that your tablespace has enough space for the big rollback segment the the big import transactions generate. Do the commit only after importing a whole table or a segment of a table.

15.11. InnoDB Transaction Model and Locking

In the InnoDB transaction model, the goal has been to combine the best properties of a multi-versioning database with traditional two-phase locking. InnoDB does locking on the row level and runs queries as non-locking consistent reads by default, in the style of Oracle. The lock table in InnoDB is stored so space-efficiently that lock escalation is not needed: Typically several users are allowed to lock every row in the database, or any random subset of the rows, without InnoDB running out of memory.

15.11.1. InnoDB and AUTOCOMMIT

In InnoDB, all user activity occurs inside a transaction. If the autocommit mode is enabled, each SQL statement forms a single transaction on its own. MySQL always starts a new connection with autocommit enabled.

If the autocommit mode is switched off with SET AUTOCOMMIT = 0, then we can consider that a user always has a transaction open. A SQL COMMIT or ROLLBACK statement ends the current transaction and a new one starts. Both statements release all InnoDB locks that were set during the current transaction. A COMMIT means that the changes made in the current transaction are made permanent and become visible to other users. A ROLLBACK statement, on the other hand, cancels all modifications made by the current transaction.

If the connection has autocommit enabled, the user can still perform a multiple-statement transaction by starting it with an explicit START TRANSACTION or BEGIN statement and ending it with COMMIT or ROLLBACK.

15.11.2. InnoDB and TRANSACTION ISOLATION LEVEL

In terms of the SQL:1992 transaction isolation levels, the InnoDB default is REPEATABLE READ. Starting from MySQL 4.0.5, InnoDB offers all four different transaction isolation levels described by the SQL standard. You can set the default isolation level for all connections by using the --transaction-isolation option on the command line or in option files. For example, you can set the option in the [mysqld] section of my.cnf like this:

```
[mysqld]
transaction-isolation = {READ-UNCOMMITTED | READ-COMMITTED
                         | REPEATABLE-READ | SERIALIZABLE}
```

A user can change the isolation level of a single session or all new incoming connections with the SET TRANSACTION statement. Its syntax is as follows:

```
SET [SESSION | GLOBAL] TRANSACTION ISOLATION LEVEL
                       {READ UNCOMMITTED | READ COMMITTED
                        | REPEATABLE READ | SERIALIZABLE}
```

Note that there are hyphens in the level names for the --transaction-isolation option, but not for the SET TRANSACTION statement.

The default behavior is to set the isolation level for the next (not started) transaction. If you use the GLOBAL keyword, the statement sets the default transaction level globally for all new connections created from that point on (but not existing connections). You need the SUPER privilege to do this. Using the SESSION keyword sets the default transaction level for all future transactions performed on the current connection.

Any client is free to change the session isolation level (even in the middle of a transaction), or the isolation level for the next transaction.

Before MySQL 3.23.50, SET TRANSACTION had no effect on InnoDB tables. Before 4.0.5, only REPEATABLE READ and SERIALIZABLE were available.

You can query the global and session transaction isolation levels with these statements:

```
SELECT @@global.tx_isolation;
SELECT @@tx_isolation;
```

In row-level locking, InnoDB uses so-called ``next-key locking.'' That means that besides index records, InnoDB can also lock the ``gap'' before an index record to block insertions by other users immediately before the index record. A next-key lock refers to a lock that locks an index record and the gap before it. A gap lock refers to a lock that only locks a gap before some index record.

A detailed description of each isolation level in InnoDB:

- READ UNCOMMITTED

  SELECT statements are performed in a non-locking fashion, but a possible earlier version of a record might be used. Thus, using this isolation level, such reads are not ``consistent.'' This is also called ``dirty read.'' Other than that, this isolation level works like READ COMMITTED.

- READ COMMITTED

  A somewhat Oracle-like isolation level. All SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE statements lock only the index records, not the gaps before them, and

thus allow free inserting of new records next to locked records. UPDATE and DELETE statements that use a unique index with a unique search condition lock only the index record found, not the gap before it. In range-type UPDATE and DELETE statements, InnoDB must set next-key or gap locks and block insertions by other users to the gaps covered by the range. This is necessary because ``phantom rows'' must be blocked for MySQL replication and recovery to work.

Consistent reads behave as in Oracle: Each consistent read, even within the same transaction, sets and reads its own fresh snapshot. See Section 15.11.3, “Consistent Non-Locking Read”.

- REPEATABLE READ

  This is the default isolation level of InnoDB. SELECT ... FOR UPDATE, SELECT ... LOCK IN SHARE MODE, UPDATE, and DELETE statements that use a unique index with a unique search condition lock only the index record found, not the gap before it. With other search conditions, these operations employ next-key locking, locking the index range scanned with next-key or gap locks, and block new insertions by other users.

  In consistent reads, there is an important difference from the previous isolation level: In this level, all consistent reads within the same transaction read the same snapshot established by the first read. This convention means that if you issue several plain SELECT statements within the same transaction, these SELECT statements are consistent also with respect to each other. See Section 15.11.3, “Consistent Non-Locking Read”.

- SERIALIZABLE

  This level is like REPEATABLE READ, but all plain SELECT statements are implicitly converted to SELECT ... LOCK IN SHARE MODE.

15.11.3. Consistent Non-Locking Read

A consistent read means that InnoDB uses its multi-versioning to present to a query a snapshot of the database at a point in time. The query see the changes made by exactly those transactions that committed before that point of time, and no changes made by later or uncommitted transactions. The exception to this rule is that the query sees the changes made by the transaction itself that issues the query.

If you are running with the default REPEATABLE READ isolation level, then all consistent reads within the same transaction read the snapshot established by the first such read in that transaction. You can get a fresher snapshot for your queries by committing the current transaction and after that issuing new queries.

Consistent read is the default mode in which InnoDB processes SELECT statements in READ COMMITTED and REPEATABLE READ isolation levels. A consistent read does not set any locks on the tables it accesses, and therefore other users are free to modify those tables at the same time a consistent read is being performed on the table.

15.11.4. Locking Reads SELECT ... FOR UPDATE and SELECT ... LOCK IN SHARE MODE

In some circumstances, a consistent read is not convenient. For example, you might want to add a new row into your table child, and make sure that the child has a parent in table parent. The following example shows how to implement referential integrity in your application code.

Suppose that you use a consistent read to read the table parent and indeed see the parent of the child in the table. Can you safely add the child row to table child? No, because it may happen that meanwhile some other user deletes the parent row from the table parent, without you being aware of it.

The solution is to perform the SELECT in a locking mode using LOCK IN SHARE MODE:

```
SELECT * FROM parent WHERE NAME = 'Jones' LOCK IN SHARE MODE;
```

Performing a read in share mode means that we read the latest available data, and set a shared mode lock on the rows we read. A shared mode lock prevents others from updating or deleting the row we have read. Also, if the latest data belongs to a yet uncommitted transaction of another client connection, we wait until that transaction commits. After we see that the preceding query returns the parent 'Jones', we can safely add the child record to the child table and commit our transaction.

Let us look at another example: We have an integer counter field in a table child_codes that we use to assign a unique identifier to each child added to table child. Obviously, using a consistent read or a shared mode read to read the present value of the counter is not a good idea, since two users of the database may then see the same value for the counter, and a duplicate-key error occurs if two users attempt to add children with the same identifier to the table.

Here, LOCK IN SHARE MODE is not a good solution because if two users read the counter at the same time, at least one of them ends up in deadlock when attempting to update the counter.

In this case, there are two good ways to implement the reading and incrementing of the counter: (1) update the counter first by incrementing it by 1 and only after that read it, or (2) read the counter first with a lock mode FOR UPDATE, and increment after that. The latter approach can be implemented as follows:

```
SELECT counter_field FROM child_codes FOR UPDATE;
UPDATE child_codes SET counter_field = counter_field + 1;
```

A SELECT ... FOR UPDATE reads the latest available data, setting exclusive locks on each row it reads. Thus it sets the same locks a searched SQL UPDATE would set on the rows.

Please note that the above is merely an example of how SELECT ... FOR UPDATE works. In MySQL, the specific task of generating a unique identifier actually can be accomplished using only a single access to the table:

```
UPDATE child_codes SET counter_field = LAST_INSERT_ID(counter_field + 1);
SELECT LAST_INSERT_ID();
```

The SELECT statement merely retrieves the identifier information (specific to the current connection). It does not access any table.

15.11.5. Next-Key Locking: Avoiding the Phantom Problem

In row-level locking, InnoDB uses an algorithm called ``next-key locking." InnoDB does the row-level locking in such a way that when it searches or scans an index of a table, it sets shared or exclusive locks on the index records it encounters. Thus the row-level locks are actually index record locks.

The locks InnoDB sets on index records also affect the ``gap" before that index record. If a user has a shared or exclusive lock on record R in an index, another user cannot insert a new index record immediately before R in the index order. This locking of gaps is done to prevent the so-called ``phantom problem." Suppose that you want to read and lock all children from the child table with an identifier value larger than 100, with the intent of updating some column in the selected rows later:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

Suppose that there is an index on the `id` column. The query scans that index starting from the first record where `id` is bigger than 100. If the locks set on the index records would not lock out inserts made in the gaps, a new row might meanwhile be inserted to the table. If you execute the same `SELECT` within the same transaction, you would see a new row in the result set returned by the query. This is contrary the isolation principle of transactions: A transaction should be able to run so that the data it has read does not change during the transaction. If we regard a set of rows as a data item, the new ``phantom'' child would violate this isolation principle.

When `InnoDB` scans an index, it can also lock the gap after the last record in the index. Just that happens in the previous example: The locks set by `InnoDB` prevent any insert to the table where `id` would be bigger than 100.

You can use next-key locking to implement a uniqueness check in your application: If you read your data in share mode and do not see a duplicate for a row you are going to insert, then you can safely insert your row and know that the next-key lock set on the successor of your row during the read prevents anyone meanwhile inserting a duplicate for your row. Thus the next-key locking allows you to ``lock'' the non-existence of something in your table.

15.11.6. An Example of How the Consistent Read Works in InnoDB

Suppose that you are running in the default `REPEATABLE READ` isolation level. When you issue a consistent read, that is, an ordinary `SELECT` statement, `InnoDB` gives your transaction a timepoint according to which your query sees the database. If another transaction deletes a row and commits after your timepoint was assigned, you do not see the row as having been deleted. Inserts and updates are treated similarly.

You can advance your timepoint by committing your transaction and then doing another `SELECT`.

This is called ``multi-versioned concurrency control.''

```
                User A                  User B

            SET AUTOCOMMIT=0;       SET AUTOCOMMIT=0;
time
|           SELECT * FROM t;
|           empty set
|                                    INSERT INTO t VALUES (1, 2);
|
v           SELECT * FROM t;
            empty set
                                    COMMIT;

            SELECT * FROM t;
            empty set

            COMMIT;

            SELECT * FROM t;
            ---------------------
            |    1    |    2    |
            ---------------------
            1 row in set
```

In this example, user A sees the row inserted by B only when B has committed the insert and A has committed as well, so that the timepoint is advanced past the commit of B.

If you want to see the ``freshest'' state of the database, you should use either the `READ COMMITTED`

isolation level or a locking read:

```
SELECT * FROM t LOCK IN SHARE MODE;
```

15.11.7. Locks Set by Different SQL Statements in InnoDB

A locking read, an UPDATE, or a DELETE generally set record locks on every index record that is scanned in the processing of the SQL query. It does not matter if there are WHERE conditions in the query that would exclude the row from the result set of the query. InnoDB does not remember the exact WHERE condition, but only knows which index ranges were scanned. The record locks are normally next-key locks that also block inserts to the ``gap'' immediately before the record.

If the locks to be set are exclusive, then InnoDB always retrieves also the clustered index record and sets a lock on it.

If you do not have indexes suitable for your query and MySQL has to scan the whole table to process the query, every row of the table becomes locked, which in turn blocks all inserts by other users to the table. It is important to create good indexes so that your queries do not unnecessarily need to scan many rows.

- SELECT ... FROM is a consistent read, reading a snapshot of the database and setting no locks unless the transaction isolation level is set to SERIALIZABLE. For SERIALIZABLE level, this sets shared next-key locks on the index records it encounters.
- SELECT ... FROM ... LOCK IN SHARE MODE sets shared next-key locks on all index records the read encounters.
- SELECT ... FROM ... FOR UPDATE sets exclusive next-key locks on all index records the read encounters.
- INSERT INTO ... VALUES (...) sets an exclusive lock on the inserted row. Note that this lock is not a next-key lock and does not prevent other users from inserting to the gap before the inserted row. If a duplicate-key error occurs, a shared lock on the duplicate index record is set.
- While initializing a previously specified AUTO_INCREMENT column on a table, InnoDB sets an exclusive lock on the end of the index associated with the AUTO_INCREMENT column. In accessing the auto-increment counter, InnoDB uses a specific table lock mode AUTO-INC where the lock lasts only to the end of the current SQL statement, instead of to the end of the whole transaction. See Section 15.11.1, “InnoDB and AUTOCOMMIT”.

  Before MySQL 3.23.50, SHOW TABLE STATUS applied to a table with an AUTO_INCREMENT column sets an exclusive row-level lock to the high end of the AUTO_INCREMENT index. This means also that SHOW TABLE STATUS could cause a deadlock of transactions, something that may surprise users. Starting from MySQL 3.23.50, InnoDB fetches the value of a previously initialized AUTO_INCREMENT column without setting any locks.
- INSERT INTO T SELECT ... FROM S WHERE ... sets an exclusive (non-next-key) lock on each row inserted into T. It does the search on S as a consistent read, but sets shared next-key locks on S if MySQL binary logging is turned on. InnoDB has to set locks in the latter case: In roll-forward recovery from a backup, every SQL statement has to be executed in exactly the same way it was done originally.
- CREATE TABLE ... SELECT ... performs the SELECT as a consistent read or with shared locks, as in the previous item.
- REPLACE is done like an insert if there is no collision on a unique key. Otherwise, an exclusive

next-key lock is placed on the row that has to be updated.

- UPDATE ... WHERE ... sets an exclusive next-key lock on every record the search encounters.
- DELETE FROM ... WHERE ... sets an exclusive next-key lock on every record the search encounters.
- If a FOREIGN KEY constraint is defined on a table, any insert, update, or delete that requires checking of the constraint condition sets shared record-level locks on the records it looks at to check the constraint. InnoDB also sets these locks in the case where the constraint fails.
- LOCK TABLES sets table locks, but it is the higher MySQL layer above the InnoDB layer that sets these locks. Beginning with MySQL 4.0.20 and 4.1.2, InnoDB is aware of table locks if innodb_table_locks=1 and AUTOCOMMIT=0, and the MySQL layer above InnoDB knows about row-level locks. Before that, the automatic deadlock detection of InnoDB cannot detect deadlocks where such table locks are involved. Also, since the higher MySQL layer does not know about row-level locks, it is possible to get a table lock on a table where another user currently has row-level locks. But that does not put transaction integrity in danger. See Section 15.11.9, “Deadlock Detection and Rollback”. See Section 15.17, “Restrictions on InnoDB Tables”.

15.11.8. When Does MySQL Implicitly Commit or Roll Back a Transaction?

MySQL begins each client connection with autocommit mode enabled by default. When autocommit is enabled, MySQL does a commit after each SQL statement if that statement did not return an error.

If you have the autocommit mode off and close a connection without calling an explicit commit of your transaction, then MySQL rolls back your transaction.

If an SQL statement returns an error, the commit/rollback behavior depends on the error. See Section 15.16, “Error Handling”.

The following SQL statements (and any synonyms for them) cause an implicit commit of the current transaction in MySQL:

- ALTER TABLE, BEGIN, CREATE INDEX, DROP DATABASE, DROP INDEX, DROP TABLE, LOAD MASTER DATA, LOCK TABLES, RENAME TABLE, SET AUTOCOMMIT=1, START TRANSACTION, TRUNCATE, UNLOCK TABLES.
- CREATE TABLE (this commits only if before MySQL 4.0.13 and MySQL binary logging is used).
- The CREATE TABLE statement in InnoDB is processed as a single transaction. This means that a ROLLBACK from the user does not undo CREATE TABLE statements the user made during that transaction.

15.11.9. Deadlock Detection and Rollback

InnoDB automatically detects a deadlock of transactions and rolls back a transaction or transactions to prevent the deadlock. Starting from MySQL 4.0.5, InnoDB tries to pick small transactions to roll back. The size of a transaction is determined by the number of rows it has inserted, updated, or deleted. Prior to 4.0.5, InnoDB always rolled back the transaction whose lock request was the last one to build a deadlock, that is, a cycle in the ``waits-for'' graph of transactions.

Beginning with MySQL 4.0.20 and 4.1.2, InnoDB is aware of table locks if in-

nodb_table_locks=1 (1 is the default), and the MySQL layer above InnoDB knows about row-level locks. Before that, InnoDB cannot detect deadlocks where a table lock set by a MySQL LOCK TABLES statement is involved, or if a lock set by another storage engine than InnoDB is involved. You have to resolve these situations by setting the value of the innodb_lock_wait_timeout system variable.

When InnoDB performs a complete rollback of a transaction, all the locks of the transaction are released. However, if just a single SQL statement is rolled back as a result of an error, some of the locks set by the SQL statement may be preserved. This is because InnoDB stores row locks in a format such it cannot know afterward which lock was set by which SQL statement.

15.11.10. How to Cope with Deadlocks

Deadlocks are a classic problem in transactional databases, but they are not dangerous unless they are so frequent that you cannot run certain transactions at all. Normally, you must write your applications so that they are always prepared to re-issue a transaction if it gets rolled back because of a deadlock.

InnoDB uses automatic row-level locking. You can get deadlocks even in the case of transactions that just insert or delete a single row. That is because these operations are not really ``atomic''; they automatically set locks on the (possibly several) index records of the row inserted or deleted.

You can cope with deadlocks and reduce the likelihood of their occurrence with the following techniques:

- Use SHOW INNODB STATUS to determine the cause of the latest deadlock. That can help you to tune your application to avoid deadlocks. This strategy can be used as of MySQL 3.23.52 and 4.0.3, depending on your MySQL series.
- Always be prepared to re-issue a transaction if it fails due to deadlock. Deadlocks are not dangerous. Just try again.
- Commit your transactions often. Small transactions are less prone to collide.
- If you are using locking reads (SELECT ... FOR UPDATE or ... LOCK IN SHARE MODE), try using a lower isolation level such as READ COMMITTED.
- Access your tables and rows in a fixed order. Then transactions form nice queues and do not deadlock.
- Add well-chosen indexes to your tables. Then your queries need to scan fewer index records and consequently set fewer locks. Use EXPLAIN SELECT to determine which indexes the MySQL server regards as the most appropriate for your queries.
- Use less locking. If you can afford to allow a SELECT to return data from an old snapshot, do not add the clause FOR UPDATE or LOCK IN SHARE MODE to it. Using READ COMMITTED isolation level is good here, because each consistent read within the same transaction reads from its own fresh snapshot.
- If nothing helps, serialize your transactions with table-level locks. The correct way to use LOCK TABLES with transactional tables, like InnoDB, is to set AUTOCOMMIT = 0 and not to call UNLOCK TABLES until you commit the transaction explicitly. For example, if you need to write table t1 and read table t2, you can do this:

  ```
  SET AUTOCOMMIT=0;
  LOCK TABLES t1 WRITE, t2 READ, ...;
  [do something with tables t1 and t2 here];
  COMMIT;
  UNLOCK TABLES;
  ```

Table-level locks make your transactions queue nicely, and deadlocks are avoided.

- Another way to serialize transactions is to create an auxiliary ``semaphore'' table that contains just a single row. Have each transaction update that row before accessing other tables. In that way, all transactions happen in a serial fashion. Note that the InnoDB instant deadlock detection algorithm also works in this case, because the serializing lock is a row-level lock. With MySQL table-level locks, the timeout method must be used to resolve deadlocks.

- In applications using AUTOCOMMIT=1 and MySQL's LOCK TABLES command, InnoDB's internal table locks that were present from 4.0.20 to 4.0.23 can cause deadlocks. Starting from 4.0.22, you can set innodb_table_locks=0 in my.cnf to fall back to the old behavior and remove the problem. 4.0.24 does not set InnoDB table locks if AUTOCOMMIT=1.

15.12. InnoDB Performance Tuning Tips

- If the Unix top tool or the Windows Task Manager shows that the CPU usage percentage with your workload is less than 70%, your workload is probably disk-bound. Maybe you are making too many transaction commits, or the buffer pool is too small. Making the buffer pool bigger can help, but do not set it bigger than 80% of physical memory.

- Wrap several modifications into one transaction. InnoDB must flush the log to disk at each transaction commit if that transaction made modifications to the database. Since the rotation speed of a disk is typically at most 167 revolutions/second, that constrains the number of commits to the same 167th/second if the disk does not fool the operating system.

- If you can afford the loss of some of the latest committed transactions, you can set the my.cnf parameter innodb_flush_log_at_trx_commit to 0. InnoDB tries to flush the log once per second anyway, although the flush is not guaranteed.

- Make your log files big, even as big as the buffer pool. When InnoDB has written the log files full, it has to write the modified contents of the buffer pool to disk in a checkpoint. Small log files cause many unnecessary disk writes. The drawback of big log files is that the recovery time is longer.

- Make the log buffer quite big as well (say, 8MB).

- Use the VARCHAR column type instead of CHAR if you are storing variable-length strings or if the column may contain many NULL values. A CHAR(*N*) column always takes *N* bytes to store data, even if the string is shorter or its value is NULL. Smaller tables fit better in the buffer pool and reduce disk I/O.

- (Relevant from 3.23.39 up.) In some versions of GNU/Linux and Unix, flushing files to disk with the Unix fsync() and other similar methods is surprisingly slow. The default method InnoDB uses is the fsync() function. If you are not satisfied with the database write performance, you might try setting innodb_flush_method in my.cnf to O_DSYNC, although O_DSYNC seems to be slower on most systems.

- When importing data into InnoDB, make sure that MySQL does not have autocommit mode enabled because that would require a log flush to disk for every insert. To disable autocommit during your import operation, surround it with SET AUTOCOMMIT and COMMIT statements:

  ```
  SET AUTOCOMMIT=0;
  /* SQL import statements ... */
  COMMIT;
  ```

If you use the **mysqldump** option `--opt`, you get dump files that are fast to import into an InnoDB table, even without wrapping them with the `SET AUTOCOMMIT` and `COMMIT` statements.

- Beware of big rollbacks of mass inserts: InnoDB uses the insert buffer to save disk I/O in inserts, but no such mechanism is used in a corresponding rollback. A disk-bound rollback can take 30 times the time of the corresponding insert. Killing the database process does not help because the rollback starts again at the server startup. The only way to get rid of a runaway rollback is to increase the buffer pool so that the rollback becomes CPU-bound and runs fast, or to use a special procedure. See Section 15.9.1, “Forcing Recovery”.

- Beware also of other big disk-bound operations. Use `DROP TABLE + CREATE TABLE` to empty a table, not `DELETE FROM tbl_name`.

- Use the multiple-row `INSERT` syntax to reduce communication overhead between the client and the server if you need to insert many rows:

  ```
  INSERT INTO yourtable VALUES (1,2), (5,5), ...;
  ```

  This tip is valid for inserts into any table type, not just InnoDB.

- If you have `UNIQUE` constraints on secondary keys, starting from MySQL 3.23.52 and 4.0.3, you can speed up table imports by temporarily turning off the uniqueness checks during the import session:

  ```
  SET UNIQUE_CHECKS=0;
  ```

  For big tables, this saves a lot of disk I/O because InnoDB can use its insert buffer to write secondary index records in a batch.

- If you have `FOREIGN KEY` constraints in your tables, starting from MySQL 3.23.52 and 4.0.3, you can speed up table imports by turning the foreign key checks off for a while in the import session:

  ```
  SET FOREIGN_KEY_CHECKS=0;
  ```

  For big tables, this can save a lot of disk I/O.

- If you often have recurring queries to tables that are not updated frequently, use the query cache available as of MySQL 4.0:

  ```
  [mysqld]
  query_cache_type = ON
  query_cache_size = 10M
  ```

  In MySQL 4.0, the query cache works only with autocommit enabled. This restriction is removed in MySQL 4.1.1 and up.

15.12.1. SHOW INNODB STATUS and the InnoDB Monitors

Starting from MySQL 3.23.42, InnoDB includes InnoDB Monitors that print information about the InnoDB internal state. Starting from MySQL 3.23.52 and 4.0.3, you can use the SQL statement `SHOW INNODB STATUS` to fetch the output of the standard InnoDB Monitor to your SQL client. The in-

formation is useful in performance tuning. If you are using the **mysql** interactive SQL client, the output is more readable if you replace the usual semicolon statement terminator by \G:

```
mysql> SHOW INNODB STATUS\G
```

Another way to use InnoDB Monitors is to let them continuously write data to the standard output of the server **mysqld**. In this case, no output is sent to clients. When switched on, InnoDB Monitors print data about every 15 seconds. Server output usually is directed to the .err log in the MySQL data directory. This data is useful in performance tuning. On Windows, you must start the server from a command prompt in a console window with the --console option if you want to direct the output to the window rather than to the error log.

Monitor output includes information of the following types:

- Table and record locks held by each active transaction
- Lock waits of a transactions
- Semaphore waits of threads
- Pending file I/O requests
- Buffer pool statistics
- Purge and insert buffer merge activity of the main InnoDB thread

To cause the standard InnoDB Monitor to write to the standard output of **mysqld**, use the following SQL statement:

```
CREATE TABLE innodb_monitor(a INT) TYPE=InnoDB;
```

The monitor can be stopped by issuing the following statement:

```
DROP TABLE innodb_monitor;
```

The CREATE TABLE syntax is just a way to pass a command to the InnoDB engine through the MySQL SQL parser: The only things that matter are the table name innodb_monitor and that it be an InnoDB table. The structure of the table is not relevant at all for the InnoDB Monitor. If you shut down the server when the monitor is running, and you want to start the monitor again, you have to drop the table before you can issue a new CREATE TABLE statement to start the monitor. This syntax may change in a future release.

In a similar way, you can start innodb_lock_monitor, which is otherwise the same as innodb_monitor but also prints a lot of lock information. A separate innodb_tablespace_monitor prints a list of created file segments existing in the tablespace and also validates the tablespace allocation data structures. Starting from 3.23.44, there is innodb_table_monitor with which you can print the contents of the InnoDB internal data dictionary.

A sample of InnoDB Monitor output:

```
mysql> SHOW INNODB STATUS\G
*************************** 1. row ***************************
Status:
=====================================
030709 13:00:59 INNODB MONITOR OUTPUT
```

```
=====================================
Per second averages calculated from the last 18 seconds
----------
SEMAPHORES
----------
OS WAIT ARRAY INFO: reservation count 413452, signal count 378357
--Thread 32782 has waited at btr0sea.c line 1477 for 0.00 seconds the semaphore:
X-lock on RW-latch at 41a28668 created in file btr0sea.c line 135
a writer (thread id 32782) has reserved it in mode wait exclusive
number of readers 1, waiters flag 1
Last time read locked in file btr0sea.c line 731
Last time write locked in file btr0sea.c line 1347
Mutex spin waits 0, rounds 0, OS waits 0
RW-shared spins 108462, OS waits 37964; RW-excl spins 681824, OS waits 375485
------------------------
LATEST FOREIGN KEY ERROR
------------------------
030709 13:00:59 Transaction:
TRANSACTION 0 290328284, ACTIVE 0 sec, process no 3195, OS thread id 34831 inser
ting
15 lock struct(s), heap size 2496, undo log entries 9
MySQL thread id 25, query id 4668733 localhost heikki update
insert into ibtest11a (D, B, C) values (5, 'khDk' ,'khDk')
Foreign key constraint fails for table test/ibtest11a:
,
  CONSTRAINT `0_219242` FOREIGN KEY (`A`, `D`) REFERENCES `ibtest11b` (`A`, `D`)
 ON DELETE CASCADE ON UPDATE CASCADE
Trying to add in child table, in index PRIMARY tuple:
 0: len 4; hex 80000101; asc ....;; 1: len 4; hex 80000005; asc ....;; 2: len 4;
 hex 6b68446b; asc khDk;; 3: len 6; hex 0000114e0edc; asc ...N..;; 4: len 7; hex
 00000000c3e0a7; asc .......;; 5: len 4; hex 6b68446b; asc khDk;;
But in parent table test/ibtest11b, in index PRIMARY,
the closest match we can find is record:
RECORD: info bits 0 0: len 4; hex 8000015b; asc ...[;; 1: len 4; hex 80000005; a
sc ....;; 2: len 3; hex 6b6864; asc khd;; 3: len 6; hex 0000111ef3eb; asc ......
;; 4: len 7; hex 800001001e0084; asc .......;; 5: len 3; hex 6b6864; asc khd;;
------------------------
LATEST DETECTED DEADLOCK
------------------------
030709 12:59:58
*** (1) TRANSACTION:
TRANSACTION 0 290252780, ACTIVE 1 sec, process no 3185, OS thread id 30733 inser
ting
LOCK WAIT 3 lock struct(s), heap size 320, undo log entries 146
MySQL thread id 21, query id 4553379 localhost heikki update
INSERT INTO alex1 VALUES(86, 86, 794,'aA35818','bb','c79166','d4766t','e187358f'
,'g84586','h794',date_format('2001-04-03 12:54:22','%Y-%m-%d %H:%i'),7
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 index symbole
trx id 0 290252780 lock mode S waiting
Record lock, heap no 324 RECORD: info bits 0 0: len 7; hex 61613335383138; asc a
a35818;; 1:
*** (2) TRANSACTION:
TRANSACTION 0 290251546, ACTIVE 2 sec, process no 3190, OS thread id 32782 inser
ting
130 lock struct(s), heap size 11584, undo log entries 437
MySQL thread id 23, query id 4554396 localhost heikki update
REPLACE INTO alex1 VALUES(NULL, 32, NULL,'aa3572','','c3572','d6012t','', NULL,'
h396', NULL, NULL, 7.31,7.31,7.31,200)
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 index symbole
trx id 0 290251546 lock_mode X locks rec but not gap
Record lock, heap no 324 RECORD: info bits 0 0: len 7; hex 61613335383138; asc a
a35818;; 1:
```

```
*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 index symbole
trx id 0 290251546 lock_mode X locks gap before rec insert intention waiting
Record lock, heap no 82 RECORD: info bits 0 0: len 7; hex 61613335373230; asc aa
35720;; 1:
*** WE ROLL BACK TRANSACTION (1)
------------
TRANSACTIONS
------------
Trx id counter 0 290328385
Purge done for trx's n:o < 0 290315608 undo n:o < 0 17
Total number of lock structs in row lock hash table 70
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0 0, not started, process no 3491, OS thread id 42002
MySQL thread id 32, query id 4668737 localhost heikki
show innodb status
---TRANSACTION 0 290328384, ACTIVE 0 sec, process no 3205, OS thread id 38929 in
serting
1 lock struct(s), heap size 320
MySQL thread id 29, query id 4668736 localhost heikki update
insert into speedc values (1519229,1, 'hgjhjgghggjgjgjgjgjgjggjgjgjgjgjgjgggjgjgjlhh
gghgggggghhjhghgggggghjhghghghghghhhhghghghjhhjghjghjkghjghjghjghjfhjfh
---TRANSACTION 0 290328383, ACTIVE 0 sec, process no 3180, OS thread id 28684 co
mmitting
1 lock struct(s), heap size 320, undo log entries 1
MySQL thread id 19, query id 4668734 localhost heikki update
insert into speedcm values (1603393,1, 'hgjhjgghggjgjgjgjgjgjggjgjgjgjgjgjgggjgjgjlh
hgghgggggghhjhghgggggghjhghghghghghhhhghghghjhhjghjghjkghjghjghjghjfhjf
---TRANSACTION 0 290328327, ACTIVE 0 sec, process no 3200, OS thread id 36880 st
arting index read
LOCK WAIT 2 lock struct(s), heap size 320
MySQL thread id 27, query id 4668644 localhost heikki Searching rows for update
update ibtest11a set B = 'kHdkkkk' where A = 89572
------- TRX HAS BEEN WAITING 0 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 0 page no 65556 n bits 232 table test/ibtest11a index PRIM
ARY trx id 0 290328327 lock_mode X waiting
Record lock, heap no 1 RECORD: info bits 0 0: len 9; hex 73757072656d756d00; asc
 supremum.;;
------------------
---TRANSACTION 0 290328284, ACTIVE 0 sec, process no 3195, OS thread id 34831 ro
llback of SQL statement
ROLLING BACK 14 lock struct(s), heap size 2496, undo log entries 9
MySQL thread id 25, query id 4668733 localhost heikki update
insert into ibtest11a (D, B, C) values (5, 'khDk' ,'khDk')
---TRANSACTION 0 290327208, ACTIVE 1 sec, process no 3190, OS thread id 32782
58 lock struct(s), heap size 5504, undo log entries 159
MySQL thread id 23, query id 4668732 localhost heikki update
REPLACE INTO alex1 VALUES(86, 46, 538,'aa95666','bb','c95666','d9486t','e200498f
','g86814','h538',date_format('2001-04-03 12:54:22','%Y-%m-%d %H:%i'),
---TRANSACTION 0 290323325, ACTIVE 3 sec, process no 3185, OS thread id 30733 in
serting
4 lock struct(s), heap size 1024, undo log entries 165
MySQL thread id 21, query id 4668735 localhost heikki update
INSERT INTO alex1 VALUES(NULL, 49, NULL,'aa42837','','c56319','d1719t','', NULL,
'h321', NULL, NULL, 7.31,7.31,7.31,200)
--------
FILE I/O
--------
I/O thread 0 state: waiting for i/o request (insert buffer thread)
I/O thread 1 state: waiting for i/o request (log thread)
I/O thread 2 state: waiting for i/o request (read thread)
I/O thread 3 state: waiting for i/o request (write thread)
Pending normal aio reads: 0, aio writes: 0,
 ibuf aio reads: 0, log i/o's: 0, sync i/o's: 0
```

```
Pending flushes (fsync) log: 0; buffer pool: 0
151671 OS file reads, 94747 OS file writes, 8750 OS fsyncs
25.44 reads/s, 18494 avg bytes/read, 17.55 writes/s, 2.33 fsyncs/s
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf for space 0: size 1, free list len 19, seg size 21,
85004 inserts, 85004 merged recs, 26669 merges
Hash table size 207619, used cells 14461, node heap has 16 buffer(s)
1877.67 hash searches/s, 5121.10 non-hash searches/s
---
LOG
---
Log sequence number 18 1212842764
Log flushed up to   18 1212665295
Last checkpoint at  18 1135877290
0 pending log writes, 0 pending chkp writes
4341 log i/o's done, 1.22 log i/o's/second
----------------------
BUFFER POOL AND MEMORY
----------------------
Total memory allocated 84966343; in additional pool allocated 1402624
Buffer pool size   3200
Free buffers       110
Database pages     3074
Modified db pages  2674
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages read 171380, created 51968, written 194688
28.72 reads/s, 20.72 creates/s, 47.55 writes/s
Buffer pool hit rate 999 / 1000
--------------
ROW OPERATIONS
--------------
0 queries inside InnoDB, 0 queries in queue
Main thread process no. 3004, id 7176, state: purging
Number of rows inserted 3738558, updated 127415, deleted 33707, read 755779
1586.13 inserts/s, 50.89 updates/s, 28.44 deletes/s, 107.88 reads/s
----------------------------
END OF INNODB MONITOR OUTPUT
============================
1 row in set (0.05 sec)
```

Some notes on the output:

- If the `TRANSACTIONS` section reports lock waits, your application may have lock contention. The output can also help to trace the reasons for transaction deadlocks.
- The `SEMAPHORES` section reports threads waiting for a semaphore and statistics on how many times threads have needed a spin or a wait on a mutex or a rw-lock semaphore. A large number of threads waiting for semaphores may be a result of disk I/O, or contention problems inside `InnoDB`. Contention can be due to heavy parallelism of queries, or problems in operating system thread scheduling. Setting `innodb_thread_concurrency` smaller than the default value of `8` can help in such situations.
- The `BUFFER POOL AND MEMORY` section gives you statistics on pages read and written. You can calculate from these numbers how many data file I/O operations your queries currently are doing.
- The `ROW OPERATIONS` section shows what the main thread is doing.

Beginning with MySQL 4.0.19, InnoDB sends diagnostic output to stderr or files instead of stdout or fixed-size memory buffers, to avoid potential buffer overflow errors. As a side effect, the output of SHOW INNODB STATUS is written to a status file every fifteen seconds. The name of the file is innodb_status.*pid*, where *pid* is the server process ID. This file is created in the MySQL data directory. InnoDB removes the file for a normal shutdown. If abnormal shutdowns have occurred, instances of these status files may be present and must be removed manually. Before removing them, you might want to examine them to see if they contain useful information to the cause of abnormal shutdowns. Beginning with MySQL 4.0.21, the innodb_status.*pid* file is only created if the configuration option innodb_status_file=1 is set.

15.13. Implementation of Multi-Versioning

Because InnoDB is a multi-versioned database, it must keep information about old versions of rows in the tablespace. This information is stored in a data structure called a rollback segment after an analogous data structure in Oracle.

Internally, InnoDB adds two fields to each row stored in the database. A 6-byte field indicates the transaction identifier for the last transaction that inserted or updated the row. Also, a deletion is treated internally as an update where a special bit in the row is set to mark it as deleted. Each row also contains a 7-byte field called the roll pointer. The roll pointer points to an undo log record written to the rollback segment. If the row was updated, the undo log record contains the information necessary to rebuild the content of the row before it was updated.

InnoDB uses the information in the rollback segment to perform the undo operations needed in a transaction rollback. It also uses the information to build earlier versions of a row for a consistent read.

Undo logs in the rollback segment are divided into insert and update undo logs. Insert undo logs are needed only in transaction rollback and can be discarded as soon as the transaction commits. Update undo logs are used also in consistent reads, and they can be discarded only after there is no transaction present for which InnoDB has assigned a snapshot that in a consistent read could need the information in the update undo log to build an earlier version of a database row.

You must remember to commit your transactions regularly, including those transactions that only issue consistent reads. Otherwise, InnoDB cannot discard data from the update undo logs, and the rollback segment may grow too big, filling up your tablespace.

The physical size of an undo log record in the rollback segment is typically smaller than the corresponding inserted or updated row. You can use this information to calculate the space need for your rollback segment.

In the InnoDB multi-versioning scheme, a row is not physically removed from the database immediately when you delete it with an SQL statement. Only when InnoDB can discard the update undo log record written for the deletion can it also physically remove the corresponding row and its index records from the database. This removal operation is called a purge, and it is quite fast, usually taking the same order of time as the SQL statement that did the deletion.

In a scenario where the user inserts and deletes rows in smallish batches at about the same rate in the table, it is possible that the purge thread starts to lag behind, and the table grows bigger and bigger, making everything disk-bound and very slow. Even if the table would carry just 10 MB of useful data, it may grow to occupy 10 GB with all the dead rows. In such a case, it would be good to throttle new row operations, and allocate more resources for the purge thread.

The InnoDB transaction system maintains a list of transactions that have delete-marked index records by UPDATE or DELETE operations. Let the length of this list be *purge_lag*.

Starting with MySQL/InnoDB-4.1.6 and 4.0.22, there is a startup option and settable global variable innodb_max_purge_lag, which is zero by default. When this parameter is non-zero, InnoDB may

delay new row operations. When the *purge_lag* exceeds innodb_max_purge_lag, each INSERT, UPDATE and DELETE operation is delayed by ((*purge_lag*/innodb_max_purge_lag)*10)-5 milliseconds. The delay is computed in the beginning of a purge batch, every ten seconds. The operations are not delayed if purge cannot run because of an old consistent read view that could see the rows to be purged. A typical setting for a problematic workload might be 1 million, assuming that our transactions are small, only 100 bytes in size, and we can allow 100 MB of unpurged rows in our tables.

15.14. Table and Index Structures

MySQL stores its data dictionary information for tables in .frm files in database directories. This is true for all MySQL storage engines. But every InnoDB table also has its own entry in InnoDB internal data dictionaries inside the tablespace. When MySQL drops a table or a database, it has to delete both an .frm file or files, and the corresponding entries inside the InnoDB data dictionary. This is the reason why you cannot move InnoDB tables between databases simply by moving the .frm files. It is also the reason why DROP DATABASE did not work for InnoDB type tables before MySQL 3.23.44.

Every InnoDB table has a special index called the clustered index where the data of the rows is stored. If you define a PRIMARY KEY on your table, the index of the primary key is the clustered index.

If you do not define a PRIMARY KEY for your table, MySQL picks the first UNIQUE index that has only NOT NULL columns as the primary key and InnoDB uses it as the clustered index. If there is no such index in the table, InnoDB internally generates a clustered index where the rows are ordered by the row ID that InnoDB assigns to the rows in such a table. The row ID is a 6-byte field that increases monotonically as new rows are inserted. Thus the rows ordered by the row ID are physically in the insertion order.

Accessing a row through the clustered index is fast because the row data is on the same page where the index search leads. If a table is large, the clustered index architecture often saves a disk I/O when compared to the traditional solution. (In many databases, the data is traditionally stored on a different page from the index record.)

In InnoDB, the records in non-clustered indexes (also called secondary indexes) contain the primary key value for the row. InnoDB uses this primary key value to search for the row from the clustered index. Note that if the primary key is long, the secondary indexes use more space.

InnoDB compares CHAR and VARCHAR strings of different lengths such that the remaining length in the shorter string is treated as if padded with spaces.

15.14.1. Physical Structure of an Index

All indexes in InnoDB are B-trees where the index records are stored in the leaf pages of the tree. The default size of an index page is 16KB. When new records are inserted, InnoDB tries to leave 1/16 of the page free for future insertions and updates of the index records.

If index records are inserted in a sequential order (ascending or descending), the resulting index pages is about 15/16 full. If records are inserted in a random order, the pages are from 1/2 to 15/16 full. If the fillfactor of an index page drops below 1/2, InnoDB tries to contract the index tree to free the page.

15.14.2. Insert Buffering

It is a common situation in a database application that the primary key is a unique identifier and new rows are inserted in the ascending order of the primary key. Thus the insertions to the clustered index do not require random reads from a disk.

On the other hand, secondary indexes are usually non-unique, and insertions into secondary indexes happen in a relatively random order. This would cause a lot of random disk I/O operations without a special

mechanism used in InnoDB.

If an index record should be inserted to a non-unique secondary index, InnoDB checks whether the secondary index page is in the buffer pool. If that is the case, InnoDB does the insertion directly to the index page. If the index page is not found in the buffer pool, InnoDB inserts the record to a special insert buffer structure. The insert buffer is kept so small that it fits entirely in the buffer pool, and insertions can be done very fast.

Periodically, the insert buffer is merged into the secondary index trees in the database. Often it is possible to merge several insertions to the same page of the index tree, saving disk I/O operations. It has been measured that the insert buffer can speed up insertions into a table up to 15 times.

15.14.3. Adaptive Hash Indexes

If a table fits almost entirely in main memory, the fastest way to perform queries on it is to use hash indexes. InnoDB has an automatic mechanism that monitors index searches made to the indexes defined for a table. If InnoDB notices that queries could benefit from building a hash index, it does so automatically.

Note that the hash index is always built based on an existing B-tree index on the table. InnoDB can build a hash index on a prefix of any length of the key defined for the B-tree, depending on the pattern of searches that InnoDB observes for the B-tree index. A hash index can be partial: It is not required that the whole B-tree index is cached in the buffer pool. InnoDB builds hash indexes on demand for those pages of the index that are often accessed.

In a sense, InnoDB tailors itself through the adaptive hash index mechanism to ample main memory, coming closer to the architecture of main memory databases.

15.14.4. Physical Record Structure

Records in InnoDB tables have the following characteristics:

- Each index record in InnoDB contains a header of six bytes. The header is used to link consecutive records together, and also in row-level locking.
- Records in the clustered index contain fields for all user-defined columns. In addition, there is a six-byte field for the transaction ID and a seven-byte field for the roll pointer.
- If no primary key was defined for a table, each clustered index record also contains a six-byte row ID field.
- Each secondary index record contains also all the fields defined for the clustered index key.
- A record contains also a pointer to each field of the record. If the total length of the fields in a record is less than 128 bytes, the pointer is one byte; otherwise, two bytes. The array of these pointers is called the record directory. The area where these pointers point is called the data part of the record.
- Internally, InnoDB stores fixed-length character columns such as CHAR(10) in a fixed-length format. InnoDB truncates trailing spaces from VARCHAR columns. Note that MySQL may internally convert CHAR columns to VARCHAR. See Section 13.2.6.1, “Silent Column Specification Changes”.
- An SQL NULL value reserves 1 or 2 bytes in the record directory. Besides that, an SQL NULL value reserves zero bytes in the data part of the record if stored in a variable length column. In a fixed-length column, it reserves the fixed length of the column in the data part of the record. The motivation behind reserving the fixed space for NULL values is that then an update of the column from

NULL to a non-NULL value can be done in place and does not cause fragmentation of the index page.

15.15. File Space Management and Disk I/O

15.15.1. Disk I/O

InnoDB uses simulated asynchronous disk I/O: InnoDB creates a number of threads to take care of I/O operations, such as read-ahead.

There are two read-ahead heuristics in InnoDB:

- In sequential read-ahead, if InnoDB notices that the access pattern to a segment in the tablespace is sequential, it posts in advance a batch of reads of database pages to the I/O system.
- In random read-ahead, if InnoDB notices that some area in a tablespace seems to be in the process of being fully read into the buffer pool, it posts the remaining reads to the I/O system.

Starting from MySQL 3.23.40b, InnoDB uses a novel file flush technique called doublewrite. It adds safety to crash recovery after an operating system crash or a power outage, and improves performance on most Unix flavors by reducing the need for fsync() operations.

Doublewrite means that before writing pages to a data file, InnoDB first writes them to a contiguous tablespace area called the doublewrite buffer. Only after the write and the flush to the doublewrite buffer has completed does InnoDB write the pages to their proper positions in the data file. If the operating system crashes in the middle of a page write, InnoDB can later find a good copy of the page from the doublewrite buffer during recovery.

15.15.2. Using Raw Devices for the Tablespace

Starting from MySQL 3.23.41, you can use raw disk partitions as tablespace data files. By using a raw disk, you can perform non-buffered I/O on Windows and on some Unix systems without filesystem overhead, which might improve performance.

When you create a new data file, you must put the keyword newraw immediately after the data file size in innodb_data_file_path. The partition must be at least as large as the size that you specify. Note that 1MB in InnoDB is 1024 * 1024 bytes, whereas 1MB usually means 1,000,000 bytes in disk specifications.

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:3Gnewraw;/dev/hdd2:2Gnewraw
```

The next time you start the server, InnoDB notices the newraw keyword and initializes the new partition. However, do not create or change any InnoDB tables yet. Otherwise, when you next restart the server, InnoDB reinitializes the partition and your changes are lost. (Starting from 3.23.44, as a safety measure InnoDB prevents users from modifying data when any partition with newraw is specified.)

After InnoDB has initialized the new partition, stop the server, change newraw in the data file specification to raw:

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:5Graw;/dev/hdd2:2Graw
```

Then restart the server and InnoDB allows changes to be made.

On Windows, starting from 4.1.1, you can allocate a disk partition as a data file like this:

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=//./D::10Gnewraw
```

The //./ corresponds to the Windows syntax of \\.\ for accessing physical drives.

When you use raw disk partitions, be sure that they have permissions that allow read and write access by the account used for running the MySQL server.

15.15.3. File Space Management

The data files you define in the configuration file form the tablespace of InnoDB. The files are simply concatenated to form the tablespace. There is no striping in use. Currently you cannot define where in the tablespace your tables are allocated. However, in a newly created tablespace, InnoDB allocates space starting from the first data file.

The tablespace consists of database pages with a default size of 16KB. The pages are grouped into extents of 64 consecutive pages. The ``files" inside a tablespace are called segments in InnoDB. The name of the ``rollback segment" is somewhat confusing because it actually contains many segments in the tablespace.

Two segments are allocated for each index in InnoDB. One is for non-leaf nodes of the B-tree, the other is for the leaf nodes. The idea here is to achieve better sequentiality for the leaf nodes, which contain the data.

When a segment grows inside the tablespace, InnoDB allocates the first 32 pages to it individually. After that InnoDB starts to allocate whole extents to the segment. InnoDB can add to a large segment up to 4 extents at a time to ensure good sequentiality of data.

Some pages in the tablespace contain bitmaps of other pages, and therefore a few extents in an InnoDB tablespace cannot be allocated to segments as a whole, but only as individual pages.

When you ask for available free space in the tablespace by issuing a SHOW TABLE STATUS, InnoDB reports the extents that are definitely free in the tablespace. InnoDB always reserves some extents for clean-up and other internal purposes; these reserved extents are not included in the free space.

When you delete data from a table, InnoDB contracts the corresponding B-tree indexes. It depends on the pattern of deletes whether that frees individual pages or extents to the tablespace, so that the freed space becomes available for other users. Dropping a table or deleting all rows from it is guaranteed to release the space to other users, but remember that deleted rows are physically removed only in an (automatic) purge operation after they are no longer needed for transaction rollbacks or consistent reads.

15.15.4. Defragmenting a Table

If there are random insertions into or deletions from the indexes of a table, the indexes may become fragmented. Fragmentation means that the physical ordering of the index pages on the disk is not close to the index ordering of the records on the pages, or that there are many unused pages in the 64-page blocks that were allocated to the index.

A symptom of fragmentation is that a table takes more space than it 'should take'. How much exactly is that, is difficult to determine. All InnoDB data and indexes are stored in B-trees, and their fillfactor may

vary 50 % - 100 %. Another symptom of fragmentation is that a table scan:

```
SELECT COUNT(*) FROM t WHERE a_non_indexed_column <> 12345;
```

takes more time than 'it should take'. (Above we are fooling the SQL optimizer to scan the clustered index, not a secondary index.) Most disks can read 10 - 50 MB/s. That can be used to estimate how fast a table scan should run.

It can speed up index scans if you periodically perform a ``null'' ALTER TABLE operation:

```
ALTER TABLE tbl_name TYPE=InnoDB
```

That causes MySQL to rebuild the table. Another way to perform a defragmention operation is to use **mysqldump** to dump the table to a text file, drop the table, and reload it from the dump file.

If the insertions to an index are always ascending and records are deleted only from the end, the InnoDB file space management algorithm guarantees that fragmentation in the index does not occur.

15.16. Error Handling

Error handling in InnoDB is not always the same as specified in the SQL standard. According to the standard, any error during an SQL statement should cause the rollback of that statement. InnoDB sometimes rolls back only part of the statement, or the whole transaction. The following items describe how InnoDB performs error handling:

- If you run out of file space in the tablespace, you get the MySQL Table is full error and InnoDB rolls back the SQL statement.
- A transaction deadlock or a timeout in a lock wait causes InnoDB to roll back the whole transaction.
- A duplicate-key error rolls back the SQL statement, if you have not specified the IGNORE option in your statement.
- A ``row too long'' error rolls back the SQL statement.
- Other errors are mostly detected by the MySQL layer of code (above the InnoDB storage engine level), and they roll back the corresponding SQL statement. Locks are not released in a rollback of a single SQL statement.

During such implicit rollbacks, as well as during the explicit ROLLBACK SQL command, SHOW PROCESSLIST displays "Rolling back" in the State column for the connection (starting from MySQL 4.1.8).

15.16.1. InnoDB Error Codes

The following is a non-exhaustive list of common InnoDB-specific errors that you may encounter, with information about why they occur and how to resolve the problem.

- 1005 (ER_CANT_CREATE_TABLE)

  Cannot create table. If the error message string refers to errno 150, table creation failed because a foreign key constraint was not correctly formed.

- 1016 (ER_CANT_OPEN_FILE)

  Cannot find the InnoDB table from the InnoDB data files though the .frm file for the table exists. See Section 15.18.1, “Troubleshooting InnoDB Data Dictionary Operations”.

- 1114 (ER_RECORD_FILE_FULL)

  InnoDB has run out of free space in the tablespace. You should reconfigure the tablespace to add a new data file.

- 1205 (ER_LOCK_WAIT_TIMEOUT)

  Lock wait timeout expired. Transaction was rolled back.

- 1213 (ER_LOCK_DEADLOCK)

  Transaction deadlock. You should rerun the transaction.

- 1216 (ER_NO_REFERENCED_ROW)

  You are trying to add a row but there is no parent row, and a foreign key constraint fails. You should add the parent row first.

- 1217 (ER_ROW_IS_REFERENCED)

  You are trying to delete a parent row that has children, and a foreign key constraint fails. You should delete the children first.

15.16.2. Operating System Error Codes

To print the meaning of an operating system error number, use the **perror** program that comes with the MySQL distribution.

The following table provides a list of some common Linux system error codes. For a more complete list, see Linux source code [http://www.iglu.org.il/lxr/source/include/asm-i386/errno.h].

- 1 (EPERM)

  Operation not permitted

- 2 (ENOENT)

  No such file or directory

- 3 (ESRCH)

  No such process

- 4 (EINTR)

  Interrupted system call

- 5 (EIO)

  I/O error

- 6 (ENXIO)

No such device or address

- 7 (E2BIG)

  Arg list too long

- 8 (ENOEXEC)

  Exec format error

- 9 (EBADF)

  Bad file number

- 10 (ECHILD)

  No child processes

- 11 (EAGAIN)

  Try again

- 12 (ENOMEM)

  Out of memory

- 13 (EACCES)

  Permission denied

- 14 (EFAULT)

  Bad address

- 15 (ENOTBLK)

  Block device required

- 16 (EBUSY)

  Device or resource busy

- 17 (EEXIST)

  File exists

- 18 (EXDEV)

  Cross-device link

- 19 (ENODEV)

  No such device

- 20 (ENOTDIR)

  Not a directory

- 21 (EISDIR)

Is a directory

- 22 (EINVAL)

  Invalid argument

- 23 (ENFILE)

  File table overflow

- 24 (EMFILE)

  Too many open files

- 25 (ENOTTY)

  Inappropriate ioctl for device

- 26 (ETXTBSY)

  Text file busy

- 27 (EFBIG)

  File too large

- 28 (ENOSPC)

  No space left on device

- 29 (ESPIPE)

  Illegal seek

- 30 (EROFS)

  Read-only file system

- 31 (EMLINK)

  Too many links

The following table provides a list of some common Windows system error codes. For a complete list see the Microsoft website [http://msdn.microsoft.com/library/default.asp?url=/library/en-us/debug/base/system_error_codes.asp].

- 1 (ERROR_INVALID_FUNCTION)

  Incorrect function.

- 2 (ERROR_FILE_NOT_FOUND)

  The system cannot find the file specified.

- 3 (ERROR_PATH_NOT_FOUND)

  The system cannot find the path specified.

- 4 (ERROR_TOO_MANY_OPEN_FILES)

  The system cannot open the file.

- 5 (ERROR_ACCESS_DENIED)

  Access is denied.

- 6 (ERROR_INVALID_HANDLE)

  The handle is invalid.

- 7 (ERROR_ARENA_TRASHED)

  The storage control blocks were destroyed.

- 8 (ERROR_NOT_ENOUGH_MEMORY)

  Not enough storage is available to process this command.

- 9 (ERROR_INVALID_BLOCK)

  The storage control block address is invalid.

- 10 (ERROR_BAD_ENVIRONMENT)

  The environment is incorrect.

- 11 (ERROR_BAD_FORMAT)

  An attempt was made to load a program with an incorrect format.

- 12 (ERROR_INVALID_ACCESS)

  The access code is invalid.

- 13 (ERROR_INVALID_DATA)

  The data is invalid.

- 14 (ERROR_OUTOFMEMORY)

  Not enough storage is available to complete this operation.

- 15 (ERROR_INVALID_DRIVE)

  The system cannot find the drive specified.

- 16 (ERROR_CURRENT_DIRECTORY)

  The directory cannot be removed.

- 17 (ERROR_NOT_SAME_DEVICE)

  The system cannot move the file to a different disk drive.

- 18 (ERROR_NO_MORE_FILES)

  There are no more files.

- 19 (ERROR_WRITE_PROTECT)

  The media is write protected.

- 20 (ERROR_BAD_UNIT)

  The system cannot find the device specified.

- 21 (ERROR_NOT_READY)

  The device is not ready.

- 22 (ERROR_BAD_COMMAND)

  The device does not recognize the command.

- 23 (ERROR_CRC)

  Data error (cyclic redundancy check).

- 24 (ERROR_BAD_LENGTH)

  The program issued a command but the command length is incorrect.

- 25 (ERROR_SEEK)

  The drive cannot locate a specific area or track on the disk.

- 26 (ERROR_NOT_DOS_DISK)

  The specified disk or diskette cannot be accessed.

- 27 (ERROR_SECTOR_NOT_FOUND)

  The drive cannot find the sector requested.

- 28 (ERROR_OUT_OF_PAPER)

  The printer is out of paper.

- 29 (ERROR_WRITE_FAULT)

  The system cannot write to the specified device.

- 30 (ERROR_READ_FAULT)

  The system cannot read from the specified device.

- 31 (ERROR_GEN_FAILURE)

  A device attached to the system is not functioning.

- 32 (ERROR_SHARING_VIOLATION)

  The process cannot access the file because it is being used by another process.

- 33 (ERROR_LOCK_VIOLATION)

  The process cannot access the file because another process has locked a portion of the file.

- 34 (ERROR_WRONG_DISK)

  The wrong diskette is in the drive. Insert %2 (Volume Serial Number: %3) into drive %1.

- 36 (ERROR_SHARING_BUFFER_EXCEEDED)

  Too many files opened for sharing.

- 38 (ERROR_HANDLE_EOF)

  Reached the end of the file.

- 39 (ERROR_HANDLE_DISK_FULL)

  The disk is full.

- 87 (ERROR_INVALID_PARAMETER)

  The parameter is incorrect. (If you get this error in MySQL-4.1.9 on Windows, and you have set innodb_file_per_table in my.cnf or my.ini, then this is Bug #8021 [http://bugs.mysql.com/8021], and a workaround is to put the line innodb_flush_method=unbuffered to your my.cnf or my.ini.)

- 112 (ERROR_DISK_FULL)

  The disk is full.

- 123 (ERROR_INVALID_NAME)

  The filename, directory name, or volume label syntax is incorrect.

- 1450 (ERROR_NO_SYSTEM_RESOURCES)

  Insufficient system resources exist to complete the requested service.

15.17. Restrictions on InnoDB Tables

- A table cannot contain more than 1000 columns.
- The internal maximum key length is 3500 bytes, but MySQL itself restricts this to 1024 bytes.
- The maximum row length, except for BLOB and TEXT columns, is slightly less than half of a database page. That is, the maximum row length is about 8000 bytes. LONGBLOB and LONGTEXT columns must be less than 4GB, and the total row length, including also BLOB and TEXT columns, must be less than 4GB. InnoDB stores the first 512 bytes of a BLOB or TEXT column in the row, and the rest into separate pages.
- On some old operating systems, data files must be less than 2GB.
- The combined size of the InnoDB log files must be less than 4GB.
- The minimum tablespace size is 10MB. The maximum tablespace size is four billion database pages (64TB). This is also the maximum size for a table.
- InnoDB tables do not support FULLTEXT indexes.
- InnoDB tables do not support spatial column types.

- ANALYZE TABLE counts cardinality by doing 10 random dives to each of the index trees and updating index cardinality estimates accordingly. Note that because these are only estimates, repeated runs of ANALYZE TABLE may produce different numbers. This makes ANALYZE TABLE fast on InnoDB tables but not 100% accurate as it doesn't take all rows into account.

  MySQL uses index cardinality estimates only in join optimization. If some join is not optimized in the right way, you may try using ANALYZE TABLE. In the few cases that ANALYZE TABLE doesn't produce values good enough for your particular tables, you can use FORCE INDEX with your queries to force the usage of a particular index, or set max_seeks_for_key to ensure that MySQL prefers index lookups over table scans. See Section 5.2.3, “Server System Variables”. See Section A.6, “Optimizer-Related Issues”.

- On Windows, InnoDB always stores database and table names internally in lowercase. To move databases in binary format from Unix to Windows or from Windows to Unix, you should have all database and table names in lowercase.

- **Warning:** Do *not* convert MySQL system tables in the mysql database from MyISAM to InnoDB tables! This is an unsupported operation. If you do this, MySQL does not restart until you restore the old system tables from a backup or re-generate them with the **mysql_install_db** script.

- InnoDB does not keep an internal count of rows in a table. (This would actually be somewhat complicated because of multi-versioning.) To process a SELECT COUNT(*) FROM T statement, InnoDB must scan an index of the table, which takes some time if the index is not entirely in the buffer pool. To get a fast count, you have to use a counter table you create yourself and let your application update it according to the inserts and deletes it does. If your table does not change often, using the MySQL query cache is a good solution. SHOW TABLE STATUS also can be used if an approximate row count is sufficient. See Section 15.12, “InnoDB Performance Tuning Tips”.

- For an AUTO_INCREMENT column, you must always define an index for the table, and that index must contain just the AUTO_INCREMENT column. In MyISAM tables, the AUTO_INCREMENT column may be part of a multi-column index.

- InnoDB does not support the AUTO_INCREMENT table option for setting the initial sequence value in a CREATE TABLE or ALTER TABLE statement. To set the value with InnoDB, insert a dummy row with a value one less and delete that dummy row, or insert the first row with an explicit value specified.

- When you restart the MySQL server, InnoDB may reuse an old value for an AUTO_INCREMENT column (that is, a value that was assigned to an old transaction that was rolled back).

- When an AUTO_INCREMENT column runs out of values, InnoDB wraps a BIGINT to -9223372036854775808 and BIGINT UNSIGNED to 1. However, BIGINT values have 64 bits, so do note that if you were to insert one million rows per second, it would still take nearly three hundred thousand years before BIGINT reached its upper bound. With all other integer type columns, a duplicate-key error results. This is similar to how MyISAM works, because it is mostly general MySQL behavior and not about any storage engine in particular.

- DELETE FROM *tbl_name* does not regenerate the table but instead deletes all rows, one by one.

- TRUNCATE *tbl_name* is mapped to DELETE FROM *tbl_name* for InnoDB and doesn't reset the AUTO_INCREMENT counter.

- SHOW TABLE STATUS does not give accurate statistics on InnoDB tables, except for the physical size reserved by the table. The row count is only a rough estimate used in SQL optimization.

- If you try to create a unique index on a prefix of a column you get an error:

  ```
  CREATE TABLE T (A CHAR(20), B INT, UNIQUE (A(5))) TYPE = InnoDB;
  ```

If you create a non-unique index on a prefix of a column, InnoDB creates an index over the whole column.

- Before MySQL 4.0.20 or 4.1.2, the MySQL LOCK TABLES operation does not know about InnoDB row-level locks set by completed SQL statements. This means that you can get a table lock on a table even if there still exist transactions by other users who have row level locks on the same table. Thus your operations on the table may have to wait if they collide with these locks of other users. Also a deadlock is possible. However, this does not endanger transaction integrity, because the row level locks set by InnoDB always take care of the integrity. Also, a table lock prevents other transactions from acquiring more row level locks (in a conflicting lock mode) on the table.

- Beginning with MySQL 4.0.20 and 4.1.2, the MySQL LOCK TABLES operation acquires two locks on each table if innodb_table_locks=1. (1 is the default.) In addition to a table lock on the MySQL layer, it also acquires an InnoDB table lock. Older versions of MySQL do not acquire InnoDB table locks. Beginning with MySQL 4.0.22 and 4.1.7, the old behavior can be selected by setting innodb_table_locks=0. If no InnoDB table lock is acquired, LOCK TABLES completes even if some records of the tables are being locked by other transactions.

- All InnoDB locks held by a transaction are released when the transaction is committed or aborted. Thus, it does not make much sense to invoke LOCK TABLES on InnoDB tables in AUTOCOMMIT=1 mode, because the acquired InnoDB table locks would be released immediately.

- Sometimes it would be useful to lock further tables in the course of a transaction. Unfortunately, LOCK TABLES in MySQL performs an implicit COMMIT and UNLOCK TABLES. An InnoDB variant of LOCK TABLES has been planned that can be executed in the middle of a transaction.

- Before MySQL 3.23.52, replication always ran with autocommit enabled. Therefore consistent reads in the slave would also see partially processed transactions, and thus the read would not be really consistent in the slave. This restriction was removed in MySQL 3.23.52.

- The LOAD TABLE FROM MASTER statement for setting up replication slave servers does not yet work for InnoDB tables. A workaround is to alter the table to MyISAM on the master, do then the load, and after that alter the master table back to InnoDB.

- The default database page size in InnoDB is 16KB. By recompiling the code, you can set it to values ranging from 8KB to 64KB. You have to update the values of UNIV_PAGE_SIZE and UNIV_PAGE_SIZE_SHIFT in the univ.i source file.

15.18. InnoDB Troubleshooting

- A general rule is that when an operation fails or you suspect a bug, you should look at the MySQL server error log, which typically has a name something like *hostname*.err, or possibly mysql.err on Windows.

- When doing troubleshooting, it is usually best to run the MySQL server from the command prompt, not through the **mysqld_safe** wrapper or as a Windows service. You then see what **mysqld** prints to the command prompt window, and you have a better grasp of what is going on. On Windows, you must start the server with the --console option to direct the output to the console window.

- Use the InnoDB Monitors to obtain information about a problem. If the problem is performance-related, or your server appears to be hung, you should use innodb_monitor to print information about the internal state of InnoDB. If the problem is with locks, use innodb_lock_monitor. If the problem is in creation of tables or other data dictionary operations, use innodb_table_monitor to print the contents of the InnoDB internal data dictionary.

- If you suspect a table is corrupt, run CHECK TABLE on that table.

15.18.1. Troubleshooting InnoDB Data Dictionary Operations

A specific issue with tables is that the MySQL server keeps data dictionary information in .frm files it stores in the database directories, while InnoDB also stores the information into its own data dictionary inside the tablespace files. If you move .frm files around, or use DROP DATABASE in MySQL versions before 3.23.44, or the server crashes in the middle of a data dictionary operation, the .frm files may end up out of sync with the InnoDB internal data dictionary.

A symptom of an out-of-sync data dictionary is that a CREATE TABLE statement fails. If this occurs, you should look in the server's error log. If the log says that the table already exists inside the InnoDB internal data dictionary, you have an orphaned table inside the InnoDB tablespace files that has no corresponding .frm file. The error message looks like this:

```
InnoDB: Error: table test/parent already exists in InnoDB internal
InnoDB: data dictionary. Have you deleted the .frm file
InnoDB: and not used DROP TABLE? Have you used DROP DATABASE
InnoDB: for InnoDB tables in MySQL version <= 3.23.43?
InnoDB: See the Restrictions section of the InnoDB manual.
InnoDB: You can drop the orphaned table inside InnoDB by
InnoDB: creating an InnoDB table with the same name in another
InnoDB: database and moving the .frm file to the current database.
InnoDB: Then MySQL thinks the table exists, and DROP TABLE will
InnoDB: succeed.
```

You can drop the orphaned table by following the instructions given in the error message.

Another symptom of an out-of-sync data dictionary is that MySQL prints an error that it cannot open a .InnoDB file:

```
ERROR 1016: Can't open file: 'child2.InnoDB'. (errno: 1)
```

In the error log you can find a message like this:

```
InnoDB: Cannot find table test/child2 from the internal data dictionary
InnoDB: of InnoDB though the .frm file for the table exists. Maybe you
InnoDB: have deleted and recreated InnoDB data files but have forgotten
InnoDB: to delete the corresponding .frm files of InnoDB tables?
```

This means that there is an orphaned .frm file without a corresponding table inside InnoDB. You can drop the orphaned .frm file by deleting it manually.

If MySQL crashes in the middle of an ALTER TABLE operation, you may end up with an orphaned temporary table inside the InnoDB tablespace. With innodb_table_monitor you see a table whose name is #sql-.... Starting from MySQL 4.0.6, you can perform SQL statements also on tables whose name contains the character '#' if you enclose the name in backticks. Thus, you can drop such an orphaned table like any other orphaned table with the method described above. Note that to copy or rename a file in the Unix shell, you need to put the file name in double quotes if the file name contains '#'.

Older MySQL versions did not allow accessing any table with a name containing '#'. The solution in older MySQL versions is to use a special InnoDB mechanism available starting from MySQL 3.23.48. When you have an orphaned table #sql-id inside the tablespace, you can cause InnoDB to rename it to rsql-id_recover_innodb_tmp_table with the following statement:

```
CREATE TABLE `rsql-id_recover_innodb_tmp_table`(...) TYPE=InnoDB;
```

Chapter 16. MySQL Cluster

MySQL Cluster is a high-availability, high-redundancy version of MySQL adapted for the distributed computing environment. It uses the **NDB Cluster** storage engine to enable running several MySQL servers in a cluster. This storage engine is available in BitKeeper from MySQL release 4.1.2, and in binary releases from MySQL-Max 4.1.3.

The operating systems on which MySQL Cluster is currently available are Linux, Mac OS X, and Solaris. We are working to make Cluster run on all operating systems supported by MySQL, including Windows.

This chapter represents a work in progress, and its contents are subject to revision as MySQL Cluster continues to evolve. Additional information regarding MySQL Cluster can be found on the MySQL AB Web site at http://www.mysql.com/products/cluster/.

You may wish to make use of two additional online resources provided by MySQL AB:

- the MySQL Cluster http://lists.mysql.com/cluster [mailing list].
- the Cluster topic area [http://forums.mysql.com/list.php?25] on the MySQL User Forums.

Answers to some commonly-asked questions about Cluster may be found in the Section 16.10, “MySQL Cluster FAQ”. If you are new to MySQL Cluster, you may also find our Developer Zone article How to set up a MySQL Cluster for two servers [http://dev.mysql.com/tech-resources/articles/mysql-cluster-for-two-servers.html] to be helpful.

16.1. MySQL Cluster Glossary

The following terms are useful to an understanding of MySQL Cluster or have specialised meanings when used in relation to it.

- **Cluster**: In its generic sense, a cluster is a set of computers functioning as a unit and working together to accomplish a single task. **NDB Cluster** is the storage engine used by MySQL to implement data storage, retrieval, and management distributed amongst several computers. **MySQL Cluster** refers to a group of computers working together using the NDB engine to support a distributed MySQL database in a **shared-nothing architecture** using **in-memory storage**.
- **Configuration files**: Text files containing directives and information regarding the cluster, its hosts, and its nodes. These are read by the cluster's management nodes when the cluster is started. See Section 16.5.4, “Configuration File” for details.
- **Backup**: A complete copy of all cluster data, transactions and logs, saved to disk or other long-term storage.
- **Restore**: Returning the cluster to a previous state as stored in a backup.
- **Checkpoint**: Generally speaking, when data is saved to disk, it is said that a checkpoint has been reached. More specific to Cluster, it is a point in time where all committed transactions are stored on disk. With regard to the NDB storage engine, there are two sorts of checkpoints which work together to ensure that a consistent view of the cluster's data is maintained:
  - **Local Checkpoint (LCP)**: This is a checkpoint that is specific to a single node; however, LCP's take place for all nodes in the cluster more or less concurrently. An LCP involves saving all of a

node's data to disk, and so usually occurs every few minutes. The precise interval varies, and depends upon the amount of data stored by the node, the level of cluster activity, and other factors.

  - **Global Checkpoint (GCP)**: A GCP occurs every few seconds, when transactions for all nodes are synchronised and the redo-log is flushed to disk.

- **Cluster host**: A computer making up part of a MySQL Cluster. A cluster has both a *physical* structure and a *logical* structure. Physically, the cluster consists of a number of computers, known as **cluster hosts** (or more simply as **hosts**). See also **Node**, **Node group**.

- **Node**: This refers to a logical or functional unit of MySQL Cluster, sometimes also referred to as a **cluster node**. In the context of MySQl Cluster, we use the term **node** to indicate a *process* rather than a physical component of the cluster. There are three node types required to implement a working MySQL Cluster. These are:

  - **Management (MGM) nodes**: Manages the other nodes within the MySQL Cluster. It provides configuration data to the other nodes; starts and stops nodes; handles network partitioning; creates backups and restores from them, and so forth.

  - **SQL (MySQL server) nodes**: Instances of MySQL Server which serve as front ends to data kept in the cluster's **storage nodes**. Clients desiring to store, retrieve, or update data can access an SQL node just as they would any other MySQL Server, employing the usual authentication methods and API's; the underlying distribution of data between node groups is transparent to users and applications. SQL nodes access the cluster's databases as a whole without regard to the data's distribution across different storage nodes or cluster hosts.

  - **Data nodes** (also referred to as **storage nodes**): These nodes store the actual data. Table data fragments are stored in a set of node groups; each node group stores a different subset of the table data. Each of the nodes making up a node group stores a replica of the fragment for which that node group is responsible. Currently a single cluster can support a total of up to 48 data nodes.

  It is possible for more than one node to co-exist on a single machine. (In fact, it is even possible to set up a complete cluster on one machine, although one would almost certainly not want to do this in a production environment.) It may be helpful to remember that, when working with MySQL Cluster, **host** refers to a physical component of the cluster whereas a **node** is a logical or functional component (that is, a process).

  Note Regarding Obsolete Terms: In older versions of the MySQL Cluster documentation, data nodes were sometimes referred to as "Database nodes" or "DB nodes". In addition, SQL nodes were sometimes known as "client nodes" or "API nodes". This older terminology has been deprecated in order to minimize confusion, and for these reasons should be avoided.

- **Node group**: A set of data nodes. All data nodes in a node group contain the same data (fragments), and each node in a group should reside on a different host. It is possible to control which nodes belong to which node groups.

- **Node failure**: MySQL Cluster is not solely dependent upon the functioning of any single node making up the cluster; the cluster can continue to run if one or more nodes fail. The precise number of node failures that the cluster can tolerate depends upon the number of nodes and the cluster's configuration.

- **Node restart**: The process of restarting a failed cluster node.

- **Initial node restart**: The process of starting a cluster node with its filesystem removed. This is sometimes used in the course of software upgrades and in other special circumstances.

- **System crash** (or **system failure**): This can occur when so many cluster nodes have failed that the

cluster's state can no longer be guaranteed.

- **System restart**: The process of restarting the cluster and reinitialising its state from disk logs and checkpoints. This is required after either a planned or an unplanned shutdown of the cluster.

- **Fragment**: A portion of a database table; in the NDB storage engine, a table is broken up into and stored as a number of fragments. A fragment is sometimes also called a **partition**; however, "fragment" is the preferred term. Tables are fragmented in MySQL Cluster in order to facilitate load balancing between machines and nodes.

- **Replica**: Under the NDB storage engine, each table fragment has number of replicas stored on other storage nodes in order to provide redundancy. Currently there may be up 4 replicas per fragment.

- **Transporter**: A protocol providing data transfer between nodes. MySQL Cluster currently supports 4 different types of transporter connections: **TCP/IP** (local), TCP/IP (remote), **SCI**, and **SHM** (experimental in MySQL 4.1).

  - **TCP/IP** is, of course, the familiar network protocol that underlies HTTP, FTP, etc., on the Internet.

  - **SCI** (**S**calable **C**oherent **I**nterface) is a high-speed protocol used in building multiprocessor systems and parallel-processing applications. Use of SCI with MySQL Cluster requires specialised hardware and is discussed in Section 16.8.1, “Configuring MySQL Cluster to use SCI Sockets”. For a basic introduction to SCI, see this essay at dolphinics.com [http://www.dolphinics.com/corporate/scitech.html].

  - **SHM** stands for Unix-style **sh**ared **m**emory segments. Where supported, SHM is used automatically to connect nodes running on the same host. This is experimental in MySQL 4.1, but we intend to enable it fully in MySQL 5.0. The Unix man page for `shmop(2)` [http://www.scit.wlv.ac.uk/cgi-bin/mansec?2+shmop] is a good place to begin obtaining additional information about this topic.

  Note: The cluster transporter is internal to the cluster. Applications using MySQL Cluster communicate with SQL nodes just as they do with any other version of MySQL Server (via TCP/IP or Windows named pipes/Unix sockets). Queries can be sent and results retrieved using the standard APIs.

- **NDB**: Refers to the storage engine used to enable MySQL Cluster. The NDB storage engine supports all the usual MySQL column types and SQL statements, and is ACID-compliant. This engine also provides full support for transactions (commits and rollbacks). "NDB" stands for **N**etwork **D**ata**b**ase.

- **Share-nothing architecture**: The ideal architecture for a MySQL Cluster. In a true share-nothing setup, each node runs on a separate host. The advantage such an arrangement is that there no single host or node can act as single point of failure or as a performance bottle neck for the system as a whole.

- **In-memory storage**: All data stored in each data node is kept in memory on the node's host computer. For each data node in the cluster, you must have available an amount of RAM equal to the size of the database times the number of replicas, divided by the number of data nodes. Thus, if the database takes up 1 gigabyte of memory, and you wish to set up the cluster with 4 replicas and 8 data nodes, a minimum of 500 MB memory will be required per node. Note that this is in addition to any requirements for the operating system and any applications running on the host.

- **Table**: As is usual in the context of a relational database, the term "table" denotes an ordered set of identically-structured records. In MySQL Cluster, a database table is stored in a data node as a set of fragments, each of which is replicated on additional data nodes. The set of data nodes replicating the same fragment or set of fragments is referred to as a **node group**.

- **Cluster programs**: These are command-line programs used in running, configuring and administer-

ing MySQL Cluster. They include both server daemons:

- `ndbd`: The data node daemon (runs a data node process)
- `ndb_mgmd`: The management server daemon (runs a management server process)

and client programs:

- `ndb_mgm`: The management client (provides an interface for executing management commands)
- `ndb_waiter`: Used to verify status of all nodes in a cluster
- `ndb_restore`: Restores cluster data from backup

For more about these programs and their uses, see Section 16.6, “Process Management in MySQL Cluster”.

- **Event log**: MySQL Cluster logs events by category (startup, shutdown, errors, checkpoints, etc.), priority, and severity. A complete listing of all reportable events may be found in Section 16.7.2, “Event Reports Generated in MySQL Cluster”. Event logs are of two types:
  - **Cluster log**: Keeps a record of all desired reportable events for the cluster as a whole.
  - **Node log**: A separate log is also kept for each individual node.

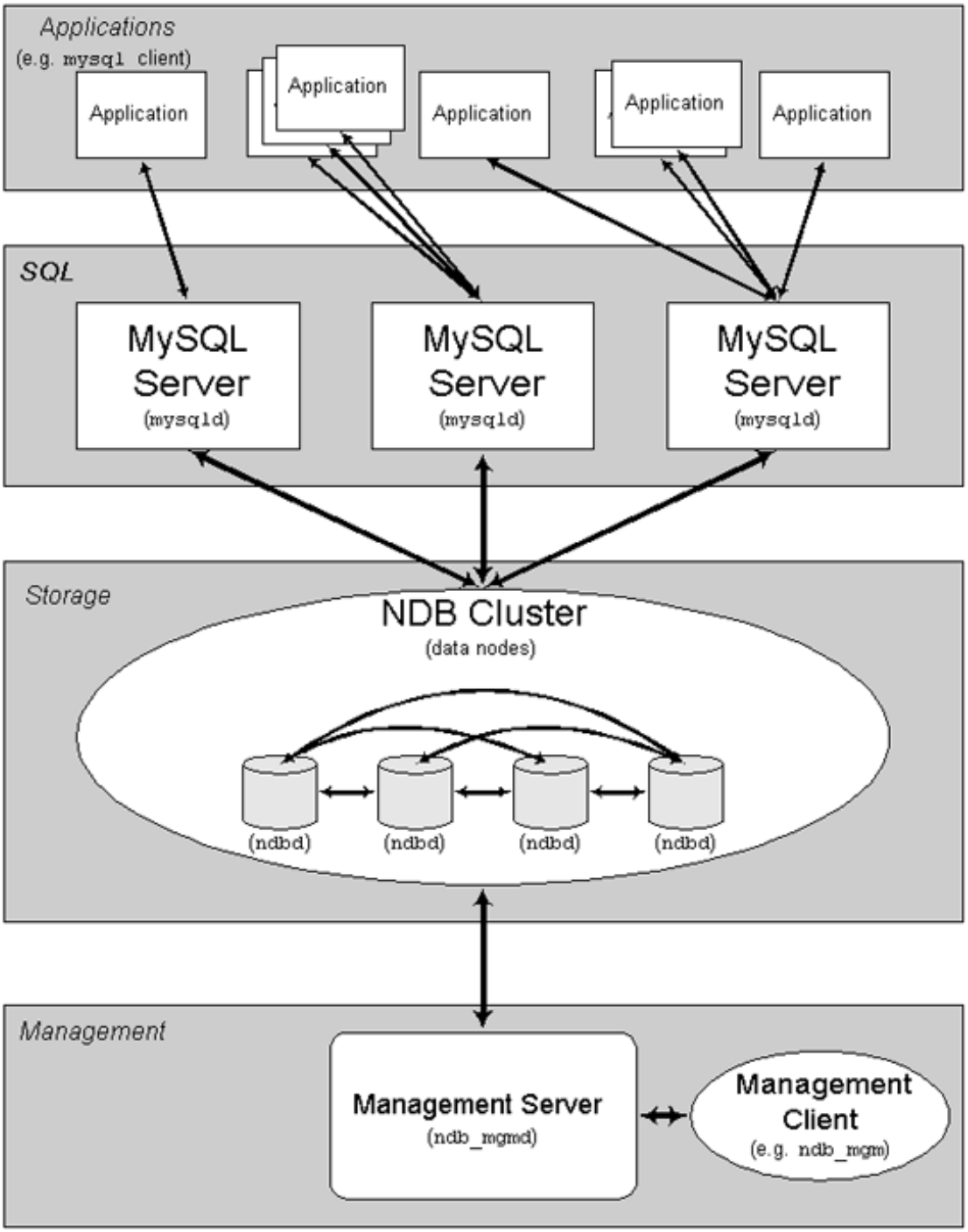
  Under normal circumstances, it is necessary and sufficient to keep and examine only the cluster log. The node logs need be comsulted only for application development and debugging purposes.

16.2. MySQL Cluster Overview

MySQL Cluster is a technology which enables clustering of in-memory databases in a share-nothing system. The share-nothing architecture allows the system to work with very inexpensive hardware, and without any specific requirements on hardware or software. It also does not have any single point of failure because each component has its own memory and disk.

MySQL Cluster integrates the standard MySQL server with an in-memory clustered storage engine called **NDB**. In our documentation, the term **NDB** refers to the part of the setup that is specific to the storage engine, whereas **MySQL Cluster** refers to the combination of MySQL and the new storage engine.

A MySQL Cluster consists of a set of computers, each running a number of processes including MySQL servers, storage nodes for NDB Cluster, management servers, and (possibly) specialised data access programs. The relationship of these components in a cluster are shown here:



All these programs work together to form MySQL Cluster. When data is stored in the NDB Cluster storage engine, the tables are stored in the storage nodes for NDB Cluster. Such tables are directly accessible from all other MySQL servers in the cluster. Thus, in a payroll application storing data in a cluster, if one application updates the salary of an employee, all other MySQL servers that query this data can see this change immediately.

The data stored in the storage nodes for MySQL Cluster can be mirrored; the cluster can handle failures of individual storage nodes with no other impact than that a small number of transactions are aborted due to losing the transaction state. Since transactional applications are expected to handle transaction

failure, this should not be a source of problems.

By bringing MySQL Cluster to the Open Source world, MySQL makes clustered data management with high availability, high performance, and scalability available to all who need it.

16.3. Basic MySQL Cluster Concepts

NDB is an in-memory storage engine offering high-availability and data-persistence features.

The NDB storage engine can be configured with a range of failover and load-balancing options, but it is easiest to start with the storage engine at the cluster level. MySQL Cluster's NDB storage engine contains a complete set of data, dependent only on other data within the cluster itself.

We will now describe how to set up a MySQL Cluster consisting of an NDB storage engine and some MySQL servers.

The cluster portion of MySQL Cluster is currently configured independently of the MySQL servers. In a MySQL Cluster, each part of the cluster is considered to be a **node**.

Note: In many contexts, the term "node" is used to indicate a computer, but when discussing MySQL Cluster it means a *process*. There can be any number of nodes on a single computer, for which we use the term **cluster host**.

There are three types of cluster nodes, and in a minimal MySQL Cluster configuration, there will be at least three nodes, one of each of these types:

- The management (`MGM`) node: The role of this type of node is to manage the other nodes within the MySQL Cluster, such as providing configuration data, starting and stopping nodes, running backup, and so forth. Because this node type manages the configuration of the other nodes, a node of this type should be started first, before any other node. A MGM node is started with the command **ndb_mgmd**.
- The **storage node** (or **data node**): This is the type of node that stores the cluster's data. There are as many storage nodes as there are replicas, times the number of fragments. For example, with two replicas, each having two fragments, you will need four storage nodes. It is not necessary to have more than one replica. A storage node is started with the command **ndbd**.
- The **SQL node**: This is the node that accesses the cluster data. In the case of MySQL Cluster, a client node is a traditional MySQL server that uses the NDB Cluster storage engine. An SQL node is typically started with the command **mysqld --ndb-cluster** or simply by using **mysqld** with `ndb-cluster` added to `my.cnf`.

Configuration of a cluster involves configuring each individual node in the cluster and setting up individual communication links between nodes. MySQL Cluster is currently designed with the intention that storage nodes are homogenous in terms of processor power, memory space, and bandwidth. In addition, in order to provide a single point of configuration, all configuration data for the cluster as a whole is located in one configuration file.

The management server (MGM node) manages the cluster configuration file and the cluster log. Each node in the cluster retrieves the configuration data from the management server, and so requires a way to determine where the management server resides. When interesting events occur in the storage nodes, the nodes transfer information about these events to the management server, which then writes the information to the cluster log.

In addition, there can be any number of cluster client processes or applications. These are of two types:

- **Standard MySQL clients**: These are no different for MySQL Cluster than they are for standard (non-Cluster) MySQL. In other words, MySQL Cluster can be accessed from existing MySQL applications written in PHP, Perl, C, C++, Java, Python, Ruby, and so on.
- **Management clients**: These clients connect to the management server and provide commands for starting and stopping nodes gracefully, starting and stopping message tracing (debug versions only), showing node versions and status, starting and stopping backups, and so on.

16.4. Simple Multi-Computer How-To

This section is a "How-To" in which we will describe the basics for how to plan, install, configure, and run a viable MySQL Cluster. Unlike the example in Section 16.5, “MySQL Cluster Configuration”, the result of following the guidelines and procedures outlined below should be a usable MySQL Cluster which meets minimum requirements for availability and safeguarding of data.

In this section, we will cover: hardware and software requirements; networking issues; installation of MySQL Cluster; configuration issues; starting, stopping, and restarting the cluster; loading of a sample database; and performing queries.

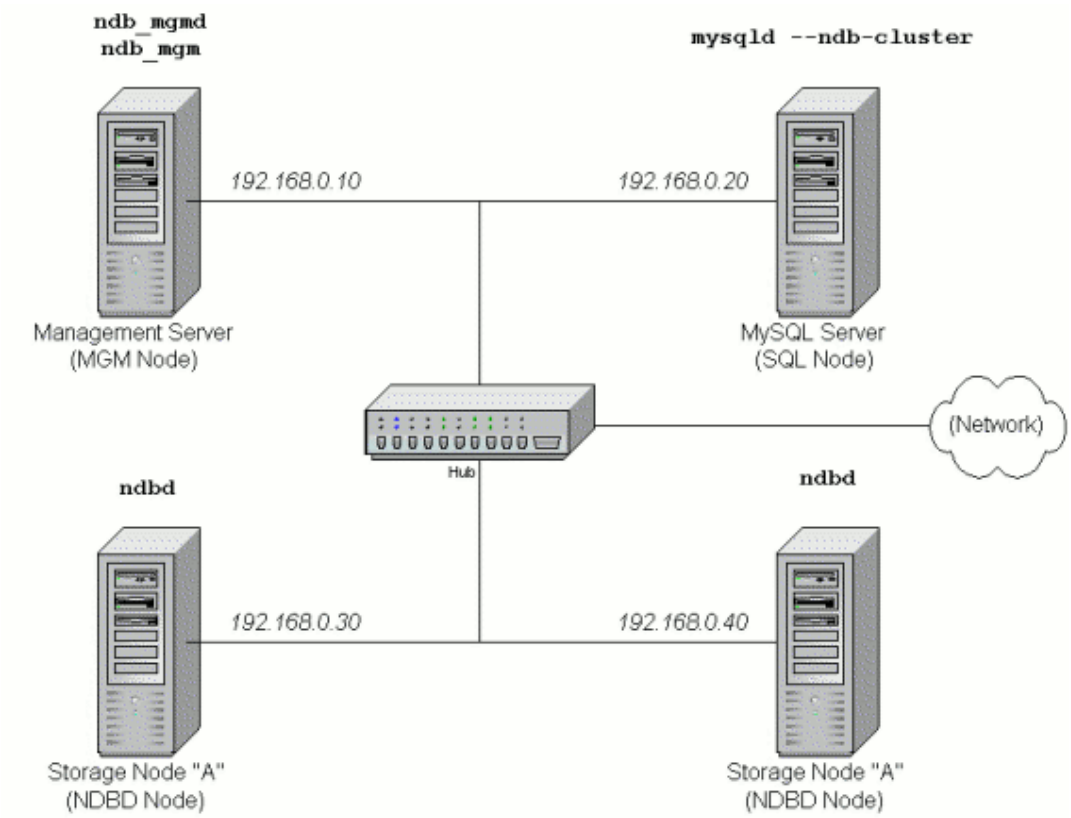
Basic Assumptions

This How-To makes the following assumptions:

1. We are setting up our cluster with 4 nodes, each on a separate host, and each with a fixed network address on a typical Ethernet as shown here:

Node	IP Address
Management (MGM) node	192.168.0.10
MySQL server (SQL) node	192.168.0.20
Storage (NDBD) node "A"	192.168.0.30
Storage node "B"	192.168.0.40

This may be made clearer in the following diagram:



Note: In the interest of simplicity (and reliability), we will use only numeric IP addresses in this How-To. However, if DNS resolution is available on your network, then it is possible to use hostnames in lieu of IP addresses in configuring Cluster. Alternatively, you can also use the `/etc/hosts` file or your operating system's equivalent for providing a means to do host lookup if such is available.

2. Each host is an Intel-based desktop PC running a common, generic Linux distribution installed to disk in a standard configuration, and with no unnecessary services. The core OS with a standard TCP/IP networking client should be sufficient. Also for the sake of simplicity, we also assume that the filesystems on all hosts are set up identically. In the event that they are not, you will need to adapt these instruciton that follow accordingly.

3. A standard 100 Mbps Ethernet card is installed on each machine, along with the proper driver for that card, and that all 4 hosts are connected via a standard-issue Ethernet networking applicance such as a switch.

4. For our sample data, we will use the **world** database which is available for download from the MySQL AB website. As this database takes up a relatively small amount of space, we assume that each machine has 256 MB RAM, which should be sufficient for running the operating system and host NDB process, and (for the storage nodes) for storing the database.

While we refer to a Linux operating system in this How-To, the instructions and procedures that we provide here should be easily adaptable to either Solaris or Mac OS X. We also assume that you already know how to perform a minimal installation and configuration of the operating system with networking capability, or that you are able to obtain assistance in this elsewhere if needed.

We discuss MySQL Cluster hardware, software, and networking requirements in somewhat greater detail in the next section. (See Section 16.4.1, “Hardware, Software, and Networking”.)

16.4.1. Hardware, Software, and Networking

One of the strengths of MySQL Cluster is that it can be run on commodity hardware and has no ususual requirements in this regard, other than for large amounts of RAM, due to the fact that all live data storage is done in memory. (Note that this is subject to change, and that we intend to implement disk-based memory in a future MySQL Cluster release.) Naturally, multiple and faster CPUs will enhance performance. Memory requirements for Cluster processes are relatively small.

The software requirements for Cluster are also modest. Host operating systems do not require any unusual modules, services, applications, or configuration to support MySQL Cluster. For Mac OS X or Solaris, the standard installation is sufficient. For Linux, a standard, "out of the box" installation should be all that is necessary. The MySQL software requirements are simple: all that is needed is a production release of MySQL-max 4.1.3 or newer; you must use the `-max` version of MySQL in order to have Cluster support. It is not necessary to compile MySQL yourself merely to be able to use Cluster. In this How-To, we will assume that you are using the `-max` binary appropriate to your operating system, available via the MySQL software downloads page at http://dev.mysql.com/downloads.

For inter-node communication, Cluster supports TCP/IP networking in any standard topology, and all that is expected for each host is a standard 100 Mbps Ethernet card, plus a switch, hub, or router to provide network connectivity for the cluster as a whole. We recommend that, for reasons of efficiency, a MySQL Cluster be run on its own subnet, which is not shared with non-Cluster machines. This is not a firm requirement; however, setting up a Cluster in this fashion allows for the cluster to make exclusive use of bandwidth between cluster hosts. Using a separate switch between cluster nodes ensures that they are shielded from other computers' communications. For even greater reliability, you can use dual switches and dual cards to remove the network as a single point of failure. Many device drivers support failover for such communication links. In addition, it is worth remembering that inter-node communication is not encrypted or otherwise protected; this means that you should not run Cluster on an open network. If you intend to use Cluster for Web applications, the cluster should definitely reside behind your firewall and not in your network's De-Militarised Zone (DMZ [http://compnetworking.about.com/cs/networksecurity/g/bldef_dmz.htm]).

It is possible to use the high-speed Scalable Coherent Interface (SCI) with MySQL Cluster, but this is not a requirement. See Section 16.8, “Using High-Speed Interconnects with MySQL Cluster” for more about this protocol and its use with MySQL Cluster.

16.4.2. Installation

Each MySQL Cluster host computer running storage or SQL nodes must have installed on it a MySQL-max binary. For management nodes, it is not necessary to install the MySQL server binary, but you do have to install the MGM server daemon and client binaries (**ndb_mgmd** and **ndb_mgm**, respectively). In this section, we will cover the steps necessary to install the correct binaries for each type of Cluster node.

As of this writing, the most recent production version for Linux was MySQL 4.1.9; if a more recent version is available, you should use that instead, and subsitute the version number in the following instrucitons as appropriate. MySQL AB provides precompiled binaries, and there is generally no need to compile these yourself. (If you do require a custom binary, see Section 2.8.3, “Installing from the Development Source Tree”.) Therefore, the first step in the installation process for each cluster host is to download the file `mysql-max-4.1.9-pc-linux-gnu-i686.tar.gz` from the MySQL downloads area [http://dev.mysql.com/downloads], and we will assume that you have placed it in each machine's `/usr/local` directory.

Note: After completing the installation, do not yet start any of the binaries. We will show you how to do so following the configuration of all nodes.

Storage and SQL Node Installation

On each of the 3 machines designated to host storage or SQL nodes, perform the following steps as the

system root user:

1. Create a new `mysql` user group, then add a `mysql` user to this group:

   ```
   groupadd mysql
   useradd -g mysql mysql
   ```

2. Change to the directory containing the downloaded file; unpack and then delete the archive; create a symlink to the mysql-max executable:

   ```
   cd /usr/local
   tar -zxvf mysql-max-4.1.9-pc-linux-gnu-i686.tar.gz
   rm mysql-max-4.1.9-pc-linux-gnu-i686.tar.gz
   ln -s mysql-max-4.1.9-pc-linux-gnu-i686 mysql
   ```

3. Change to the `mysql` directory, and run the supplied script for creating the system databases:

   ```
   cd mysql
   scripts/mysql_install_db --user=mysql
   ```

4. Set the necessary permissions for the MySQL server and data directories:

   ```
   chown -R root .
   chown -R mysql data
   chgrp -R mysql .
   ```

   Note that the data directory on each machine hosting a storage node is `/usr/local/mysql/data`. We will make use of this bit of information when we configure the management node. (See Section 16.4.3, “Configuration”.)

5. Copy the MySQL startup script to the appropriate directory, make it executable, and set it to start when the operating system is booted up:

   ```
   cp support-files/mysql.server /etc/rc.d/init.d/
   chmod +x /etc/rc.d/init.d/mysql.server
   chkconfig --add mysql.server
   ```

   Here we use Red Hat's **chkconfig** for creating links to the startup scripts; use whatever means is appropriate for this purpose on your operating system/distribution, such as **update-rc.d** on Debian.

Remember that the steps listed above must be performed separately for each machine on which a storage or SQL node is to reside.

Management Node Installation

For the MGM (management) node, it is not necessary to install the **mysqld** executable, only the binaries for the MGM server and client, which can be found in the downloaded `-max` archive. Again we assume that you have placed this file in `/usr/local`. As root, perform the following steps to install **ndb_mgmd** and **ndb_mgm** on the Cluster management node host:

1. Move to the `/usr/local` directory, unpack the archive, then delete the archive and change to directory into which the archive was unpacked:

```
cd /usr/local
tar -zxvf mysql-max-4.1.9-pc-linux-gnu-i686.tar.gz
rm mysql-max-4.1.9-pc-linux-gnu-i686.tar.gz
cd mysql-max-4.1.9-pc-linux-gnu-i686
```

2. Move the **ndb_mgmd** and **ndb_mgm** binaries from the `bin` directory into the current directory, make both of these executable, then move them to the system's `/usr/bin` directory.

```
mv bin/ndb_mgm .
mv bin/ndb_mgmd .
chmod +x ndb_mg*
mv ndb_mg* /usr/bin/
```

3. Finally, delete the directory created in the first step by extracting the archive, as it is no longer required:

```
cd
rm -rf /usr/local/mysql-max-4.1.9-pc-linux-gnu-i686
```

In Section 16.4.3, “Configuration”, we will create and write configuration files for all of the nodes in our example Cluster.

16.4.3. Configuration

For our 4-node, 4-host MySQL Cluster, we will need to write 4 configuration files, 1 per node/host.

- Each storage node or SQL node will require a `my.cnf` file that provides two pieces of information: a **connectstring** telling the node where to find the MGM node, and a line telling the MySQL server on this host (the machine hosting the data node) to run in NDB mode.

  For more information on connectstrings, see Section 16.5.4.2, “The MySQL Cluster `connectstring`”.

- The management node will need a `config.ini` file telling it how many replicas are to be maintained, how much memory to allocate for data and indexes on each storage node, where to find the storage nodes, where data will be saved to disk on each storage node, and where to find any SQL nodes.

Configuring the Storage and SQL Nodes

The `my.cnf` file needed for the storage nodes is fairly simple. The configuration file should be located in the `/etc` directory and can be edited (and created if necessary) using any text editor, for example:

```
vi /etc/my.cnf
```

For each storage node and SQL node in our example setup, `my.cnf` should look like this:

```
[MYSQLD]                        # Options for mysqld process:
ndbcluster                      # run NDB engine
ndb-connectstring=192.168.0.10  # location of MGM node

[MYSQL_CLUSTER]                 # Options for ndbd process:
```

```
ndb-connectstring=192.168.0.10  # location of MGM node
```

After entering the above, save this file and exit the text editor. Do this for the machines hosting storage node "A", storage node "B", and the SQL node.

Configuring the Management Node

The **ndb_mgmd** and **ndb_mgm** processes running on the MGM node will expect to find their configuration data in the file `/var/lib/mysql-cluster/config.ini`. Therefore, the first step in configuring the MGM node is to create the directory in which this file can be found and then create the file itself. For example (running as root):

```
mkdir /var/lib/mysql-cluster
cd /var/lib/mysql-cluster
vi config.ini
```

We show **vi** being used here to create the file, but any text editor should work just as well.

For our representative setup, the `config.ini` file should read as follows:

```
[NDBD DEFAULT]    # Options affecting ndbd processes on all storage nodes:
NoOfReplicas=2    # Number of replicas
DataMemory=80M    # How much memory to allocate for data storage
IndexMemory=52M   # How much memory to allocate for index storage
                  # For DataMemory and IndexMemory, we have used the
                  # default values. Since the "world" database takes up
                  # only about 500KB, this should be more than enough for
                  # this example Cluster setup.

[TCP DEFAULT]     # TCP/IP options:
portnumber=2202   # This the default; however, you can use any
                  # port that is free for all the hosts in cluster
                  # Note: It is recommended beginning with MySQL 5.0 that
                  # you do not specify the portnumber at all and simply allow
                  # the default value to be used instead

[NDB_MGMD]                        # Management process options:
hostname=192.168.0.10             # Hostname or IP address of MGM node
datadir=/var/lib/mysql-cluster    # Directory for MGM node logfiles

[NDBD]                            # Options for storage node "A":
                                  # (one [NDBD] section per storage node)
hostname=192.168.0.30             # Hostname or IP address
datadir=/usr/local/mysql/data     # Directory for this storage node's datafiles

[NDBD]                            # Options for storage node "B":
hostname=192.168.0.40             # Hostname or IP address
datadir=/usr/local/mysql/data     # Directory for this storage node's datafiles

[MYSQLD]                          # SQL node options:
hostname=192.168.0.20             # Hostname or IP address
datadir=/usr/local/mysql/data     # Directory for SQL node's datafiles
                                  # (additional mysqld connections can be
                                  # specified for this node for various
                                  # purposes such as running ndb_restore)
```

Once all the configuration files have been created and these minimal options have been specified, you are ready to proceed with starting the cluster and verifying that all processes are running. We discuss

how this is done in Section 16.4.4, “Initial Startup”.

For more detailed information about the available MySQL Cluster configuration parameters and their uses, see Section 16.5.4, “Configuration File” and Section 16.5, “MySQL Cluster Configuration”. For configuration of MySQL Cluster as relates to making backups, see Section 16.7.4.4, “Configuration for Cluster Backup”.

Note: The default port for Cluster management nodes is 1186; the default port for storage nodes is 2202.

16.4.4. Initial Startup

Starting the cluster is not very difficult once it has been configured. Each cluster node process must be started separately, and on the host where it resides. While it is possible to start the nodes in any order, it is recommended that the management node be started first, followed by the storage nodes, and then finally by any SQL nodes:

1. On the management host, issue the following command from the system shell to start the MGM node process:

   ```
   shell> ndb_mgmd
   ```

2. On each of the storage node hosts, run this command to start the NDBD process for the first time:

   ```
   shell> ndbd --initial
   ```

   Note that it is very important to use the `--initial` parameter **only** when starting **ndbd** for the first time, or when restarting after backup/restore or configuration change. This is because this parameter will cause the node to delete any files created by earlier **ndbd** instances needed for recovery, including the recovery log files.

3. On the cluster host where the SQL node is to reside, execute a normal **mysqld** startup command as shown here:

   ```
   shell> mysqld &
   ```

If all has gone well, and the cluster has been set up correctly, the cluster should now be operational. You can test this by invoking the **ndb_mgm** management node client; the output should look like what is shown below:

```
shell> ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> show
Connected to Management Server at: localhost:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=2    @192.168.0.30  (Version: 4.1.11, Nodegroup: 0, Master)
id=3    @192.168.0.40  (Version: 4.1.11, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @192.168.0.10  (Version: 4.1.11)

[mysqld(SQL)]   1 node(s)
id=4   (Version: 4.1.11)
```

You may see some slight differences depending upon the exact version of MySQL that you are using.

Note: If you are using an older version of MySQL, you may see the SQL node referenced as '`[mysqld(API)]`'. This reflects an older usage which is now deprecated.

You should now be ready to work with databases, tables, and data in MySQL Cluster. See Section 16.4.5, “Loading Sample Data and Performing Queries” for a brief discussion.

16.4.5. Loading Sample Data and Performing Queries

Working with data in MySQL Cluster is not much different than working with it MySQL without Cluster. There are two points to keep in mind when doing so:

- Tables must be created with the `ENGINE=NDB` or `ENGINE=NDBCLUSTER` option in order to have them replicated in the cluster. If you are importing tables from an existing database using the output of **mysqldump**, you should first open the SQL script(s) in a text editor and add this option to any table creation statements, or replace any existing `ENGINE` (or `TYPE`) option with one of these. For example, suppose that you have the sample **world** database on another MySQL server, and you wish to export the definition for the `CITY` table:

  ```
  shell> mysqldump --add-drop-table world City < city_table.sql
  ```

  The resulting `city_table.sql` file will contain this table creation statement (and the `INSERT` statements necessary to import the table data):

  ```
  DROP TABLE IF EXISTS City;
  CREATE TABLE City (
  ID int(11) NOT NULL auto_increment,
  Name char(35) NOT NULL default '',
  CountryCode char(3) NOT NULL default '',
  District char(20) NOT NULL default '',
  Population int(11) NOT NULL default '0',
  PRIMARY KEY  (ID)
  ) TYPE=MyISAM;

  INSERT INTO City VALUES (1,'Kabul','AFG','Kabol',1780000);
  INSERT INTO City VALUES (2,'Qandahar','AFG','Qandahar',237500);
  INSERT INTO City VALUES (3,'Herat','AFG','Herat',186800);
  # (remaining INSERT statements omitted)
  ```

  Before importing this into the Cluster database, you will need to modify it so that it reads:

  ```
  DROP TABLE IF EXISTS City;
  CREATE TABLE City (
  ID int(11) NOT NULL auto_increment,
  Name char(35) NOT NULL default '',
  CountryCode char(3) NOT NULL default '',
  District char(20) NOT NULL default '',
  Population int(11) NOT NULL default '0',
  PRIMARY KEY  (ID)
  ) ENGINE=NDBCLUSTER;

  INSERT INTO City VALUES (1,'Kabul','AFG','Kabol',1780000);
  INSERT INTO City VALUES (2,'Qandahar','AFG','Qandahar',237500);
  INSERT INTO City VALUES (3,'Herat','AFG','Herat',186800);
  # (etc.)
  ```

Assuming that you have already created a database named **world** on the SQL node of the cluster, you can then use the **mysql** command-line client to read `city_table.sql`, and create and populate the corresponding table in the usual manner:

```
shell> mysql world < city_table.sql
```

It is very important to keep in mind that the above command must be executed on the host where the SQL node is running -- in this case, on the machine with the IP address **192.168.0.20**.

To create a copy of the **world** database on the SQL node, simply do a search-and-replace on the `world.sql` file, replacing all instances of `TYPE=MyISAM` with `ENGINE=NDBCLUSTER`, save the file to `/usr/local/mysql/data`, then run

```
shell> cd /usr/local/mysql/data
shell> mysql world < world.sql
```

Of course, the SQL script must be readable by the **mysql** user. If you save the file to a different location, adjust the above accordingly.

Running **SELECT** queries on the SQL node is no different than running them on any other instance of a MySQL server. To run queries from the command line, you first need to log in to the MySQL Monitor in the usual way:

```
shell> mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 4.1.9-max

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Note that we simply use the default root account with its empty password here. Of course, in a production setting, you should always follow the standard security precautions for installing a MySQL server, including the setting of a strong root password and creation of a user account with only those privileges required to accomplish necessary tasks by that user. For more information about these, see Section 5.5, “The MySQL Access Privilege System”.

It is worth taking into account that Cluster nodes do not make use of the MySQL privileges system when accessing one another, and setting or changing MySQL user accounts (including the root account) has no effect on interaction between nodes, only on applications accessing the SQL node.

Selecting a database and running a **SELECT** query against a table in that database is also accomplished in the usual manner, as is exiting the MySQL Monitor:

```
mysql> USE world;
mysql> SELECT Name, Population FROM City ORDER BY Population DESC LIMIT 5;
+-----------+------------+
| Name      | Population |
+-----------+------------+
| Bombay    |   10500000 |
| Seoul     |    9981619 |
| SÃ£o Paulo |    9968485 |
| Shanghai  |    9696300 |
| Jakarta   |    9604900 |
+-----------+------------+
5 rows in set (0.34 sec)
```

```
mysql> \q
Bye

shell>
```

Applications using MySQL can use standard APIs. It is important to remember that your application must access the SQL node, and not the MGM or storage nodes. This brief example shows how we might execute the same query as above using PHP 5's **mysqli** extension running on a Web server elsewhere on the network:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1">
  <title>SIMPLE mysqli SELECT</title>
</head>
<body>
<?php
  # connect to SQL node:
  $link = new mysqli('192.168.0.20', 'root', '', 'world');
  # parameters for mysqli constructor are:
  #   host, user, password, database

  if( mysqli_connect_errno() )
    die("Connect failed: " . mysqli_connect_error());

  $query = "SELECT Name, Population
            FROM City
            ORDER BY Population DESC
            LIMIT 5";

  # if no errors...
  if( $result = $link->query($query) )
  {
?>
<table border="1" width="40%" cellpadding="4" cellspacing ="1">
  <tbody>
  <tr>
    <th width="10%">City</th>
    <th>Population</th>
  </tr>
<?
    # then display the results...
    while($row = $result->fetch_object())
      echo "<tr>\n  <td align=\"center\">$row->Name</td>
            <td>$row->Population</td>\n</tr>\n";
?>
  </tbody
</table>
<?
  # ...and verify the number of rows that were retrieved
  echo "<p>Affected rows: " . $link->affected_rows . "</p>\n";
  }
  else
    # otherwise, tell us what went wrong
    echo mysqli_error();

  # free the result set and the mysqli connection object
```

```
  $result->close();
  $link->close();
?>
</body>
</html>
```

We assume that the process running on the Web server can reach the IP address of the SQL node.

In a similar fashion, you can use the MySQL C API, Perl-DBI, Python-mysql, or MySQL AB's Connectors to perform the tasks of data definition and manipulation just as you would normally with MySQL.

- Also remember that all NDB tables must have a primary key. If no primary key is defined by the user when a table is created, the NDB Cluster storage engine will automatically generate a hidden one.

16.4.6. Safe Shutdown and Restart

To shut down the cluster simply enter the following in a shell on the machine hosting the MGM node:

```
shell> ndb_mgm -e shutdown
```

This will cause the **ndb_mgm**, **ndb_mgm**, and any **ndbd** processes to terminate gracefully. Any SQL nodes can be terminated using **mysqladmin shutdown** and other means.

To restart the cluster, simply run these commands:

- On the management host (192.168.0.10 in our setup):

  ```
  shell> ndb_mgmd
  ```

- On each of the storage node hosts (192.168.0.30 and 192.168.0.40):

  ```
  shell> ndbd
  ```

  Remember **not** to invoke this command with the `--initial` option when restarting an NDBD node normally.

- And on the SQL host (192.168.0.20):

  ```
  shell> mysqld &
  ```

For information on making backups, see Section 16.7.4.2, “Using The Management Server to Create a Backup”.

To restore the cluster from backup requires the use of the **ndb_restore** command. This is covered in Section 16.7.4.3, “How to Restore a Cluster Backup”.

More information on configuring MySQL Cluster can be found in Section 16.5, “MySQL Cluster Configuration”.

16.5. MySQL Cluster Configuration

A MySQL server that is part of a MySQL Cluster differs in only one respect from a normal (non-clustered) MySQL server, in that it employs the NDB Cluster storage engine. This engine is also referred to simply as **NDB**, and the two forms of the name are synonomous.

In order to avoid unnecessary allocation of resources, the server is configured by default with the NDB storage engine disabled. To enable NDB, you will need to modify the server's `my.cnf` configuration file, or start it with the `--ndbcluster` option.

Since the MySQL server is a part of the cluster, it will also need to know how to access an MGM node in order to obtain the cluster configuration data. The default behavior is to look for the MGM node on `localhost`. However, should you need to specify its location elsewhere, this is can be done in `my.cnf` or on the MySQL server command line. Before the NDB storage engine can be used, at least one MGM node must be operational, as well as any desired data nodes.

16.5.1. Building from Source Code

NDB, the Cluster storage engine, is available in binary distributions beginning with MySQL-Max 4.1.3 for Linux, Mac OS X, and Solaris. It is not yet supported on Windows, but we intend to make it available for win32 and other platforms in the near future.

If you choose to build from a source tarball or the MySQL 4.1 BitKeeper tree, be sure to use the `--with-ndbcluster` option when running **configure**. You can also use the **BUILD/compile-pentium-max** build script. Note that this script includes OpenSSL, so you must have or obtain OpenSSL to build successfully; otherwise you will need to modify **compile-pentium-max** to exclude this requirement. Of course, you can also just follow the standard instructions for compiling your own binaries, then perform the usual tests and installation procedure. See Section 2.8.3, “Installing from the Development Source Tree”.

16.5.2. Installing the Software

In the next few sections, we assume that you are already familiar with installing MySQL, and here we cover only the differences between configuring MySQL Cluster and configuring MySQL without clustering. (See Chapter 2, *Installing MySQL* if you require more information about the latter.)

You will find Cluster configuration easiest if you have already have all management and storage nodes running first; this is likely to be the most time-consuming part of the configuration. Editing the `my.cnf` file is fairly straightforward, and this section will cover only any differences from configuring MySQL without clustering.

16.5.3. Quick Test Setup of MySQL Cluster

In order to familiarise you with the basics, we will describe the simplest possible configuration for a functional MySQL Cluster. After this, you should be able to design your desired setup from the information provided in the other relevant sections of this chapter.

First, you need to create a configuration directory such as `/var/lib/mysql-cluster`, by executing the following command as the system root user:

```
shell> mkdir /var/lib/mysql-cluster
```

In this directory, create a file named `config.ini` with the following information, substituting appropriate values for `HostName` and `DataDir` as necessary for your system.

```
# file "config.ini" - showing minimal setup consisting of 1 storage node,
# 1 management server, and 3 MySQL servers.
# The empty default sections are not required, and are shown only for
# the sake of completeness.
# Storage nodes must to provide a hostname but MySQL Servers are not required
# to do so.
# If you don't know the hostname for your machine, use localhost.
# The DataDir parameter also has a default value, but it is recommended to
# set it explicitly.
# Note: DB, API, and MGM are aliases for NDBD, MYSQLD, and NDB_MGMD
# respectively. DB and API are deprecated and should not be used in new
# installations.
[NDBD DEFAULT]
NoOfReplicas= 1

[MYSQLD DEFAULT]
[NDB_MGMD DEFAULT]
[TCP DEFAULT]

[NDB_MGMD]
HostName= myhost.example.com

[NDBD]
HostName= myhost.example.com
DataDir= /var/lib/mysql-cluster

[MYSQLD]
[MYSQLD]
[MYSQLD]
```

You can now start the management server as follows:

```
shell> cd /var/lib/mysql-cluster
shell> ndb_mgmd
```

Then start a single DB node by running **ndbd**. When starting **ndbd** for a given DB node for the very first time, you should use the `--initial` option as shown here:

```
shell> ndbd --initial
```

For subsequent **ndbd** starts, you will generally **not** want to use this option:

```
shell> ndbd
```

This is because the `--initial` option will delete all existing data and log files (as well as all table metadata) for this storage node and create new ones. One exception to this rule is when restarting the cluster and restoring from backup after adding new storage nodes.

By default, **ndbd** will look for the management server at `localhost` on port 1186. (Prior to MySQL 4.1.8, the default port was 2200.)

Note: If you have installed MySQL from a binary tarball, you will need to specify the path of the **ndb_mgmd** and **ndbd** servers explicitly. (Normally, these will be found in `/usr/local/mysql/bin`.)

Finally, go to the MySQL data directory (usually `/var/lib/mysql` or `/usr/local/mysql/data`), and make sure that the `my.cnf` file contains the option necessary to enable the NDB storage engine:

```
[mysqld]
ndbcluster
```

You can now start the MySQL server as usual:

```
shell> mysqld_safe --user=mysql &
```

Wait a moment to make sure the MySQL server is running properly. If you see the notice `mysql ended`, check the server's `.err` file to find out what went wrong.

If all has gone well so far, you now can start using the cluster:

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 4.1.7

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SHOW ENGINES;
+------------+---------+-------------------------------------------------------
----+
| Engine     | Support | Comment
   |
+------------+---------+-------------------------------------------------------
----+
...
| NDBCLUSTER | DEFAULT | Clustered, fault-tolerant, memory-based tables
   |
| NDB        | YES     | Alias for NDBCLUSTER
   |
...

mysql> USE test;
Database changed

mysql> CREATE TABLE ctest (i INT) ENGINE=NDBCLUSTER;
Query OK, 0 rows affected (0.09 sec)

mysql> SHOW CREATE TABLE ctest \G
*************************** 1. row ***************************
       Table: ctest
Create Table: CREATE TABLE `ctest` (
  `i` int(11) default NULL
) ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

To check that your nodes were set up properly, start the management client as shown:

```
shell> ndb_mgm
```

You can then use the **SHOW** command from within the management client in order to obtain a report on the cluster's status:

```
NDB> SHOW
Cluster Configuration
---------------------
[ndbd(NDB)]     1 node(s)
id=2    @127.0.0.1  (Version: 3.5.3, Nodegroup: 0, Master)
```

```
[ndb_mgmd(MGM)] 1 node(s)
id=1    @127.0.0.1  (Version: 3.5.3)

[mysqld(API)]   3 node(s)
id=3    @127.0.0.1  (Version: 3.5.3)
id=4 (not connected, accepting connect from any host)
id=5 (not connected, accepting connect from any host)
```

At this point, you have successfully set up a working MySQL Cluster. You can now store data in the cluster by using any table created with `ENGINE=NDBCLUSTER` or its alias `ENGINE=NDB`.

16.5.4. Configuration File

Configuring MySQL Cluster requires working with two files:

- `my.cnf`: Specifies options for all MySQL Cluster executables. This file, with which you should be familiar with from previous work with MySQL, must be accessible by each executable running in the cluster.
- `config.ini`: This file is read only by the MySQL Cluster management server, which then distributes the information contained in this file to all processes participating in the cluster. `config.ini` contains a description of each node involved in the cluster. This includes configuration parameters for storage nodes and configuration parameters for connections between all nodes in the cluster.

We are continuously making improvements in Cluster configuration and attempting to simplify this process. While we strive to maintain backwards compatibility, there may be times when introduce an incompatible change. In such cases we will try to let Cluster users know in advance if a change is not backwards compatible. If you find such a change which we have not documented, please use our Bugs Database [http://bugs.mysql.com/] to report it.

16.5.4.1. Example Configuration for a MySQL Cluster

In order to support MySQL Cluster, you will need to update `my.cnf` as shown in the example below.

From version 4.1.8 some simplifications in `my.cnf` were made, including new sections for `ndb-cluster` executables. However, these should not be confused with those occurring in `config.ini` files. As always, you may specify these parameters when invoking those executables from the command line.

```
# my.cnf
# example additions to my.cnf for MySQL Cluster
# (valid from 4.1.8)

# enable ndbcluster storage engine, and provide connectstring for
# management server host (default port is 1186)
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com


# provide connectstring for management server host (default port: 1186)
[ndbd]
connect-string=ndb_mgmd.mysql.com

# provide connectstring for management server host (default port: 1186)
[ndb_mgm]
connect-string=ndb_mgmd.mysql.com
```

```
# provide location of cluster configuration file
[ndb_mgmd]
config-file=/etc/config.ini
```

(For more information on connectstrings, see Section 16.5.4.2, “The MySQL Cluster `connectstring`”.)

```
# my.cnf
# example additions to my.cnf for MySQL Cluster
# (will work on all versions)

# enable ndbcluster storage engine, and provide connectstring for management
# server host to the default port 1186
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

Also starting with MySQL 4.1.8, the `my.cnf` file supports a separate `[mysql_cluster]` section for settings to be read by and affecting all executables in the cluster:

```
# cluster-specific settings
[mysql_cluster]
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

Currently the configuration file is in INI format, and is named `config.ini` by default. It is read by **ndb_mgmd** at startup and it can be placed anywhere. Its location and name are specified by using `--config-file=[<path>]<filename>` on the command line with **ndb_mgmd**. If the configuration file is not specified, **ndb_mgmd** will by default try to read a `config.ini` located in the current working directory.

Default values are defined for most parameters, and can also be specified in `config.ini`. To create a default value section, simply add the word `DEFAULT` to the section name. For example, storage nodes are configured using `[NDBD]` sections. If all storage nodes use the same data memory size, and this is not the same as the default size, create an `[NDBD DEFAULT]` section containing a `DataMemory` line to specify the default data memory size for all data nodes.

The INI format consists of sections preceded by section headings (surrounded by square brackets), followed by the appropriate parameter names and values. One deviation from the standard format is that the parameter name and value can be separated by a colon (`':'`) as well as the equals sign (`'='`); another is that sections are not uniquely identified by name. Instead, unique entries (such as two different nodes of the same type) are identified by a unique ID.

At a minimum, the configuration file must define the computers and nodes involved in the cluster and on which computers these nodes are located. An example of a simple configuration file for a cluster consisting of one management server, two storage nodes and two MySQL servers is shown below:

```
# file "config.ini" - 2 data nodes and 2 SQL nodes
# This file is placed in the startup directory of ndb_mgmd (the management
# server)
# The first MySQL Server can be started from any host. The second can be started
# only on the host mysqld_5.mysql.com

[NDBD DEFAULT]
NoOfReplicas= 2
DataDir= /var/lib/mysql-cluster

[NDB_MGMD]
```

```
Hostname= ndb_mgmd.mysql.com
DataDir= /var/lib/mysql-cluster

[NDBD]
HostName= ndbd_2.mysql.com

[NDBD]
HostName= ndbd_3.mysql.com

[MYSQLD]
[MYSQLD]
HostName= mysqld_5.mysql.com
```

There are six different sections in this configuration file:

- `[COMPUTER]`: Defines the the cluster hosts.
- `[NDBD]`: Defines the cluster's storage nodes.
- `[MYSQLD]`: Defines the cluster's MySQL server nodes.
- `[MGM|NDB_MGMD]`: Defines the management server node in the cluster.
- `[TCP]`: Defines TCP/IP connections between nodes in the cluster, with TCP/IP being the default connection protocol.
- `[SHM]`: Defines shared-memory connections between nodes. Prior to MySQL 4.1.9, this type of connection was available only in binaries that were built using the `--with-ndb-shm` option. Beginning with MySQL 4.1.9-max, it is enabled by default, but should still be considered experimental.

Note that each node has its own section in the `config.ini`. For instance, since this cluster has two storage nodes, the configuration file contains two sections defining these nodes.

One can define `DEFAULT` values for each section. As of MySQL 4.1.5, all parameter names are case-insensitive.

16.5.4.2. The MySQL Cluster `connectstring`

With the exception of the MySQL Cluster management server (**ndb_mgmd**), each node making up a MySQL Cluster requires a **connectstring** which points to the management server's location. This is used in establishing a connection to the management server as well as in performing other tasks depending on the node's role in the cluster. The syntax for a connectstring is as follows:

```
<connectstring> :=
[<nodeid-specification>,]<host-specification>[,<host-specification>]
<nodeid-specification> := nodeid=<id>
<host-specification> := <host>[:<port>]
```

`<id>` is an integer larger than 1 identifying a node in config.ini `<port>` is an integer referring to a regular unix port `<host>` is a string which is a valid Internet host address

```
example 1 (long):    "nodeid=2,myhost1:1100,myhost2:1100,192.168.0.3:1200"
example 2 (short):   "myhost1"
```

All nodes will use `localhost:1186` as the default connectstring value if none is provided. If `<port>` is omitted from the connectstring, the default port is 1186. (**Note**: Prior to MySQL 4.1.8, the

default port was 2200.) This port should always be available on the network, since it has been assigned by IANA for this purpose (see http://www.iana.org/assignments/port-numbers for details).

By listing multiple `<host-specification>` values, it is possible to designate several redundant management servers. A cluster node will attempt to contact successive management servers on each host in the order specified, until a successful connection has been established.

There are a number of different ways to specify the connectstring:

- Each executable has its own command line option which enables specifying the management server at startup. (See the documentation for the respective executable.)
- Beginning with MySQL 4.1.8, it is also possible to set the connectstring for all nodes in the cluster at once by placing it in a `[mysql_cluster]` section in the management server's `my.cnf` file.
- For backwards compatibility, two other options are available, using the same syntax:
  1. Set the `NDB_CONNECTSTRING` environment variable to contain the connectstring.
  2. Write the connectstring for each executable into a text file named `Ndb.cfg` and place this file in the executable's startup directory.

The recommended method for specifying the connectstring is to set it on the command line or in the `my.cnf` file for each executable.

16.5.4.3. Defining the Computers Making up a MySQL Cluster

The `[COMPUTER]` section has no real significance other than serving as a way to avoid the need of defining host names for each node in the system. All parameters mentioned here are required.

- `[COMPUTER]Id`

  This is an integer value, used to refer to the host computer elsewhere in the configuration file.

- `[COMPUTER]HostName`

  This is the computer's hostname or IP address.

16.5.4.4. Defining the MySQL Cluster Management Server

The `[NDB_MGMD]` section (or its alias `[MGM]`) is used to configure the behavior of the management server. Either the `ExecuteOnComputer` or `HostName` parameter must be present. All other parameters in the following can be omitted and, if so, will assume their default values.

- `[NDB_MGMD]Id`

  Each node in the cluster has a unique identity, which is represented by an integer value in the range 1 - 63 inclusive. This ID is used by all internal cluster messages for addressing the node.

- `[NDB_MGMD]ExecuteOnComputer`

  This refers to one of the computers defined in the `[COMPUTER]` section.

- `[NDB_MGMD]PortNumber`

This is the port number on which the management server listens for configuration requests and management commands.

- `[NDB_MGMD]LogDestination`

  This parameter specifies where to send cluster logging information. There are three options in this regard: `CONSOLE`, `SYSLOG`, and `FILE`:

  - `CONSOLE` outputs the log to `stdout`:

    ```
    CONSOLE
    ```

  - `SYSLOG` sends the log to a `syslog` facility, possible values being one of `auth`, `authpriv`, `cron`, `daemon`, `ftp`, `kern`, `lpr`, `mail`, `news`, `syslog`, `user`, `uucp`, `local0`, `local1`, `local2`, `local3`, `local4`, `local5`, `local6`, or `local7`.

    Note: Not every facility is necessarily supported by every operating system.

    ```
    SYSLOG:facility=syslog
    ```

  - `FILE` pipes the cluster log output to a regular file on the same machine. The following values can be specified:

    - `filename`: The name of the logfile.
    - `maxsize`: The maximum size to which the file can grow before logging rolls over to a new file. When this occurs, the old logfile is renamed by appending `.x` to the filename, where `x` is the next number not yet used with this name.
    - `maxfiles`: The maximum number of logfiles.

    ```
    FILE:filename=cluster.log,maxsize=1000000,maxfiles=6
    ```

    It is possible to specify multiple log destinations as shown here, using a semicolon-delimited string:

    ```
    CONSOLE;SYSLOG:facility=local0;FILE:filename=/var/log/mgmd
    ```

    The default value for the `FILE` parameter is `FILE:filename=ndb_<id>_cluster.log,maxsize=1000000,maxfiles=6`, where `<id>` is the ID of the node.

- `[NDB_MGMD]ArbitrationRank`

  This parameter is used to define which nodes can act as arbitrators. Only MGM nodes and SQL nodes can be arbitrators. `ArbitrationRank` can take one of the following values:

  - `0`: The node will never be used as an arbitrator.
  - `1`: The node has high priority; that is, it will be preferred as an arbitrator over low-priority nodes.
  - `2`: Indicates a low-priority node which be used as an arbtrator only if a node with a higher priority is not available for that purpose.

Normally, the management server should be configured as an arbitrator by setting its `ArbitrationRank` to 1 (the default value) and that of all SQL nodes to 0.

- `[NDB_MGMD]ArbitrationDelay`

  An integer value which causes the management server's responses to arbitration requests to be delayed by that number of milliseconds. By default, this value is 0; it is normally not necessary to change it.

- `[NDB_MGMD]DataDir`

  This sets the directory where output files from the management server will be placed. These files include cluster log files, process output files, and the daemon's pid file. (For log files, this can be overridden by setting the `FILE` parameter for `[NDB_MGMD]LogDestination` as discussed previously in this section.)

16.5.4.5. Defining MySQL Cluster Storage Nodes

The `[NDBD]` section is used to configure the behavior of the storage nodes. There are many parameters which control buffer sizes, pool sizes, timeouts, and so forth. The only mandatory parameters are:

- Either `ExecuteOnComputer` or `HostName`.
- The parameter `NoOfReplicas`

These need to be defined in the `[NDBD DEFAULT]` section.

Most storage node parameters are set in the `[NDBD DEFAULT]` section. Only those parameters explicitly stated as being possible set locally local values are allowed to be changed in the `[NDBD]` section. `HostName`, `Id` and `ExecuteOnComputer` *must* be defined in the local `[NDBD]` section.

Identifying Storage Nodes

The `Id` value (that is, the the storage node identifier) can be allocated on the command line when the node is started or in the configuration file.

For each parameter it is possible to use `k`, `M`, or `G` as a suffix to indicate units of 1024, 1024*1024, or 1024*1024*1024. (For example, `100k` means 10 * 1024 = 102400.) Parameters and values are currently case-sensitive.

- `[NBDB]Id`

  This is the node ID used as the address of the node for all cluster internal messages. This is an integer between 1 and 63. Each node in the cluster has a unique identity.

- `[NDBD]ExecuteOnComputer`

  This refers to one of the computers (hosts) defined in the `COMPUTER` section.

- `[NDBD]HostName`

  Specifying this parameter has an effect similar to specifying `ExecuteOnComputer`. It defines the hostname of the computer the storage node on which is to reside. Either this parameter or `ExecuteOnComputer` is required.

- `[NDBD]ServerPort`

  Each node in the cluster will use one port as the port other nodes use to connect the transporters to each other. This port is used also for non-TCP transporters in the connection setup phase. The default port will be calculated to ensure that no nodes on the same computer receive the same port number.

- `[NDBD]NoOfReplicas`

  This global parameter can be set only in the `[NDBD DEFAULT]` section, and defines the number of replicas for each table stored in the cluster. This parameter also specifies the size of node groups. A node group is a set of nodes that all store the same information.

  Node groups are formed implicitly. The first node group is formed by the set of storage nodes with the lowest node IDs, the next node group by the set of the next lowest node identities, and so on. By way of example, assume that we have 4 storage nodes and that `NoOfReplicas` is set to 2. The four storage nodes have node IDs 2, 3, 4 and 5. Then the first node group is formed from nodes 2 and 3, and the second node group by nodes 4 and 5. It is important to configure the cluster in such a manner that nodes in the same node groups are not placed on the same computer, as in this situation a single hardware failure would cause the entire cluster to crash.

  If no node IDs are provided then the order of the storage nodes will be the determining factor for the node group. Whether or not explicit assignments are made, they can be viewed in the output of the management client's `SHOW` command.

  There is no default value for `NoOfReplicas`; the maximum possible value is 4.

- `[NDBD]DataDir`

  This parameter specifies the directory where trace files, log files, pid files and error logs are placed.

- `[NDBD]FileSystemPath`

  This parameter specifies the directory where all files created for metadata, REDO logs, UNDO logs and data files are placed. The default is the directory specified by `DataDir`. **Note**: This directory must exist before the **ndbd** process is initiated.

  The recommended directory hierarchy for MySQL Cluster includes `/var/lib/mysql-cluster`, under which a directory for the node's filesystem is created. This subdirectory contains the node ID. For example, if the node ID is 2, then this subdirectory is named `ndb_2_fs`.

- `[NDBD]BackupDataDir`

  It is also possible to specify the directory in which backups are placed. By default, this directory is *`FileSystemPath/`*`BACKUP`. (See above.)

Data Memory and Index Memory

`DataMemory` and `IndexMemory` are parameters specifying the size of memory segments used to store the actual records and their indexes. In setting values for these, it is important to understand how `DataMemory` and `IndexMemory` are used, as they usually need to be updated in order to reflect actual usage by the cluster:

- `[NDBD]DataMemory`

  This parameter defines the amount of space available to store database records. The entire amount

will be allocated in memory, so it is extremely important that the machine has sufficient physical memory to accomodate this value.

The memory allocated by `DataMemory` is used to store both the the actual records and indexes. Each record is currently of fixed size. (Even `VARCHAR` columns are stored as fixed-width columns.) There is a 16-byte overhead on each record; an additional amount for each record is incurred because it is stored in a 32KB page with 128 byte page overhead (see below). There is also a small amount wasted per page due to the fact that each record is stored in only one page. The maximum record size is currently 8052 bytes.

The memory space defined by `DataMemory` is also used to store ordered indexes, which use about 10 bytes per record. Each table row is represented in the ordered index.

The memory space allocated by `DataMemory` consists of 32KB pages, which are allocated to table fragments. Each table is normally partitioned into the same number of fragments as there are storage nodes in the cluster. Thus, for each node, there are the same number of fragments as are set in `NoOfReplicas`. Once a page has been allocated, it is currently not possible to return it to the pool of free pages, except by deleting the table. Performing a node recovery also will compress the partition because all records are inserted into empty partitions from other live nodes.

The `DataMemory` memory space also contains UNDO information: For each update, a copy of the unaltered record is allocated in the `DataMemory`. There is also a reference to each copy in the ordered table indexes. Unique hash indexes are updated only when the unique index columns are updated, in which case a new entry in the index table is inserted and the old entry is deleted upon commit. For this reason, it is also necessary to allocate enough memory to handle the largest transactions performed by applications using the cluster. In any case, performing a few large transactions holds no advantage over using many smaller ones, for the following reasons:

- Large transactions are not any faster than smaller ones
- Large transactions increase the number of operations that are lost and must be repeated in event of transaction failure
- Large transactions use more memory

The default value for `DataMemory` is 80MB; the minimum is 1MB. There is no maximum size, but in reality the maximum size has to be adapted so that the process does not start swapping when the limit is reached.

- `[NDBD]IndexMemory`

  This parameter controls the amount of storage used for hash indexes in MySQL Cluster. Hash indexes are always used for primary key indexes, unique indexes, and unique constraints. Note that when defining a primary key and a unique index, two indexes will be created, one of which is a hash index used for all tuple accesses as well as lock handling. It is also used to enforce unique constraints.

  The size of the hash index is 25 bytes per record, plus the size of the primary key. For primary keys larger than 32 bytes another 8 bytes is added.

  Consider a table defined by

```
CREATE TABLE example
(
  a INT NOT NULL,
  b INT NOT NULL,
```

```
  c INT NOT NULL,
  PRIMARY KEY(a),
  UNIQUE(b)
) ENGINE=NDBCLUSTER;
```

There are 12 bytes overhead (having no nullable columns saves 4 bytes of overhead) plus 12 bytes of data per record. In addition we have two ordered indexes on columns **a** and **b** consuming about 10 bytes each per record. There is a primary key hash index on the base table with roughly 29 bytes per record. The unique constraint is implemented by a separate table with **b** as primary key and **a** as a column. This table will consume another 29 bytes of index memory per record in the **example** table as well as 12 bytes of overhead, plus 8 bytes of record data.

Thus, for one million records, we need 58 MB for index memory to handle the hash indexes for the primary key and the unique constraint. We also need 64 MB for the records of the base table and the unique index table, plus the two ordered index tables.

You can see that hash indexes takes up a fair amount of memory space; however, they provide very fast access to the data in return. They are also used in MySQL Cluster to handle uniqueness constraints.

Currently the only partitioning algorithm is hashing and ordered indexes are local to each node. Thus ordered indexes cannot be used to handle uniqueness constraints in the general case.

An important point for both `IndexMemory` and `DataMemory` is that the total database size is the sum of all data memory and all index memory for each node group. Each node group is used to store replicated information, so if there are four nodes with 2 replicas, then there will be two node groups. Thus, the total data memory available is 2*`DataMemory` for each storage node.

It is highly recommended that `DataMemory` and `IndexMemory` be set to the same values for all nodes. Since data is distributed evenly over all nodes in the cluster the maximum amount of space available is no better than that of the smallest node in the cluster.

`DataMemory` and `IndexMemory` can be changed, but decreasing either of these can be risky; doing so can easily lead to a node or even an entire Cluster that is unable to restart due to there being insufficient memory space. Increasing these values should be quite okay, but it is recommended that such upgrades are performed in the same manner as a software upgrade, beginning with an update of the configuration file, then restarting the management server followed by restarting each storage node in turn.

Updates do not increase the amount of index memory used. Inserts take effect immediately; however, rows are not actually deleted until the transaction is committed.

The default value for `IndexMemory` is 18MB. The minimum is 1MB.

Transaction Parameters

The next three parameters which we discuss are important because they affect the number of parallel transactions and the sizes of transactions that can be handled by the system. `MaxNoOfConcurrentTransactions` sets the number of parallel transactions possible in a node; `MaxNoOfConcurrentOperations` sets the number of records that can be in update phase or locked simultaneously.

Both of these parameters (especially `MaxNoOfConcurrentOperations`) are likely targets for users setting specific values and not using the default value. The default value is set for systems using small transactions, in order to ensure that these do not use excessive memory.

- `[NDBD]MaxNoOfConcurrentTransactions`

  For each active transaction in the cluster there must be a record in one of the cluster nodes. The task of co-ordinating transactions is spread amongst the nodes: the total number of transaction records in the cluster is the number of transactions in any given node times the number of nodes in the cluster.

  Transaction records are allocated to individual MySQL servers. Normally there is at least one transaction record allocated per connection that using any table in the cluster. For this reason, one should ensure that there are more transaction records in the cluster than there are concurrent connections to all MySQL servers in the cluster.

  This parameter must be set to the same value for all cluster nodes.

  Changing this parameter is never safe and can cause a cluster to crash. When a node crashes one of the node (actually the oldest surviving node) will build up the transaction state of all transactions ongoing in the crashed node at the time of the crash. It is thus important that this node has as many transaction records as the failed node.

  The default value for this parameter is 4096.

- `[NDBD]MaxNoOfConcurrentOperations`

  It is a good idea to adjust the value of this parameter according to the size and number of transactions. When performing transactions of only a few operations each and not involving a great many records, there is no need to set this parameter very high. When performing large transactions involving many records need to set this parameter higher.

  Records are kept for each transaction updating cluster data, both in the transaction co-ordinator and in the nodes where the actual updates are performed. These records contain state information needed in order to find UNDO records for rollback, lock queues, and other purposes.

  This parameter should be set to the number of records to be updated simultaneously in transactions, divided by the number of Cluster storage nodes. For example, in a cluster which has 4 storage nodes and which is expected to handle 1,000,000 concurrent updates using transactions, you should set this value to 1000000 / 4 = 250000.

  Read queries which set locks also cause operation records to be created. Some extra space is allocated in the individual nodes to accomodate cases where the distribution is not perfect over the nodes.

  When queries make use of the unique hash index, there are actually two operation records used per record in the transaction. The first record represents the read in the index table and the second handles the operation on the base table.

  The default value for this parameter is 32768.

  This parameter actually handles two values that can be configured separately. The first of these specifies how many operation records are to be placed with the transaction co-ordinator. The second part specifies how many operation records are to be local to the database.

  A very large transaction performed on a 8-node cluster requires as many operation records in the transaction co-ordinator as there are reads, updates, and deletes involved in the transaction. However, the operation records of the are spread over all 8 nodes. Thus, if it is necessary to configure the system for one very large transaction, then it is a good idea to configure the two parts separately. `MaxNoOfConcurrentOperations` will always be used to calculate the number of operation records in the transaction co-ordinator portion of the node.

  It is also important to have an idea of the memory requirements for operation records. In MySQL

4.1, these consume about 1KB per record. This figure is expected to be reduced in MySQL 5.x.

- `[NDBD]MaxNoOfLocalOperations`

  By default, this parameter is calculated as 1.1 * `MaxNoOfConcurrentOperations` which fits systems with many simultaneous transactions, none of them being very large. If there is a need to handle one very large transaction at a time and there are many nodes, then it is a good idea to override the default value by explicitly specifying this parameter.

Transaction Temporary Storage

The next set of parameters is used to determine temporary storage when executing a query which is part of a Cluster transaction. All records are released when the query is completed and the cluster is waiting for the commit or rollback.

The default values for these parameters are adequate for most situations. However, users with a need to support transactions involving large numbers of rows or operations may need to increase these to enable better parallelism in the system, while users whose applications require relatively small transactions can decrease the values in order to save memory.

- `[NDBD]MaxNoOfConcurrentIndexOperations`

  For queries using a unique hash index another, temporary set of operation records is used during a query's execution phase. This parameter sets the size of that pool of records. Thus this record is only allocated while executing a part of a query, as soon as this part has been executed the record is released. The state needed to handle aborts and commits is handled by the normal operation records where the pool size is set by the parameter `MaxNoOfConcurrentOperations`.

  The default value of this parameter is 8192. Only in rare cases of extremely high parallelism using unique hash indexes should it be necessary to increase this value. Using a smaller value is possible and can save memory if the DBA is certain that a high degree parallelism is not required for the cluster.

- `[NDBD]MaxNoOfFiredTriggers`

  The default value of `MaxNoOfFiredTriggers` is 4000, which is sufficient for most situations. In some cases it can even be decreased if the DBA feels certain the need for parallelism in the cluster is not high.

  A record is created when an operation is performed that affects a unique hash index. Inserting or deleting a record in a table with unique hash indexes or updating a column that is part of a unique hash index fires an insert or a delete in the index table. The resulting record is used to represent this index table operation while waiting for the original operation that fired it to complete. This operation is short lived but can still require a large number of records in its pool for situations with many parallel write operations on a base table containing a set of unique hash indexes.

- `[NDBD]TransactionBufferMemory`

  The memory affected by this parameter is used for tracking operations fired when updating index tables. This memory is used to store the key and column information for these operations. It is only very rarely that this parameter needs to be altered from the default.

  Normal read and write operations use a similar buffer, whose usage is even more short-lived. The compile-time parameter `ZATTRBUF_FILESIZE` (found in `ndb/src/kernel/blocks/Dbtc/Dbtc.hpp`) set to 4000*128 bytes (500KB). A similar buffer for key info, `ZDATABUF_FILESIZE` (also in `Dbtc.hpp`) contains 4000*16 = 62.5KB of buffer

space. `Dbtc` is the module which handles transaction co-ordination.

Scans and Buffering

There are additional parameters in the `Dblqh` module (in `ndb/src/kernel/blocks/Dblqh/Dblqh.hpp`) which affect reads and updates. These include `ZATTRINBUF_FILESIZE`, set by default to 10000*128 bytes (1250KB) and `ZDATABUF_FILE_SIZE`, set by default to 10000*16 bytes (roughly 156KB) of buffer space. To date, there have been neither any reports from users nor any results from our own extensive tests suggesting that either of these compile-time limits should be increased.

The default value for `TransactionBufferMemory` is 1MB.

- `[NDBD]MaxNoOfConcurrentScans`

  This parameter is used to control the number of parallel scans that can be performed in the cluster. Each transaction co-ordinator can handle the number of parallel scans defined for this parameter. Each scan query is performed by scanning all partitions in parallel. Each partition scan uses a scan record in the node where the partition is located, the number of records being the value of this parameter times the number of nodes. The cluster should be able to sustain `MaxNoOfConcurrentScans` scans concurrently from all nodes in the cluster.

  Scans are actually performed in two cases. The first of these cases occurs when no hash or ordered indexes exists to handle the query, in which case the query is executed by performing a full table scan. The second case is encountered when there is no hash index to support the query but there is an ordered index. Using the ordered index means executing a parallel range scan. Since the order is kept on the local partitions only, it is necessary to perform the index scan on all partitions.

  The default value of `MaxNoOfConcurrentScans` is 256. The maximum value is 500.

  This parameter specifies the number of scans possible in the transaction co-ordinator. If the number of local scan records is not provided, it is calculated as the product of `MaxNoOfConcurrentScans` and the number of storage nodes in the system.

- `[NDBD]MaxNoOfLocalScans`

  Specifies the number of local scan records if many scans are not fully parallelized.

- `[NDBD]BatchSizePerLocalScan`

  This parameter is used to calculate the number of lock records which needs to be there to handle many concurrent scan operations.

  The default value is 64; this value has a strong connection to the `ScanBatchSize` defined in the SQL nodes.

- `[NDBD]LongMessageBuffer`

  This is an internal buffer used for passing messages within individual nodes and between nodes. Whilke it is highly unlikely that this would need to be changed, it is configurable. By default it is set to 1MB.

Logging and Checkpointing

- `[NDBD]NoOfFragmentLogFiles`

This parameter sets the size of the node's REDO log files. REDO log files are organized in a ring. It is extremely important that the first and last log files (sometimes referred to as the "head" and "tail" log files, respectively) do not meet; when these approach one another too closely, the node will begin aborting all transactions encompassing updates due to a lack of room for new log records.

A REDO log record is not removed until three local checkpoints have been completed since that log record was inserted. Checkpointing frequency is determined by its own set of configuration parameters discussed elsewhere in this chapter.

The default parameter value is 8, which means 8 sets of 4 16MB files for a total of 512MB. In other words, REDO log space must be allocated in blocks of 64MB. In scenarios requiring a great many updates, the value for `NoOfFragmentLogFiles` may need to be set as high as 300 or even higher in order to provide sufficient space for REDO logs.

If the checkpointing is slow and there are so many writes to the database that the log files are full and the log tail cannot be cut without jeapo rdising recovery, then all updating transactions will be aborted with internal error code 410, or `Out of log file space temporarily`. This condition will prevail until a checkpoint has completed and the log tail can be moved forward.

- `[NDBD]MaxNoOfSavedMessages`

  This parameter sets the maximum number of trace files that will be kept before overwriting old ones. Trace files are generated when, for whatever reason, the node crashes.

  The default is 25 trace files.

Metadata Objects

The next set of parameters defines pool sizes for metadata objects. It is necessary to define the maximum number of attributes, tables, indexes, and trigger objects used by indexes, events, and replication between clusters.

- `[NDBD]MaxNoOfAttributes`

  Defines the number of attributes that can be defined in the cluster.

  The default value for this parameter is 1000, with the minimum possible value being 32. There is no maximum. Each attribute consumes around 200 bytes of storage per node due to the fact that all metadata is fully replicated on the servers.

- `[NDBD]MaxNoOfTables`

  A table object is allocated for each table, unique hash index, and ordered index. This parameter sets the maximum number of table objects for the cluster as a whole.

  For each attribute that has a `BLOB` data type an extra table is used to store most of the `BLOB` data. These tables also must be taken into account when defining the total number of tables.

  The default value of this parameter is 128. The minimum is 8 and the maximum is 1600. Each table

object consumes approximately 20KB per node.

- `[NDBD]MaxNoOfOrderedIndexes`

  For each ordered index in the cluster, an object is allocated describing what is being indexed and its storage segments. By default each index so defined also defines an ordered index. Each unique index and primary key has both an ordered index and a hash index.

  The default value of this parameter is 128. Each object consumes approximately 10KB of data per node.

- `[NDBD]MaxNoOfUniqueHashIndexes`

  For each unique index that is not a primary key, a special table is allocated that maps the unique key to the primary key of the indexed table. By default there an ordered index is also defined for each unique index. To prevent this, you must use the `USING HASH` option when defining the unique index.

  The default value is 64. Each index consumes approximately 15KB per node.

- `[NDBD]MaxNoOfTriggers`

  Internal update, insert, and delete triggers are allocated for each unique hash index. (This means that 3 triggers are created for each unique hash index.) However, an **ordered** index requires only a single trigger object. Backups also use three trigger objects for each normal table in the cluster.

  Note: When replication between clusters is supported, this will also make use of internal triggers.

  This parameter sets the maximum number of trigger objects in the cluster.

  The default value for this parameter is 768.

- `[NDBD]MaxNoOfIndexes`

  This parameter was deprecated in MySQL 4.1.5. In MySQL 4.1.6 and newer versions, you should use `MaxNoOfOrderedIndexes` and `MaxNoOfUniqueHashIndexes` instead.

  This parameter is used only by unique hash indexes. There needs to be one record in this pool for each unique hash index defined in the cluster.

  The default value of this parameter is 128.

Boolean Parameters

The behavior of storage nodes is also affected by a set of parameters taking boolean values. These parameters can be specified as **TRUE** by setting them equal to **1** or **Y**, and as **FALSE** by setting them equal to **0** or **N**.

- `[NDBD]LockPagesInMainMemory`

  For a number of operating systems, including Solaris and Linux, it is possible to lock a process into memory and avoid any swapping to disk. This can be used to help guarantee the cluster's real-time characteristics.

  By default, this feature is disabled.

- `[NDBD]StopOnError`

This parameter specifies whether an NDBD process is to exit or to perform an automatic restart when an error condition is encountered.

This feature is enabled by default.

- `[NDBD]Diskless`

  It is possible to specify Cluster tables as **diskless**, meaning that tables are not checkpointed to disk and that no logging occurs. Such tables exist only in main memory. A consequence of using diskless tables is that the tables will still be defined after a crash, but records in those tables will not be preserved.

  This feature makes the entire cluster operate in diskless mode. We plan to add the capability to employ diskless mode on a per-table basis in a future (5.0 or 5.1) release of MySQL Cluster.

  Currently, when this feature is enabled, backups are performed but backup data is not actually stored. In a future release we hope to make diskless backup a separate, configurable parameter.

  This feature is enabled by setting `Diskless` to either **1** or **y**. `Diskless` is disabled by default.

- `[NDBD]RestartOnErrorInsert`

  This feature is accessible only when building the debug version where it is possible to insert errors in the execution of individual blocks of code as part of testing.

  By default, this feature is disabled.

Controlling Timeouts, Intervals, and Disk Paging

There are a number of parameters specifying timeouts and intervals between various actions in Cluster storage nodes. Most of the timeout values are specified in milliseconds. Any exceptions to this will be mentioned below.

- `[NDBD]TimeBetweenWatchDogCheck`

  To prevent the main thread from getting stuck in an endless loop at some point, a "watch dog" thread checks the main thread. This parameter specifies the number of milliseconds between checks. If the prrocess remains in the same state after three checks, it is terminated by the watch dog thread.

  This parameter can easily be changed for purposes of experimentation or to ad apt to local conditions. It can be specified on a per-node nodes although there seems to be little reason for doing so.

  The default timeout is 4000 milliseconds (4 seconds).

- `[NDBD]StartPartialTimeout`

  This parameter specifies the time that the cluster will wait for all storage nodes to come up before the cluster initialisation routine is invoked. This timeout is used to avoid a partial Cluster startup whenever possible.

  The default value is 30000 milliseconds (30 seconds). **0** means eternal time out; in other words, the cluster may start only if all nodes are available.

- `[NDBD]StartPartitionedTimeout`

  If the cluster is ready to start after waiting for `StartPartialTimeout` milliseconds but is still in a possibly partitioned state, the cluster waits until this timeout has also passed.

The default timeout is 60000 milliseconds (60 seconds).

- `[NDBD]StartFailureTimeout`

  If a storage node has not completed its startup sequence within the time specified by this parameter, then the node startup will fail. Setting this parameter to **0** means that no storage node timeout is applied.

  The default value is 60000 milliseconds (60 seconds). For storage nodes containing extremely large amounts of data, this parameter should be increased. For example, in the case of a storage node containing several gigabytes of data, it could very well take 10-15 minutes (that is, 600,000 to 1,000,000 milliseconds) to perform a node restart.

- `[NDBD]HeartbeatIntervalDbDb`

  One of the primary methods of discovering failed nodes is by the use of heartbeats. This parameter states how often heartbeat signals are sent and how often to expect to receive them. After missing three heartbeat intervals in a row, the node is declared dead. Thus the maximum time for discovering a failure through the heartbeat mechanism is four times the heartbeat interval.

  The default heartbeat interval is 1500 milliseconds (1.5 seconds). This parameter must not be changed drastically and should not vary widely between nodes. If one node uses 5000 milliseconds and the node watching it uses 1000 milliseconds then obviously the node will be declared dead very quickly. This parameter can be changed during an online software upgrade but only in small increments.

- `[NDBD]HeartbeatIntervalDbApi`

  Each storage node sends heartbeat signals to each MySQL server (SQL node) to ensure that it remains in contact. If a MySQL server fails to send a heartbeat in time it is declared "dead", in which case all ongoing transactions are completed and all resources released. The SQL node cannot reconnect until the completion of all activities started by the previous MySQL instance have been completed. The three-heartbeat criteria for this determination are the same as described for `HeartbeatIntervalDbDb`.

  The default interval is 1500 milliseconds (1.5 seconds). This interval can vary between indidivual storage nodes because each storage node watches the MySQL servers connected to it, independently of all other storage nodes.

- `[NDBD]TimeBetweenLocalCheckpoints`

  This parameter is an exception in that it doesn't specify a time to wait before starting a new local checkpoint; it is used to ensure that local checkpoints a re not performed in a cluster where not so many updates are taking place. In most clusters with high update rates, it is likely that a new local checkpoint is started immediately after the previous one has been completed.

  The size of all write operations executed since the start of the previous local checkpoints is added. This parameter is also exceptional in that it is specified as the logarithm of the number of 4-byte words, so the default value 20 means 4MB (4 times 2 ^ 21) of write operations, 21 would mean 8MB and so forth up until the maximum value 31 which means 8GB of write operations.

  All the write operations in the cluster are added together. Setting `TimeBetweenLocalCheckpoints` to 6 or less means that local checkpoints will be executed continuously without pause, independent of the cluster's workload.

- `[NDBD]TimeBetweenGlobalCheckpoints`

  When a transaction is committed, it is committed in main memory in all nodes on which the data is

mirrored. However, transaction log records are not flushed to disk as part of the commit. The reasoning behing this behavior is that having the transaction safely committed on at least two autonomous host machines should meet reasonable standards for durability.

It is also important to ensure that even the worst of cases -- a complete crash of the cluster -- is handled properly. To guarantee that this happens, all transactions taking place within a given interval are put into a global checkpoint, which can be thought of as a set of committed transactions that has been flushed to disk. In other words, as part of the commit process, a transaction is placed in a global checkpoint group; later, this group's log records are flushed to disk, and then the entire group of transactions is safely committed to disk on all computers in the cluster as well.

This parameter defines the interval between global checkpoints. The default is 2000 milliseconds.

- `[NDBD]TimeBetweenInactiveTransactionAbortCheck`

  Time-out handling is performed by checking a timer on each transaction once for every interval specified by this parameter. Thus, if this parameter is set to 1000 milliseconds, then every transaction will be checked for timing out once per second.

  The default value for this parameter is 1000 milliseconds (1 second).

- `[NDBD]TransactionInactiveTimeout`

  If the transaction is currently not performing any queries but is waiting for further user input, this parameter states the maximum time that the user can wait before the transaction is aborted.

  The default for this parameter is zero (no timeout). For a real-time database that needs to ensure that no transaction keeps locks for too long a time this parameter should be set to a much smaller value. The unit is milliseconds.

- `[NDBD]TransactionDeadlockDetectionTimeout`

  When a node executes a query involving a transaction, the node waits for the other nodes in the cluster to respond before continuing. A failure to respond can occur for any of the following reasons:

  1. The node is "dead"
  2. The operation has entered a lock queue
  3. The node requested to perform the action could be heavily overloaded.

  This timeout parameter states how long the transaction coordinator will wait for query execution by another node before aborting the transaction, and is important for both node failure handling and deadlock detection. Setting it too high can cause a undesirable behavior in situations involving deadlocks and node failure.

  The default timeout value is 1200 milliseconds (1.2 seconds).

- `[NDBD]NoOfDiskPagesToDiskAfterRestartTUP`

  When executing a local checkpoint the algorithm flushes all data pages to disk. Merely doing as quickly as possible without any moderation is likely to impose excessive loads on processors, networks, and disks. To control the write speed, this parameter specifies how many pages per 100 milliseconds are to be written. In this context, a "page" is defined as 8KB; thus, this parameter is specified in units of 80KB per second. Therefore, setting `NoOfDiskPagesToDiskAfterRestartTUP` to a value of 20 means writing 1.6MB of data pages to disk each second during a local checkpoint. This value includes the writing of UNDO log records for data pages; that is, this para-

meter handles the limitation of writes from data memory. UNDO log records for index pages are handled by the parameter `NoOfDiskPagesToDiskAfterRestartACC`. (See the entry for `IndexMemory` for information about index pages.)

In short, this parameter specifies how quickly local checkpoints will be executed, and operates in conjunction with `NoOfFragmentLogFiles`, `DataMemory`, and `IndexMemory`.

The default value is 40 (3.2MB of data pages per second).

- `[NDBD]NoOfDiskPagesToDiskAfterRestartACC`

  This parameter uses the same units as `NoOfDiskPagesToDiskAfterRestartTUP` and acts in a similar fashion, but limits the speed of writing index pages from index memory.

  The default value of this parameter is 20 index memory pages per second (1.6MB per second).

- `[NDBD]NoOfDiskPagesToDiskDuringRestartTUP`

  This parameter in a similar fashion as `NoOfDiskPagesToDiskAfterRestartTUP` and `NoOfDiskPagesToDiskAfterRestartACC`, only it does so with regard to local checkpoints executed in the node when a node is restarting. As part of all node restarts a local checkpoint is always performed. During a node restart it is possible to write to disk at a higher speed than at other times, because fewer activities are being performed in the node.

  This parameter covers pages written from data memory.

  The default value is 40 (3.2MB per second).

- `[NDBD]NoOfDiskPagesToDiskDuringRestartACC`

  Controls the number of index memory pages that can be written to disk during the loca checkpoint phase of a node restart.

  As with `NoOfDiskPagesToDiskAfterRestartTUP` and `NoOfDiskPagesToDiskAfterRestartACC`, values for this parameter are expressed in terms of 8KB pages written per 100 milliseconds (80KB/second).

  The default value is 20 (1.6MB per second).

- `[NDBD]ArbitrationTimeout`

  This parameter specifies the time that the storage node will wait for a response from the arbitrator to an arbitration message. If this is exceeded, then it is assumed that the network has split.

  The default value is 1000 milliseconds (1 second).

Buffering and Logging

Several configuration parameters corresponding to former compile-time parameters were introduced in MySQL 4.1.5. These enable the advanced user to have more control over the resources used by node processes and to adjust various buffer sizes at need.

These buffers are used as front-ends to the file system when writing log records to disk. If the node is running in diskless mode, then these parameters can be set to their minimum values without penalty due to the fact that disk writes are "faked" by the NDB storage engine's filesystem abstraction layer.

- `[NDBD]UndoIndexBuffer`

  This buffer is used during local checkpoints. The NDB storage engine uses a recovery scheme based on checkpoint consistency in conjunciton with an operational REDO log. In order to produce a consistent checkpoint without blocking the entire system for writes, UNDO logging is done while performing the local checkpoint. UNDO logging is activated on a single table fragment at a time. This optimization is possible because tables are stored entirely in main memory.

  The UNDO index buffer buffer is used for the updates on the primary key hash index. Inserts and deletes rearrange the hash index; the NDB storage engine writes UNDO log records that map all physical changes to an index page so that they can be undone at system restart. It also logs all active insert operations for each fragment at the start of a local checkpoint.

  Reads and updates set lock bits and update a header in the hash index entry. These changes are handled by the page-writing algorithm to insure that these operations need no UNDO logging.

  This buffer is 2MB by default. The minimum value is 1MB, and for most applications the minimum is sufficient. For applications doing extremely large and/or numerous inserts and deletes together with large transactions and large primary keys, it may be necessary to increase the size of this buffer. If this buffer is too small, the NDB storage engine issues internal error code 677 `Index UNDO buffers overloaded`.

- `[NDBD]UndoDataBuffer`

  The UNDO data buffer plays the same role as the UNDO index buffer, except that it is used with regard to data memory rather than index memory. This buffer is used during the local checkpoint phase of a fragment for inserts, deletes, and updates.

  Since UNDO log entries tend to grow larger as more operations are logged, this buffer is also larger than its index memory counterpart, with a default value of 16MB.

  For some applications this amount of memory may be unnecessarily large. In such cases it is possible to decrease this size down to a minimum of 1MB.

  It is rarely necessary to increase the size of this buffer. If there is such a need, then it is a good idea to check if the disks can actually handle the load caused by database update activity. A lack of sufficient disk space cannot be overcome by increasing the size of this buffer.

  If this buffer is too small and gets congested, the NDB storage engine issues internal error code 891 `Data UNDO buffers overloaded`.

- `[NDBD]RedoBuffer`

  All update activities also need to be logged. This log makes it possible to replay these updates whenever the system is restarted. The NDB recovery algorithm uses a "fuzzy" checkpoint of the data together with the UNDO log, then applies the REDO log to play back all changes up to the restoration point.

  This buffer is 8MB by default. The minimum value is 1MB.

  If this buffer is too small, the NDB storage engine issues error code 1221 `REDO log buffers overloaded`.

In managing the cluster, it is very important to be able to control the number of log messages sent for various event types to `stdout`. There are 16 possible event levels (numbered 0 through 15). Setting event reporting for a given event category to level 15 means all event reports in that category are sent to `stdout`; setting it to 0 means that there will be no event reports made in that category.

By default, only the startup message is sent to `stdout`, with the reaimaining event reporting level defaults being set to 0. The reason for this is that these messages are also sent to the management server's cluster log.

An analogous set of levels can be set for the management client to determine which event levels to record in the cluster log.

- `[NDBD]LogLevelStartup`

  Reporting level for events generated during startup of the process.

  The default level is 1.

- `[NDBD]LogLevelShutdown`

  Reporting level for events generated as part of graceful shutdown of a node.

  The default level is 0.

- `[NDBD]LogLevelStatistic`

  Reporting level for statistical events such as number of primary key reads, number of updates, number of inserts, information relating to buffer usage, and so on.

  The default level is 0.

- `[NDBD]LogLevelCheckpoint`

  Reporting level for events generated by local and global checkpoints.

  The default level is 0.

- `[NDBD]LogLevelNodeRestart`

  Reporting level for events generated during node restart.

  The default level is 0.

- `[NDBD]LogLevelConnection`

  Reporting level for events generated by connections between cluster nodes.

  The default level is 0.

- `[NDBD]LogLevelError`

  Reporting level for events generated by errors and warnings by the cluster as a whole. These errors do not cause any node failure but are still considered worth reporting.

  The default level is 0.

- `[NDBD]LogLevelInfo`

  Reporting level for events generated for information about the general state of the cluster.

  The default level is 0.

Backup Parameters

The parameters discussed in this section define memory buffers set aside for execution of online backups.

- `[NDBD]BackupDataBufferSize`

  In creating a backup, there are two buffers used for sending data to the disk. The backup data buffer buffer is used to fill in data recorded by scanning a node's tables. Once this buffer has been filled to the level specified as `BackupWriteSize` (see below), the pages are sent to disk. While flushing data to disk, the backup process can continue filling this buffer until it runs out of space. When this happens, the backup process pauses the scan and waits until some disk writes have completed and have thus freed up memory so that scanning may continue.

  The default value for this parameter is 2MB.

- `[NDBD]BackupLogBufferSize`

  The backup log buffer fulfils a role similar to that played by the backup data buffer, except that it used for generating a log of all table writes made during execution of the backup. The same principles apply for writing these pages as with the backup data buffer, except that when there is no more space in the backup log buffer, the backup fails. For that reason, the size of the back up logbuffer must be big enough to handle the load caused by write activities while the backup is being made.

  The default value for this parameter should be sufficient for most applications. In fact, it is more likely for a backup failure to be caused by insufficient disk write speed than it is for the backup log buffer to become full. If the disk subsystem is not configured for the write load caused by applications, the cluster will be likely to be able to perform the desired operations.

  It is preferable to configure cluster nodes in such a manner that the processor becomes the bottleneck rather than the disks or the network connections.

  The default value is 2MB.

- `[NDBD]BackupMemory`

  This parameter is simply the sum of `BackupDataBufferSize` and `BackupLogBufferSize`.

  The default value is 2MB + 2MB = 4MB.

- `[NDBD]BackupWriteSize`

  This parameter specifies the size of messages written to disk by the backup log and backup data buffers.

  The default value is 32KB.

16.5.4.6. Defining the SQL Nodes in a MySQL Cluster

The `[MYSQLD]` sections in the `config.ini` file define the behavior of the MySQL servers (SQL nodes) used to access cluster data. None of the parameters shown is required. If no computer or host name is provided, then any host can use this SQL node.

- `[MYSQLD]Id`

  This value is used as the address of the node by all cluster internal messages, and must be an integer between 1 and 63. Each cluster node must have a unique identity.

- `[MYSQLD]ExecuteOnComputer`

  This refers to one of the computers (hosts) defined in a `[COMPUTER]` section of the configuration file.

- `[MYSQLD]ArbitrationRank`

  This parameter is used to define which nodes can act as arbitrators. Both MGM nodes and SQL nodes can be arbitrators. A value of 0 means the given node will never be used as an arbitrator, a value of 1 gives the node high priority as an arbiortator, and a value of 2 gives it low priority. A normal configuration uses the management server as arbitrator, setting its ArbitrationRank to 1 (the default) and those for all SQL nodes to 0.

- `[MYSQLD]ArbitrationDelay`

  Setting this parameter to any other value than 0 (the default) means that responses by the arbitrator to arbitration requests will be delayed by the stated number of milliseconds. It is usually not necessary to change this value.

- `[MYSQLD]BatchByteSize`

  For queries that are translated into full table scans or range scans on indexes, it is important for best performance to fetch records in properly sized batches. It is possible to set the proper size both in terms of number of records (`BatchSize`) and in terms of bytes (`BatchByteSize`). The actual batch size will be limited by both parameters.

  The spped at which queries are performed can vary by more than 40% depending upon how this parameter is set. In future releases, the MySQL Server will make educated guesses on how to set parameters relating to batch size, based on the query type.

  This parameter is measured in bytes and by default is equal to 32KB.

- `[MYSQLD]BatchSize`

  This parameter is measured in number of records and is by default set to 64. The maximum size is 992.

- `[MYSQLD]MaxScanBatchSize`

  The batch size is the size of each batch sent from each storage node. Most scans are performed in parallel in order to protect the MySQL Server from receiving too much data from many nodes in parallel; this parameter sets a limit to the total batch size over all nodes.

  The default value of this parameter is set to 256KB. Its maximum size is 16MB.

16.5.4.7. MySQL Cluster TCP/IP Connections

TCP/IP is the default transport mechanism for establishing connections in MySQL Cluster. It is actually not necessary to define connections because Cluster automatically set ups a connection between each of the storage nodes, between each storage node and all MySQL server nodes, and between each storage node and the management server.

It is only necessary to define a connection in order to override the default connection parameters. In that case it is necessary to define at least *`NodeId1`*, *`NodeId2`*, and the parameters to change.

It is also possible to change the default values fro these parameters by setting them in the `[TCP DEFAULT]` section.

- `[TCP]NodeId1` , `[TCP]NodeId2`

  To identify a connection between two nodes it is necessary to provide the node identities for both of them in the `[TCP]` seciton of the configuration file.

- `[TCP]SendBufferMemory`

  TCP transporters use a buffer to store all messages before performing the send call to the operating system. When this buffer reaches 64KB its contents are sent; these is also sent when a round of messages have beenexecuted. To handle temporary overload situations it is also possible to define a bigger send buffer. The default size of the send buffer is 256KB.

- `[TCP]SendSignalId`

  In order to be able to retrace a distributed message diagram it is necessary to identify each message. When this parameter is set to **Y**, message IDs are transported over the network. This feature is disabled by default.

- `[TCP]Checksum`

  This parameter is also a boolean (Y/N or 1/0) parameter which is disabled by default. When it is enabled, checksums for all messages are calculated before they placed in the send buffer. This feature insures that messages are not corrupted while waiting in the send buffer, or by the transport mechanism.

- `[TCP]PortNumber`

  (**DEPRECATED**.) This is the port number to be used to listen for connections from other nodes. This port number is normally specified in the configuration file's `[TCP DEFAULT]` section.

  This parameter should no longer be used. Use the `ServerPort` instead.

- `[TCP]ReceiveBufferMemory`

  Specifies the size of the buffer used when receiving data from the TCP/IP socket. There is seldom any need to change this parameter from its default value of 64KB, except possibly in order to save memory.

16.5.4.8. MySQL Cluster Shared-Memory Connections

Beginning with MySQL 4.1.9-max, MySQL Cluster will attempt to use the shared memory transporter and configure it automatically where possible, chiefly where more than one node runs concurrently on the same cluster host. (In previous versions of MySQL Cluster, shared memory segments were supported only when the `-max` binary was built using `--with-ndb-shm`.) When explicitly defining shared memory as the connection method it is necessary to define at least `NodeId1`, `NodeId2` and `ShmKey`. All other parameters have default values that will work out fine in most cases.

Note: SHM support should still be considered experimental in MySQL 4.1.

- `[SHM]NodeId1` , `[SHM]NodeId2`

  To identify a connection between two nodes it is necessary to provide node identifiers for each of them, as `NodeId1` and `NodeId2`.

- `[SHM]ShmKey`

  When setting up shared memory segments, a node ID is used to identify uniquely the shared memory

segment to use for the communication. This is expressed as an integer which does not have a default value.

- `[SHM]ShmSize`

  Each SHM connection has a shared memory segment where messages between nodes are placed by the sender and read by the reader. The size of this segment is defined by `ShmSize`. The default value is 1MB.

- `[SHM]SendSignalId`

  In order to retrace the path of a distributed message, it is necessary to provide each message with a unique identifier. Setting this parameter to **Y** causes these message IDs to be transported over the network as well. This feature is disabled by default.

- `[SHM]Checksum`

  This parameter is also a Y/N parameter which is disabled by default. When it is enabled, checksums for all messages are calculated before placed in the send buffer.

  This feature prevents messages from being corrupted while waiting in the send buffer. It also serves as a check against data being corrupted during transport.

16.5.4.9. MySQL Cluster SCI Transport Connections

Using SCI transporters to connect nodes in a MySQL Cluster is supported only when the MySQL-max binaries are built using `--with-ndb-sci=/your/path/to/SCI`. The *`path`* should point to a directory that contains at a minimum `lib` and `include` directories containing SISCI libraries and header files.

In addition, SCI requires specialised hardware.

It is strongly recommended to use SCI Transporters only for communication between **ndbd** processes. In addition, using SCI Transporters means that the **ndbd** processes never sleep. For this reason, SCI Transporters should be used only on machines having at least 2 CPUs dedicated for use by **ndbd** processes. There should be at least 1 CPU per ndbd process, with at least 1 CPU left in reserve to handle operating system activities.

- `[SCI]NodeId1`, `[SCI]NodeId2`

  To identify a connection between two nodes it is necessary to provide node identifiers for each of them, as `NodeId1` and `NodeId2`.

- `[SCI]Host1SciId0`

  This identifies the SCI node ID on the first Cluster node (identified by `NodeId1`).

- `[SCI]Host1SciId1`

  It is possible to set up SCI Transporters for failover between two SCI cards which then should use separate networks between the nodes. This identifies the node ID and the second SCI card to be used on the first node.

- `[SCI]Host2SciId0`

  This identifies the SCI node ID on the second Cluster node (identified by `NodeId2`).

- `[SCI]Host2SciId1`

  When using two SCI cards to provide failover, this parameter identifies the second SCI card to be used on the second node.

- `[SCI]SharedBufferSize`

  Each SCI transporter has a shared memory segment used for communication between the two nodes. Setting the size of this segment to the default value of 1 MB should be sufficient for most applications. Using a smaller value can le ad to problems when performing many parallel inserts; if the shared buffer is too small, this can also result in a crash of the **ndbd** process.

- `[SCI]SendLimit`

  A small buffer in front of the SCI media stores messages before transmitting them over the SCI network. By default this is set to 8kB. Our benchmarks show that performance is best at 64 kB but 16kB reaches within a few percent of this, and there was little if any advantage to increasing it beyond 8kB.

- `[SCI]SendSignalId`

  In order to trace a distributed message it is necessary to identify each message uniquely. When this parameter is set to **Y**, message IDs are transported over the network. This feature is disabled by default.

- `[SCI]Checksum`

  This parameter is also a boolean value, and is disabled by default. When `Checksum` is enabled, checksums are calculated for all messages before they are placed in the send buffer. This feature prevents messages from being corrupted while waiting in the send buffer. It also serves as a check against data being corrupted during transport.

16.6. Process Management in MySQL Cluster

Understanding how to manage MySQL Cluster requires a knowledge of four essential processes. In the next few sections of this chapter, we cover the roles played by these processes in a cluster, how to use them, and what startup options are available for each of them.

16.6.1. MySQL Server Process Usage for MySQL Cluster

mysqld is the traditional MySQL server process. To be used with MySQL Cluster, **mysqld** needs to be built with support for the NDB Cluster storage engine, as it is in the precompiled `-max` binaries available from mysql.com for MySQL versions 4.1.3 and newer.

If the **mysqld** binary has been built in such a manner, the NDB Cluster storage engine is still disabled by default. You can use either of two possible options for enabling the NDB Cluster storage engine:

Use `--ndbcluster` as a startup option when starting **mysqld**

- Insert a line containing `ndbcluster` in the `[mysqld]` section of your `my.cnf` file.

An easy way to verify that your server is running with the NDB Cluster storage engine enabled is to issue the command `SHOW ENGINES` in the MySQL Monitor (**mysql**). You should see the value `YES` in the row listing `NDBCLUSTER`; if you see `NO` in this row (or if there is no such row displayed in the out-

put), you are not running an NDB-enabled version of **mysqld**. If you see `DISABLED` in this row, then you need to enable it in either one of the two ways shown above.

In order to read cluster configuration data, the MySQL server requires 3 pieces of information:

- The MySQL server's own cluster node ID.
- The hostname or IP address for the management server (MGM node).
- The port on which it can connect to the management server.

Beginning with MySQL 4.1.5, node IDs can be dynamically allocated, in which case there is no need to specify it explicitly.

The **mysqld** parameter `ndb-connectstring` is used to specify the connectstring either on the command line when starting **mysqld** or in `my.cnf`. The connectstring contains the hostname/IP and port where the management server can be found.

In the following example, `ndb_mgmd.mysql.com` is the host where the management server resides, and the management server listens for cluster messages on port 1186:

```
shell> mysqld --ndb-connectstring=ndb_mgmd.mysql.com:1186
```

See Section 16.5.4.2, “The MySQL Cluster `connectstring`” for more info on connectstrings.

Given this information, the MySQL server will be a full participant in the cluster. (We sometimes refer to a **mysqld** process running in this manner as an SQL node.) It will be fully aware of all cluster storage nodes as well as their status, and will establish connections to all storage nodes. In this case, is able to use any storage node as a transaction coordinator and to access storage node data for reading and updating.

16.6.2. ndbd, the Storage Engine Node Process

ndbd is the process that is used to handle all the data in tables using the NDB Cluster storage engine. This is the process empowers a storage node to accomplish distributed transaction handling, node recovery, checkpointing to disk, online backup, and related tasks.

In a MySQL Cluster, a set of **ndbd** processes co-operate in handling data. These processes can execute on the same computer (host) or on different computers. The correspondences between storage nodes and Cluster hosts is completely configurable.

In MySQL versions prior to 4.1.5, each **ndbd** process should be started in a separate directory, the reason for this being that **ndbd** generated a set of log files in its starting directory. In MySQL 4.1.5, this behavior was changed such that the these files are placed in the directory specified by `DataDir` in the configuration file. Thus **ndbd** can be started from anywhere.

These log files are listed below. `<NodeID>` is the node's unique identifier. For example, `ndb_2_error.log` is the error log generated by the storage node whose node ID is **2**.

- `ndb_<NodeID>_error.log` (was `error.log` in version 4.1.3) is a file containing records of all crashes which the referenced **ndbd** process has encountered. Each record in this file contains a brief error string and a reference to a trace file for this crash. A typical entry in this file might appear as shown here:

  ```
  Date/Time: Saturday 31 January 2004 - 00:20:01
  Type of error: error
  ```

```
Message: Internal program error (failed ndbrequire)
Fault ID: 2341
Problem data: DbtupFixAlloc.cpp
Object of reference: DBTUP (Line: 173)
ProgramName: NDB Kernel
ProcessID: 14909
TraceFile: ndb_2_trace.log.2
***EOM***
```

- `ndb_<NodeID>_trace.log.<TraceID>` (was `NDB_TraceFile_<TraceID>.trace` in version 4.1.3) is a trace file describing exactly what happened just before the error occurred. This information is useful for anaylsis by the MySQL Cluster development team.

  It is possible to configure the number of these trace files that will be created before old files are overwritten. `<TraceID>` is a number which is incremented for each successive trace file.

- `ndb_<NodeID>_trace.log.next` (was `NextTraceFileNo.log` in version 4.1.3) is the file that keeps track of the next trace file number to be assigned.

- `ndb_<NodeID>_out.log` is a file containing any data output by the **ndbd** process. This file is created only if **ndbd** is started as a daemon, which is the default behavior beginning with MySQL 4.1.5. This file was named `node<NodeID>.out` in versions 4.1.3 and 4.1.4.

- `ndb_<NodeID>.pid` is a file containing the process ID of the **ndbd** process when started as a daemon. It also functions as a lock file to avoid the starting of nodes with the same identifier.

- `ndb_<NodeID>_signal.log` (was `Signal.log` in version 4.1.3) is a file used only in debug versions of **ndbd**, where it is possible to trace all incoming, outgoing, and internal messages with their data in the **ndbd** process.

It is recommended not to use a directory mounted through NFS because in some environments this can cause problems whereby the lock on the pid-file remains in effect even after the process has terminated.

When starting **ndbd** it may also be necessary to specify the hostname of the management server and the port on which it is listening. Optionally, one may also specify the node ID that the process is to use.

```
shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

(See Section 16.5.4.2, “The MySQL Cluster `connectstring`” for additional information about this issue.)

When **ndbd** starts, it actually initiates two processes. The first of these is called the "angel process"; its only job is to discover when the execution of the execution process has been completed, and then to restart the **ndbd** process if it is configured to do so. Thus, if you attempt to kill **ndbd** via the Unix **kill** command, it is necessary to kill both processes. A more proper way to terminate an **ndbd** process is to use the management client and stop the process from there.

The execution process uses one thread for reading, writing, and scanning data, as well as all other activities. This thread is implemented asynchronously so that it can easily handle thousands of concurrent activites. In addition, a watch-dog thread supervises the execution thread to make sure that it does not hang in an endless loop. A pool of threads handles file I/O, with each thread able to handle one open file. Threads can also be used for transporter connections by the transporters in the **ndbd** process. In a system performing a large number of operations, including updates, the **ndbd** process can consume up to 2 CPUs if permitted to do so. For a machine with many CPUs it is recommended to use several **ndbd** processes which belong to different node groups.

16.6.3. ndb_mgmd, the Management Server Process

The management server is the process that reads the cluster configuration file and distributes this information to all nodes in the cluster that request it. It also maintains a log of cluster activities. Management clients can connect to the management server and check the cluster's status.

As of MySQL 4.1.5, it is no longer necessary to specify a connectstring when starting the management server. However, if you are using more than one management server, a connectstring should be provided and each node in the cluster should specify its node ID explicitly.

The following files are created or used by **ndb_mgmd** in its starting directory. From MySQL 4.1.5, the log and PID files are placed in the `DataDir` as specified in the configuration file. In the list that follows, `<NodeID>` is the unique node identifier.

- `config.ini` is the configuration file for the cluster as a whole. This file is created by the user and read by the management server. Setting up this file is discussed in Section 16.5, “MySQL Cluster Configuration”.
- `ndb_<NodeID>_cluster.log` (was `cluster.log` in version 4.1.3) is the cluster events log file. Examples of such events include checkpoint startup and completion, node startup events, node failures, and levels of memory usage. A complete listing of cluster events with descriptions may be found in Section 16.7, “Management of MySQL Cluster”.

  When the size of the cluster log reaches one million bytes then the file is renamed to `ndb_<NodeID>_cluster.log.SeqID` (was `cluster.log.SeqID` where `SeqID` is the sequence number of the cluster log file. (For example: If 1, 2, and 3 already exist, the next log file will be named using the number **4**.)
- `ndb_<NodeID>_out.log` (was `node1.out` in version 4.1.3) is the file used for `stdout` and `stderr` when running the management server as a daemon.
- `ndb_<NodeID>.pid` (was `node<NodeID>.pid` in version 4.1.3) is the PID file used when running the management server as a daemon.

16.6.4. ndb_mgm, the Management Client Process

The management client process is actually not needed to run the cluster. Its value lies in providing a set of commands for checking the cluster's status, starting backups, and performing other administrative functions. The management client access the management server using a C API that advanced users can also employ for programming dedicated management processes which can perform tasks similar to those performed by **ndb_mgm**.

When starting the management client, it is necessary to supply the hostname and port of the management server as in the example below. The default values are **localhost** and **1186** (prior to MySQL 4.1.8, the default Cluster port was 2200).

```
shell> ndb_mgm localhost 1186
```

Additional information about using **ndb_mgm** can be found in Section 16.6.5.4, “Command Options for **ndb_mgm**” and Section 16.7.1, “Commands in the Management Client”.

16.6.5. Command Options for MySQL Cluster Processes

All MySQL Cluster executables (except for **mysqld**) take the following options as of 4.1.8. If you are using an earlier version, please read carefully, as we have made changes in some of these switches in or-

der to make them consistent with one another as well as with **mysqld**. (For example, the `-V` switch was `-v` in earlier versions.) You can also use the `-?` option to see what is supported in your version.

- `-?, --usage, --help`

  Prints a short list with descriptions of the available command options.

- `-V, --version`

  Prints the version number of the **ndbd** process. The version number is the MySQL Cluster version number. The version number is relevant because not all versions can be used together, and at startup the MySQL Cluster processes verifies that the versions of the binaries being used can co-exist in the same cluster. This is also important when performing an online software upgrade of MySQL Cluster (see `Software Upgrade of MySQL Cluster`).

- `-c` *`connect_string`*`, --connect-string`

  `connect_string` Sets the connect string to the management server as a command option. (for reasons of backwards compatability, **ndb_mgmd** does not accept the -c option before MySQL 5.0, as this switch specifies the configuration file in previous versions). Available with **ndb_mgm** from MySQL 4.1.8 onwards.

  ```
  shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1186"
  ```

- `--debug[=`*`options`*`]`

  This option can only be used for versions compiled with debugging enabled. It is used to enable output from debug calls in the same manner as for the **mysqld** process.

16.6.5.1. MySQL Cluster-Related Command Options for mysqld

- `--ndbcluster`

  If the binary includes support for the NDB Cluster storage engine the default disabling of support for the NDB storage engine can be overridden by using this option. Use of the NDB Cluster storage engine is necessary for using MySQL Cluster.

- `--skip-ndbcluster`

  Disables the NDB Cluster storage engine. This is disabled by default for binaries where it is included; the option only applies if the server was configured to use the NDB Cluster storage engine.

- `--ndb-connectstring=`*`connect_string`*

  When using the NDB storage engine, it is possible through setting this option to specify the management server that distributes cluster configuration data.

16.6.5.2. Command Options for ndbd

For some common options see Section 16.6.5, “Command Options for MySQL Cluster Processes”.

- `-d, --daemon`

Instructs **ndbd** to execute as a daemon process. From MySQL 4.1.5 on, this is the default behavior.

- `--nodaemon`

  Instructs **ndbd** not to start as a daemon process. Useful when **ndbd** is being debugged and one wants output redirected to the screen.

- `--initial`

  Instructs **ndbd** to perform an initial start. An initial start erases any files created for recovery purposes by earlier instances of **ndbd**. It also re-creates recovery log files. Note that on some operating systems this process can take a substantial amount of time.

  An `--initial` start is to be used only the very first time that the **ndbd** process is started, as it removes all files from the Cluster filesystem and re-creates all REDO log files. The exceptions to this rule are:

  - When performing a software upgrade which has changed the contents of any files.
  - When restarting the node with a new version of **ndbd**.
  - As a measure of last resort when for some reason the node restart or system restart repeatedly fails. In this case be aware that that this node can no longer be used to restore data due to the destruction of the datafiles.

  This option does not affect any backup files that have been created by the affected node.

  In older versions of MySQL Cluster, it was possible to use `-i` for this option. This shortcut was removed to prevent this option from being used by mistake.

- `--nostart`

  Instructs **ndbd** not to start automatically. When this option is used, **ndbd** connects to the management server, obtains configuration data from it, and initialises communication objects. However, it will not actually start the execution engine until specifically requested to do so by the management server. This can be accomplished by issuing the proper command to the management client.

16.6.5.3. Command Options for ndb_mgmd

For some common options see Section 16.6.5, “Command Options for MySQL Cluster Processes”.

- `-f` *`filename`*`, --config-file=`*`filename`*`, -c` *`filename`*

  Instructs the management server as to which file it should use for its configuration file. This option must be specified. The file name defaults to `config.ini`. The `-f` shortcut is available beginning with MySQL Cluster 4.1.8. The `-c` shortcut became obsolete in MySQL Cluser 5.0.

- `-d, --daemon`

  Instructs **ndb_mgmd** to start as a daemon. This is the default behavior.

- `-nodaemon`

  Instructs the management server not to start as a daemon.

16.6.5.4. Command Options for ndb_mgm

For some common options see Section 16.6.5, “Command Options for MySQL Cluster Processes”.

- `[host_name [port_num]]`

  To start the management client it is necessary to specify where the management server resides, which means specifying the hostname and the port. The default hostname is `localhost` and the default port is 1186 (this was port 2200 prior to version 4.1.8).

- `--try-reconnect=number`

  If the connection to the management server is broken, then the node tries to reconnect to it every 5 seconds until it succeeds. By using this option, it is possible to limit the number of attempts to *number* before giving up and reporting an error instead.

16.7. Management of MySQL Cluster

Managing a MySQL Cluster involves a number of tasks, the first of which is to configure and start MySQL Cluster. This is covered in Section 16.5, “MySQL Cluster Configuration” and Section 16.6, “Process Management in MySQL Cluster”.

The following sections cover the management of a running MySQL Cluster.

There are essentially two methods of actively managing a running MySQL Cluster. The first of these is through the use of commands entered into the management client whereby cluster status can be checked, log levels changed, backups started and stopped and nodes stopped and started. The second method involves studying the contents of the cluster log `ndb_<NodeID>_cluster.log` in the management server's `DataDir` directory. (Recall that `<NodeID>` represents the unique identifier of the node whose activity is being logged.) The cluster log contains event reports generated by **ndbd**. It is also possible to send cluster log entries to a Unix system log.

16.7.1. Commands in the Management Client

In addition to the central configuration file, a cluster may also be controlled through a command line interface available through the management client **ndb_mgm**. This is the primary administrative interface to a running cluster.

The management client has the following basic commands. In the listing that follows, `<id>` denotes either a database node ID or the keyword `ALL`, which indicates that the command should be applied to all of the cluster's storage nodes.

- `HELP`

  Displays information on all available commands.

- `SHOW`

  Displays information on the cluster's status.

- `<id> START`

  Starts the storage node identified by `<id>` (or all storage nodes).

- `<id> STOP`

  Stops the storage node identified by `<id>` (or all storage nodes).

- `<id> RESTART [-N] [-I]`

  Restarts the storage node identified by `<id>` (or all storage nodes).

- `<id> STATUS`

  Displays status information for the storage node identified by `<id>` (or all storage nodes).

- `ENTER SINGLE USER MODE <id>`

  Enters single-user mode, whereby only the MySQL server identified by the node ID `<id>` is allowed to access the database.

- `EXIT SINGLE USER MODE`

  Exists single user mode allowing all SQL nodes (that is, all running **mysqld** processes) to access the database.

- `QUIT`

  Terminates the management client.

- `SHUTDOWN`

  Shuts down all cluster nodes, except for SQL nodes, and exits.

Commands for the event logs are given in the next section; commands for creating backups and restoring from backup are provided in a separate section on these topics.

16.7.2. Event Reports Generated in MySQL Cluster

In this section, we discuss the types of event logs provided by MySQL Cluster, and the types of events that are logged.

MySQL Cluster provides two types of event log. These are the **cluster log**, which includes events generated by all cluster nodes, and **node logs**, which are local to each storage node.

Output generated by cluster event logging can have multiple destinations including a file, the management server console window, or `syslog`. Output generated by node event logging is written to the storage node's console window.

Both sorts of event logs can be set to log different subsets of events.

Note: The cluster log is the log recommended for most uses, since it provides logging information for an entire cluster in a single file. Node logs are intended to be used only during application development, or for debugging application code.

Each reportable event can be distinguished according to 3 different criteria:

- *Category*: This can be any one of the following values: `STARTUP`, `SHUTDOWN`, `STATISTICS`, `CHECKPOINT`, `NODERESTART`, `CONNECTION`, `ERROR`, or `INFO`.

- *Priority*: This is represented by a number 1 - 15 inclusive, where 1 indicates "most important" and

15 "least important".

- *Severity Level*: This can be any one of the following values: `ALERT`, `CRITICAL`, `ERROR`, `WARNING`, `INFO`, or `DEBUG`.

Both the cluster log and the node log can be filtered on these properties.

16.7.2.1. Logging Management Commands

The following management commands are related to the cluster log:

- `CLUSTERLOG ON`

  Turn cluster log on.

- `CLUSTERLOG OFF`

  Turn cluster log off.

- `CLUSTERLOG INFO`

  Information about cluster log settings.

- `<id> CLUSTERLOG <category>=<threshold>`

  Logs *`category`* events with priority less than or equal to *`threshold`* in the cluster log.

- `CLUSTERLOG FILTER <severity>`

  Toggles cluster logging of events of the specified *`severity`*.

The following table describes the default setting (for all storage nodes) of the cluster log category threshold. If an event has a priority with a value lower than or equal to the priority threshold, then it is reported in the cluster log.

Note that events are reported per storage node, and that the threshold can be set to different values on different nodes.

Category	Default threshold (All storage nodes)
STARTUP	**7**
SHUTDOWN	**7**
STATISTICS	**7**
CHECKPOINT	**7**
NODERESTART	**7**
CONNECTION	**7**
ERROR	**15**
INFO	**7**

Thresholds are used to filter events within each category. For example, a `STARTUP` event with a priority of 3 is not logged unless the threshold for `STARTUP` is changed to 3 or lower. Only events with priority 3 or lower are sent if the threshold is 3.

Event severity levels are shown below. (**Note**: These correspond to UNIX `syslog` levels, except for `LOG_EMERG` and `LOG_NOTICE`, which are not used or mapped.)

1	`ALERT`	A condition that should be corrected immediately, such as a corrupted system database
2	`CRITICAL`	Critical conditions, such as device errors or insufficient resources
3	`ERROR`	Conditions that should be corrected, such as configuration errors
4	`WARNING`	Conditions that are not errors, but that might require handling
5	`INFO`	Informational messages
6	`DEBUG`	Debugging messages used for NDB Cluster development

Event severity levels can be turned on or off. If a severity level is turned on, then all events with a priority less than or equal to the category thresholds are logged. If the severity level is turned off then no events belonging to that severity level are logged.

16.7.2.2. Log Events

An event report as reported in the event logs has the following format: `<datetime> [<string>] <severity> -- <message>`. For example:

```
09:19:30 2005-04-24 [NDB] INFO -- Node 4 Start phase 4 completed
```

All reportable events are discussed in this section, ordered by category and severity level within each category.

`CONNECTION` Events

These are events associated with connections between Cluster nodes.

Event	Priority	Severity Level	Description
DB nodes connected	8	INFO	Storage nodes connected
DB nodes disconnected	8	INFO	Storage nodes disconnected
Communication closed	8	INFO	SQL or storage node connection closed
Communication opened	8	INFO	SQL or storage node connection opened

`CHECKPOINT` Events

The logging messages shown below are assocated with checkpoints.

(**Note**: GCP = Global Checkpoint, LCP = Local Checkpoint.)

Event	Priority	Severity Level	Description
LCP stopped in calc keep GCI	0	ALERT	LCP stopped
Local checkpoint fragment com-	11	INFO	LCP on a fragment has been completed

pleted			
Global checkpoint completed	10	INFO	GCP finished
Global checkpoint started	9	INFO	Start of GCP: REDO log is written to disk
Local checkpoint completed	8	INFO	LCP completed normally
Local checkpoint started	7	INFO	Start of LCP: data written to disk
Report undo log blocked	7	INFO	UNDO logging blocked; buffer near overflow

`STARTUP` Events

The following events are generated in response to the startup of a node or of the cluster and of its success or failure. They also provide information relating to the progress of the startup process, including information concerning logging activities.

Event	Priority	Severity Level	Description
Internal start signal received STTORRY	15	INFO	Blocks received after completion of restart
Undo records executed	15	INFO	
New REDO log started	10	INFO	GCI keep *X*, newest restorable GCI *Y*
New log started	10	INFO	Log part *X*, start MB *Y*, stop MB *Z*
Node has been refused to be included into the cluster	8	INFO	Node cannot be included in cluster due to misconfiguration, inability to establish communication, or other problem
DB node neighbours	8	INFO	Shows neighbouring storage nodes
DB node start phase *X* completed	4	INFO	A storage node start phase has been completed
Node has been successfully included into the cluster	3	INFO	Displays the node, managing node, and dynamic ID
DB node start phases initiated	1	INFO	NDB Cluster nodes starting
DB node all start phases completed	1	INFO	NDB Cluster nodes started
DB node shutdown initiated	1	INFO	Shutdown of storage node has commenced
DB node shutdown aborted	1	INFO	Unable to shut down storage node normally

`NODERESTART` Events

The following events are generated when restarting a node and relate to the success or failure of the node restart process.

Event	Priority	Severity Level	Description
Node failure phase completed	8	ALERT	Reports completion of node failure phases
Node has failed, node state was *X*	8	ALERT	Reports that a node has failed
Report arbitrator results	2	ALERT	There are 8 different possible results for arbitration attempts: * Arbitration check failed - less than 1/2 nodes left * Arbitration check succeeded - node group majority * Arbitration check failed - missing node group * Network partitioning - arbitration required * Ar-

			bitration succeeded - affirmative response from node *X* * Arbitration failed - negative response from node *X* * Network partitioning - no arbitrator available * Network partitioning - no arbitrator configured
Completed copying a fragment	10	INFO	
Completed copying of dictionary information	8	INFO	
Completed copying distribution information	8	INFO	
Starting to copy fragments	8	INFO	
Completed copying all fragments	8	INFO	
GCP takeover started	7	INFO	
GCP takeover completed	7	INFO	
LCP takeover started	7	INFO	
LCP takeover completed (state = *X*)	7	INFO	
Report whether an arbitrator is found or not	6	INFO	There are 7 different possible outcomes when seeking an arbitrator: * Management server restarts arbitration thread [state=*X*] * Prepare arbitrator node *X* [ticket=*Y*] * Receive arbitrator node *X* [ticket=*Y*] * Started arbitrator node *X* [ticket=*Y*] * Lost arbitrator node *X* - process failure [state=*Y*] * Lost arbitrator node *X* - process exit [state=*Y*] * Lost arbitrator node *X* <error msg>[state=*Y*]

`STATISTICS` Events

The following events are of a statistical nature. They provide information such as numbers of transactions and other operations, amount of data sent or received by individual nodes, and memory usage.

Event	Priority	Severity Level	Description
Report job scheduling statistics	9	INFO	Mean internal job scheduling statistics
Sent number of bytes	9	INFO	Mean number of bytes sent to node *X*
Received # of bytes	9	INFO	Mean number of bytes received from node *X*
Report transaction statistics	8	INFO	Number of: transactions, commits, reads, simple reads, writes, concurrent operations, attribute info, aborts
Report operations	8	INFO	Number of operations
Report table create	7	INFO	
Memory usage	5	INFO	Data and index memory usage (80%, 90% and 100%)

`ERROR` Events

These events relate Cluster errors and warnings and generally indicate that a major malfunction or failure has occurred.

Event	Priority	Severity	Description
Dead due to missed heartbeat	8	ALERT	Node *X* declared dead due to missed heartbeat
Transporter errors	2	ERROR	
Transporter warnings	8	WARN-ING	
Missed heartbeats	8	WARN-ING	Node *X* missed heartbeat #*Y*
General warning events	2	WARN-ING	

`INFO` Events

These events provide general information about the state of the cluster and activities associated with Cluster maintenance, such as logging and heartbeat transmission.

Event	Priority	Severity	Description
Sent heartbeat	12	INFO	Heartbeat sent to node *X*
Create log bytes	11	INFO	Log part, log file, MB
General info events	2	INFO	

16.7.3. Single User Mode

Single user mode allows the database administrator to restrict access to the database system to a single MySQL server (SQL node). When entering single user mode all connections to all other MySQL servers are closed gracefully and all running transactions are aborted. No new transactions are allowed to be started.

Once the cluster has entered single user mode, only the designated SQL node is granted access to the database. You can use the **all status** command to see when the cluster has entered single user mode.

Example:

```
ENTER SINGLE USER MODE 5
```

After this command has executed and the cluster has entered single user mode, the SQL node whose node ID is **5** becomes the cluster's only permitted user.

The node specified in the command above must be a MySQL Server node; An attempt to specify any other type of node will be rejected.

Note: When the above coommand is invoked, all transactions running on the designated node are aborted, the connection is closed, and the server must be restarted.

The command **EXIT SINGLE USER MODE** changes the state of the cluster's storage nodes from single user mode to normal mode. MySQL Servers waiting for a connection (that is, for the cluster to become ready and available), are now allowed to connect. The MySQL Server denoted as the single-user SQL node continues to run (if it is connected), during and after the state change.

Example:

```
EXIT SINGLE USER MODE
```

The recommend way to a node failure when running in single user mode is to do either of the following:

-
  1. Finish all single user mode transactions
  2. Issue the **EXIT SINGLE USER MODE** command
  3. Restart the cluster's storage nodes

or

- Or restart database nodes prior to entering single user mode.

16.7.4. On-line Backup of MySQL Cluster

This section describes how to create a backup and how to restore the database from a backup at a later time.

16.7.4.1. Cluster Backup Concepts

A backup is a snapshot of the database at a given time. The backup consists of three main parts:

- **Metadata**: the names and definitions of all database tables.
- **Table records**: the data actually stored in the database tables at the time that the backup was made
- **Transaction log**: a sequential record telling how and when data was stored in the database

Each of these is saved on all nodes participating in the backup. During backup each node saves these three parts into three files on disk:

- `BACKUP-<BackupId>.<NodeId>.ctl`

  A control file containing control information and metadata. Each node saves the same table definitions (for all tables in the cluster) to its own version of this file.

- `BACKUP-<BackupId>-0.<NodeId>.data`

  A data file containing the table records, which are saved on a per-fragment basis; that is, different nodes save different fragments during the backup. The file saved by each node starts with a header that states to which tables the records belong. Following the list of records there is a footer containing a checksum for all records.

- `BACKUP-<BackupId>.<NodeId>.log`

  A log file containing records of committed transactions. Only transactions on tables stored in the backup are stored in the log. Nodes involved in the backup save different records, since different nodes host different database fragments.

In the table above, *<BackupId>* stands for the backup identifier and *<NodeId>* is the unique identifier for the node creating the file.

16.7.4.2. Using The Management Server to Create a Backup

Before starting a backup, make sure that the cluster is properly configured for performing one. (See Section 16.7.4.4, “Configuration for Cluster Backup”.)

To create a backup using the management server involves the following steps:

1. Start the management server (**ndb_mgm**).
2. Execute the command `START BACKUP`.
3. The management server will reply with a message '``Start of backup ordered''. This means that the management server has submitted the request to the cluster, but has not yet received any response.
4. The management server will reply '``Backup *<BackupId>* started'', where *<BackupId>* is the unique identifier for this particular backup. (This ID will also be saved in the cluster log, if it has not been configured otherwise.) This means that the cluster has received and processed the backup request. It does not mean that the backup has completed.
5. The management server will signal that the backup is finished with the message '``Backup *<BackupId>* completed''.

To abort a backup already in progress:

1. Start the management server.
2. Execute the command '`ABORT BACKUP` *<BackupId>*'. The number *<BackupId>* is the identifier of the backup that was included in the response of the management server when the backup was started (in the message '``Backup *<BackupId>* started'').
3. The management server will acknowledge the abort request with '``Abort of backup *<BackupId>* ordered''; note that no actual response to this request has yet been received.
4. Once the the backup has been aborted, the management server will report '``Backup *<BackupId>* has been aborted for reason *XYZ*''. This means that the cluster has terminated the backup and that all files re lated to this backup have been removed from the cluster filesystem.

Note: If there is no backup with ID *<BackupId>* running when it is aborted, the management server will not make any exlicit response. However, there will be a line in the cluster log indicating that an invalid abort command was sent.

16.7.4.3. How to Restore a Cluster Backup

The cluster restoration program is implemented as separate command line utility **ndb_restore**, which reads the files created by the backup and inserts the stored information into the database. The restore program must be executed once for each set of backup files, that is, as many times as there were database nodes running when the backup was created.

The first time you run the restoration program, you also need to restore the metadata; in other words,

you must re-create the database tables. (Note that the cluster should have an empty database when starting to restore a backup.) The restore program acts as an API to the cluster and therefore requires a free connection in order to connect to the cluster. This can be verified with the **ndb_mgm** command **SHOW**. The switch `-c <connectstring>` may be used to locate the MGM node (see Section 16.5.4.2, "The MySQL Cluster `connectstring`" for information on connectstrings). The backup files must be present in the directory given as an argument to the restoration program.

It is possible to restore a backup to a database with a different configuration than it was created from. For example, suppose that a backup with backup ID **12**, created in a cluster with two database nodes having the node IDs **2** and **3** is to be restored to a cluster with four nodes. Then **ndb_restore** must be run twice - once for each database node in the cluster where the backup was taken.

Note: For rapid restoration, the data may be restored in parallel, provided that there is a sufficient number of cluster connections available. However, the data files must always be applied before the logs.

16.7.4.4. Configuration for Cluster Backup

There are four configuration parameters essential for backup:

- `BackupDataBufferSize`

  Amount of memory used to buffer data before it is written to disk.

- `BackupLogBufferSize`

  Amount of memory used to buffer log records before these are written to disk.

- `BackupMemory`

  Total memory allocated in a database node for backups. This should be the sum of the memory allocated for the backup data buffer and the backup log buffer.

- `BackupWriteSize`

  Size of blocks written to disk. This applies for both the backup data buffer and the backup log buffer.

More detailed information about these parameters can be found in Section 16.5, "MySQL Cluster Configuration".

16.7.4.5. Backup Troubleshooting

If an error code is returned when issuing a backup request, the most likely cause is insufficient memory or insufficient disk space. You should check that there is enough memory allocated for the backup. Also check that there is enough space on the hard drive partition of the backup target.

16.8. Using High-Speed Interconnects with MySQL Cluster

Even before design of NDB Cluster began in 1996, it was evident that one of the major problems to be encountered in building parallel databases would be communication between the nodes in the network. For this reason, NDB Cluster was designed from the very to allow for different transporters.

Currently the code base includes 4 different transporters. Most users today employ TCP/IP over Ethernet since this ubiquitous. This is also by far the best-tested transporter in MySQL Cluster.

We are working to make sure that communication with the **ndbd** process is made in as large chunks as possible since this will benefit all types of data transmission.

For users who desire it, it is also possible to use cluster interconnects to enhance performance even further. There are two ways to achieve this: either a custom transporter can be designed to handle this case, or one can use socket implementations that bypass the TCP/IP stack to one extent or another. We have experimented with both of these techniques using the SCI (Scalable Coherent Interface) technology developed by Dolphin [http://www.dolphinics.com/].

16.8.1. Configuring MySQL Cluster to use SCI Sockets

In this section we will show how one can use a cluster configured for normal TCP/IP communication to use SCI Sockets instead. This documentation is based on SCI Sockets version 2.3.0 as of 1 October 2004.

Prerequisites

Any machines with which you wish to use SCI Sockets need to be equipped with SCI cards.

It is possible to use SCI Sockets with any version of MySQL Cluster. No special builds are needed since it uses normal socket calls which are already available in MySQL Cluster. However, SCI Sockets are currently supported only on Linux 2.4 and 2.6 kernels. SCI Transporters works on additional operating systems although we have verified these only with Linux 2.4 to date.

There are essentially four requirements for SCI Sockets. First, it is necessary to build the SCI Socket libraries. Second, the SCI Socket kernel libraries need to be installed. Third, one or two configuration files need to be installed. Finally, the SCI Socket kernel library needs to be enabled either for the entire machine or for the shell where the MySQL Cluster processes are started. This process needs to be repeated for each machine in the cluster where you plan to use SCI Sockets for inter-node communication.

Two packages need to be retrieved to get SCI Sockets working. The first of these builds the libraries on which SCI Sockets are built; the second contains the SCI Socket libraries themselves. Currently, these are available only in source code format. The latest versions of these packages at the time of this writing were `DIS_GPL_2_5_0_SEP_10_2004.tar.gz` and `SCI_SOCKET_2_3_0_OKT_01_2004.tar.gz`> You should be able to find them (or possibly newer versions) at http://www.dolphinics.no/support/downloads.html.

Package Installation

Once you have obtained the library packages, the next step is to unpack them into appropriate directories, with the SCI Sockets library unpacked into a directory below the DIS code. Next, you need to build the libraries. This example shows the commands used on Linux/x86 to perform this task:

```
shell> tar xzf DIS_GPL_2_5_0_SEP_10_2004.tar.gz
shell> cd DIS_GPL_2_5_0_SEP_10_2004/src/
shell> tar xzf ../../SCI_SOCKET_2_3_0_OKT_01_2004.tar.gz
shell> cd ../adm/bin/Linux_pkgs
shell> ./make_PSB_66_release
```

It is possible to build these libraries for some 64-bit procesors. To build the libraries for Opteron CPUs using the 64-bit extensions, run **make_PSB_66_X86_64_release** rather than **make_PSB_66_release**; if the build is made on an Itanium machine, then you should use **make_PSB_66_IA64_release**. The X86-64 variant should work for Intel EM64T architectures but this has not yet (to our knowledge) been tested.

Once the build process is complete, the compiled libraries will be found in a zipped tar file with a name along the lines of `DIS-<operating-system>-time-date`. It is now time to install the package

in the proper place. In this example we will place the installation in `/opt/DIS`. (**Note**: You will most likely need to run the following as the root user.)

```
shell> cp DIS_Linux_2.4.20-8_181004.tar.gz /opt/
shell> cd /opt
shell> tar xzf DIS_Linux_2.4.20-8_181004.tar.gz
shell> mv DIS_Linux_2.4.20-8_181004 DIS
```

Network Configuration

Now that all the libraries and binaries are in their proper place we need to ensure that the SCI cards have proper node IDs within the SCI address space.

It is also necessary to decide on the network structure before proceeding. There are three types of network structures which can be used in this context:

- A simple one-dimensional ring
- One or more SCI switches with one ring per switch port
- A two- or three-dimensional torus.

Each of these topologies has its own method for providing node IDs. We discuss each of them in brief.

A simple ring uses node IDs which are nonzero multiples of 4: 4, 8, 12,...

The next possibility uses SCI switches. An SCI switch has 8 ports, each of which can support a ring. It is necessary to make sure that that different rings use different node ID spaces. In a typical configuration, the first port uses node IDs below 64 (4 - 60), the next 64 node IDs (68 - 124) are assigned to the next port, and so on, with node IDs 452 - 508 being assigned to the eighth port.

Two- and three-dimensional torus network structures take into account where each node is located in each dimension, incrementing by 4 for each node in the first dimension, by 64 in the second dimension, and (where applicable) by 1024 in the third dimension. See Dolphin's Web site [http://www.dolphinics.com/support/index.html] for more thorough documentation.

In our testing we have used switches, although most large cluster installations use the 2-D/3-D torus structures. The advantage provided by switches is that, with dual SCI cards and dual switches, it is possible to build relatively easily a redundant network where the average failover time on the SCI network is on the order of 100 microseconds. This is supported by the SCI transporter in MySQL Cluster and is also under development for the SCI Socket implementation.

Failover for the 2-D/3-D torus is also possible but requires sending out new routing indexes to all nodes. However, this requires only 100 milliseconds or so to complete and should be acceptable for most high-availability cases.

By placing cluster storage nodes properly within the switched architecture, it is possible to use 2 switches to build a structure whereby 16 computers can be interconnected and no single failure can hinder more than one computer. With 32 computers and 2 switches it is possible to configure the cluster in such a manner that no single failure can cause the loss of more than two nodes; in this case it is also possible to know which pair of nodes will be affected. Thus, by placing the two in separate node groups, it is possible to build a safe MySQL Cluster installation.

To set the node ID for an SCI card use the following command in the `/opt/DIS/sbin` directory. In this example `-c 1` refers to the number of the SCI card (this will always be 1 if there is only 1 card in the machine), `-a 0` refers to adapter 0; and `68` is the node ID:

```
shell> ./sciconfig -c 1 -a 0 -n 68
```

If you have multiple SCI cards in the same machine, you can determine which card has which slot by issuing the following command:

```
shell> ./sciconfig -c 1 -gsn
```

This will give you the SCI card's serial number. Then repeat this procedure for `-c 2` and so on, for each card in the machine. Once you have matched each card with a slot, you can set node IDs for all cards.

After installing the necessary libraries and binaries, and setting the SCI node IDs, the next step is to set up the mapping from hostnames (or IP addresses) to SCI node IDs. This is done in the SCI sockets configuration file, which should be saved as `/etc/sci/scisock.conf`. In this file, the SCI node ID will be mapped to the hostname or IP address to communicate with, through the proper SCI card. Here is a very simple example of such a configuration file:

```
#host           #nodeId
alpha           8
beta            12
192.168.10.20   16
```

It is also possible to limit this configuration so that it applies only to a subset of the available ports for these hosts. An additional configuration file `/etc/sci/scisock_opt.conf` can be used to accomplish this, as shown here:

```
#-key                        -type        -values
EnablePortsByDefault              yes
EnablePort                  tcp           2200
DisablePort                 tcp           2201
EnablePortRange             tcp           2202 2219
DisablePortRange            tcp           2220 2231
```

Driver Installation

With the configuraiton files in place, the drivers can be installed. First, the the low-level drivers and then the SCI socket driver need to be installed:

```
shell> cd DIS/sbin/
shell> ./drv-install add PSB66
shell> ./scisocket-install add
```

If desired, the installation can be checked by invoking a script which verifies that all nodes in the SCI socket configuration files are accessible:

```
shell> cd /opt/DIS/sbin/
shell> ./status.sh
```

If you discover an error and need to change the SCI socket configuration, it is necessary to use the **ksocketconfig** to accomplish this task:

```
shell> cd /opt/DIS/util
shell> ./ksocketconfig -f
```

Testing the Setup

To make sure that SCI sockets are actually being used, you can use the test program **latency_bench**. Using this utility's server component, clients can connect to the server to test the latency of the connection; determining whether or not SCI is enabled should be fairly simple from observing the latency. (**Note**: Before using **latency_bench**, it is necessary to set the `LD_PRELOAD` environment variable as shown later in this section.)

To set up a server, use the following:

```
shell> cd /opt/DIS/bin/socket
shell> ./latency_bench -server
```

To run a client, use **latency_bench** again, except this time with the `-client` option:

```
shell> cd /opt/DIS/bin/socket
shell> ./latency_bench -client hostname_of_server
```

SCI socket configuration should now be complete and MySQL Cluster ready to use both SCI Sockets and the SCI transporter (see Section 16.5.4.9, “MySQL Cluster SCI Transport Connections”).

Starting the Cluster

The next step in the process is to start MySQL Cluster. To enable usage of SCI Sockets it is necessary to set the environment variable `LD_PRELOAD` before starting **ndbd**, **mysqld**, and **ndb_mgmd**. This variable should point to the kernel library for SCI Sockets.

To start **ndbd** in a bash shell, do the following:

```
bash-shell> export LD_PRELOAD=/opt/DIS/lib/libkscisock.so
bash-shell> ndbd
```

In a tcsh environment the same thing can be accomplished with:

```
tcsh-shell> setenv LD_PRELOAD=/opt/DIS/lib/libkscisock.so
tcsh-shell> ndbd
```

Note: MySQL Cluster can use only the kernel variant of SCI Sockets.

16.8.2. Understanding the Impact of Cluster Interconnects

The **ndbd** process has a number of simple constructs which are used to access the data in a MySQL Cluster. We have created a very simple benchmark to check the performance of each of these and the effects which various interconnects have on their performance.

There are four access methods:

- **Primary key access**

  This is the simple access of a record through its primary key. In the simplest case only one record is accessed at a time, which means that the full cost of setting up a number of TCP/IP messages and a number of costs for context switching are borne by this single request. In the case where multiple primary key accesses are sent in one batch then those accesses will share the cost of setting up the necessary TCP/IP messages and context switches. If the TCP/IP messages are for different destinations, then additional TCP/IP messages need to be set up.

- **Unique key access**

  Unique key accesses are similar to primary key accesses, except that a unique key accesses is executed as a read on an index table followed by a primary key access on the table. However, only one request is sent from the MySQL Server, and the read of the index table is handled by **ndbd**. Such requests also benefit from batching.

- **Full table scan**

  When no indexes exist for a lookup on a table, then a full scan of a table is performed. This is sent as a single request to the **ndbd** process, which then divides the table scan into a set of parallel scans on all cluster **ndbd** processes. In future versions of MySQL Cluster, an SQL node will be able to filter some of these scans.

- **Range scan using ordered index**

  When an ordered index is used, it performs a scan in the same manner as the full table scan, except that it scans only those records which are in the range used by the query transmitted by the MySQL server (SQL node). A special optimisation is planned for future versions of MySQL Cluster whereby only one partition will be scanned instead of all partitions in parallel when all bound index attributes include all attributes in the partitioning key.

To check the base performance of these access methods we have developed a set of benchmarks. One such benchmark, **testReadPerf**, tests simple and batched primary and unique key accesses. This benchmark also measures the setup cost of range scans by issuing scans returning a single record. There is also a variant of this benchmark which uses a range scan to fetch a batch of records.

In this way, we can test the cost of single key access and single record scan accesses, and measure the impact of the communication media used, on base access methods.

In our tests, we ran the base benchmarks for both a normal transporter using TCP/IP sockets and a similar setup using SCI sockets. The figures reported below are for small accesses of 20 records per access. The difference between serial and batched access decreases by a factor of 3 to 4 when using 2 kB records instead. SCI Sockets were not tested with 2 kB records. Tests were performed on a cluster with 2 storage nodes running on 2 dual-CPU machines equipped with AMD MP1900+ processors.

```
Access type:         TCP/IP sockets           SCI Socket
Serial pk access:    400 microseconds         160 microseconds
Batched pk access:    28 microseconds          22 microseconds
Serial uk access:    500 microseconds         250 microseconds
Batched uk access:    70 microseconds          36 microseconds
Indexed eq-bound:   1250 microseconds         750 microseconds
Index range:          24 microseconds          12 microseconds
```

We also performed another set of tests to check the performance of SCI Sockets vis-à-vis that of the SCI transporter, and both of these as compared with the TCP/IP transporter. All these tests used primary key accesses either serially and multi-threaded, or multi-threaded and batched.

The tests showed that SCI sockets were about 100% faster than TCP/IP. The SCI transporter was faster in most cases compared to SCI sockets. One notable case occurred with many threads in the test program, which showed that the SCI transporter did not perform very well when used for the **mysqld** process.

Our overall conclusion was that, for most benchmarks, using SCI sockets improves performance by approximately 100% over TCP/IP, except in rare instances when communication performance is not an issue. This can occur when scan filters make up most of processing time or when very large batches of primary key accesses are achieved. In that case the CPU processing in the **ndbd** processes becomes a

fairly large part of the overhead.

Using the SCI transporter instead of SCI Sockets is only of interest in communicating between **ndbd** processes. Using the SCI transporter is also only of interest if a CPU can be dedicated to the **ndbd** process since the SCI transporter ensures that this process will never go to sleep. It is also important to ensure that the **ndbd** process priority is set in such a way that the process does not lose priority due to running for an extended period of time, as can be done by locking processes to CPU's in Linux 2.6. If such a configuration is possible, then the **ndbd** process will benefit by 10-70% compared to using SCI sockets. (The larger figures will be seen when performing updates and probably on parallel scan operaitons as well.)

There are several other optimized socket implementations for computer clusters, including Myrinet, Gigabit Ethernet, Infiniband and the VIA interface. We have tested MySQL Cluster so far only with SCI sockets. We also include documentation above on how to set up SCI sockets using ordinary TCP/IP for MySQL Cluster.

16.9. Cluster Limitations in MySQL 4.1

In this section, we provide a listing of known limitations in MySQL Cluster release 4.1 when compared to features available when using the MyISAM and InnoDB storage engines. Currently there are no plans to address these in coming releases of 4.1; howver, we will supply fixes for these issues in MySQL 5.0 and subsequent releases. If you check the Cluster category in the MySQL bugs database at http://bugs.mysql.com, you can find known bugs which (if marked 4.1) we intend to correct in upcoming releases of MySQL 4.1.

- **Noncompliance in syntax** (resulting in errors when running existing applications):
  - Not all charsets and collations supported; see Section D.10.5, “MySQL Cluster-4.1.6 (10 Oct 2004)” for a list of those that are supported.
  - There are no prefix indexes; only entire fields can be indexed.
  - Text indexes are not supported.
  - Geometry datatypes (WKT and WKB) are not supported.
- **Non compliance in limits/behavior** (may result in errors when running existing applications):
  - There is no partial rollback of transactions. A duplicate key or similar error will result in a rollback of the entire transaction.
  - A number of hard limits exist which are configurable, but available main memory in the cluster sets limits. See the complete list of configuration parameters in Section 16.5.4, “Configuration File”. Most configuration parameters can be upgraded online. These hard limits include:
    - Database memory size and index memory size (`DataMemory` and `IndexMemory`, repectively).
    - The maximum number of transactions that can be performed is set using the configuration parameter `MaxNoOfConcurrentOperations`. Note that bulk loading, `TRUNCATE TABLE`, and `ALTER TABLE` are handled as special cases by running multiple transactions, and so are not subject to this limitation.
    - Different limits related to tables and indexes. For example, the maximum number of ordered indexes per table is determined by `MaxNoOfOrderedIndexes`.
  - Database names, table names and attribute names cannot be as long in NDB tables as with other table handlers. Attibute names are truncated to 31 characters, and if not unique after truncation

give rise to errors. Database names and table names can total maximum of 122 characters. (That is, the maximum length for an NDB Cluster table name is 122 characters less the number of characters in the name of the database of which that table is a part.)

  - In MySQL 4.1 and 5.0, all CLuster table rows are of fixed length. This means (for example) that if a table has one or more VARCHAR fields containing only relatively small values, more memory and disk space will be required when using the NDB storage engine than would be for the same table and data using the MyISAM engine. We are working to rectify this issue in MySQL 5.1.
  - The maximum number of metadata objects is limited to 1600, including database tables, system tables, indexes and BLOBs.
  - The maximum number of attributes per table is limited to 128.
  - The maximum permitted size of any one row is 8k, not including data stored in BLOB columns.
  - The maximum number of attributes per key is 32.
- **Unsupported features** (do not cause errors, but are not supported or enforced):
  - The foreign key construct is ignored, just as it is in MyISAM tables.
  - Savepoints and rollbacks to savepoints are ignored as in MyISAM.
- **Performance and limitation-related issues**:
  - The query cache disabled since it is not invalidated if an update occurs on a different MySQL server
  - There are query performance issues due to sequential access to the NDB storage engine; it is also relatively more expensive to do many range scans than it is with either MyISAM or InnoDB.
  - The `Records in range` statistic is not supported, resulting in non-optimal query plans in some cases. Employ `USE INDEX` or `FORCE INDEX` as a workaround.
  - Unique hash indexes created with `USING HASH` cannot be used for accessing a table if `NULL` is given as part of the key.
- **Missing features**:
  - The only supported isolation level is `READ_COMMITTED`. (InnoDB supports `READ_COMMITTED`, `REPEATABLE_READ`, and `SERIALIZABLE`.)
  - No durable commits on disk. Commits are replicated, but there is no guarantee that logs are flushed to disk on commit.
- **Problems relating to multiple MySQL servers** (not relating to MyISAM or InnoDB):
  - `ALTER TABLE` is not fully locking when running multiple MySQL servers (no distributed table lock).
  - MySQL replication will not work correctly off if updates are done on multiple MySQL servers. However, if the database partitioning scheme done at the application level, and no transactions take place across these partitions, then replication can be made to work.
- **Issues exclusive to MySQL Cluster** (not related to MyISAM or InnoDB):

- All machines used in the cluster must have the same architecture; that is, all machines hosting nodes must be either big-endian or little-endian, and you cannot use a mixture of both. For example, you cannot have a management node running on a PPC which directs a server node that is running on an x86 machine. This restriction does not apply to machines simply running **mysql** clients that may be accessing the cluster.
- It is not possible to make online schema changes such as those accomplished using `ALTER TABLE` or `CREATE INDEX`.
- Online adding or dropping nodes is possible (the cluster must be restarted in such cases).
- When using multiple management servers one must give nodes explicit IDs in connectstrings since automatic allocation of node IDs does not work across multiple management servers.
- When using multiple management servers one must take extreme care to have the same configurations for all management servers. No special checks for this are performed by the cluster.
- The maximum number of storage nodes is 48.
- The total maximum number of nodes in a MySQL Cluster is 63. This number includes all MySQL Servers (SQL nodes), storage nodes, and management servers.

This listing is intended to be complete with respect to the conditions set forth at the beginning of this section. You can report any discrepancies that you encounter to the MySQL bugs database at http://bugs.mysql.com/. If we do not plan to fix the problem in MySQL 4.1, we will add it to the list above.

16.10. MySQL Cluster FAQ

- *What's the difference in using Cluster vs. using replication?*

  In a replication setup, a master MySQL server updates one or more slaves. Transactions are committed sequentially, and a slow transaction can cause the slave to lag behind the master. This means that if the master fails, it is possible that the slave might not have recorded the last few transactions. If a transaction-safe engine such as InnoDB is being used, then a transaction will either be complete on the slave or not applied at all, but replication does not guarantee that all data on the master and the slave will be consistent at all times. In MySQL Cluster, all storage nodes are kept in synch, and a transaction committed by any one storage node is committed for all storage nodes. In the event of a storage node failure, all remaining storage nodes will remain in a consistent state.

  In short, whereas MySQL replication is asynchronous, MySQL Cluster is synchronous.

- *Do I need to do any special networking to run Cluster? (How do computers in a cluster communicate?)*

  MySQL Cluster is intended to be used in a high-bandwidth environment, with computers connecting via TCP/IP. Its performance depends directly upon the connection speed between the cluster's computers. The usage scenario for Cluster includes a typical 100-megabit Ethernet network or the equivalent. The faster SCI protocol is also supported, but requires special hardware.

- *How many computers do I need to run a cluster, and why?*

  A minimum of three computers is required to run a viable cluster. However, the minimum **recommended** number of computers in a MySQL Cluster is four: one each to run the management and SQL nodes, and two computers to serve as storage nodes. The purpose of the two storage nodes is to

provide redundancy; the management node must run on a separate machine in order to guarantee continued arbitration services in the event that one of the storage nodes fails.

- *What do the different computers do in a cluster?*

  A MySQL Cluster has both a physical and logical organisation, with computers being the physical elements. The logical or functional elements of a cluster are referred to as **nodes**, and a computer housing a cluster node is sometimes referred to as a **cluster host**. Ideally, there will be one node per cluster host, although it is possible to run multiple nodes on a single host. There are three types of nodes, each corresponding to a specific role within the cluster. These are:

  1. **management node (MGM node)**: Provides management services for the cluster as a whole, including startup, shutdown, backups, and configuration data for the other nodes. The management node server is implemented as the application **ndb_mgmd**; the management client used to control MySQL Cluster via the MGM node is **ndb_mgm**.

  2. **storage node (data node)**: Stores and replicates data. Storage node functionality is handled by an instance of the NDB storage node process **ndbd**.

  3. **SQL node**: This is simply an instance of MySQL Server (**mysqld**) started with the **--ndb-cluster** option.

- *With which operating systems can I use Cluster?*

  As of MySQL 4.1.10, MySQL Cluster is officially supported on Linux, Mac OS X, and Solaris. We are working to add Cluster support for other platforms, including Windows (Windows support expected in MySQL 5.0), and our goal is eventually to offer Cluster on all platforms for which MySQL itself is supported.

  It may be possible to run Cluster processes on other operating systems (including Windows), but Cluster on any but the three mentioned here should be considered alpha software and not for production use.

- *What are the hardware requirements for running MySQL Cluster?*

  Cluster should run on any platform for which NDB-enabled binaries are available. Naturally, faster CPUs and more memory will improve performance, and 64-bit CPUs will likely be more effective than 32-bit processors. There must be sufficent memory on machines used for storage nodes to hold each node's share of the database (see **How much RAM does Cluster require?** for more info). Nodes can communicate via a standard TCP/IP network and hardware. For SCI support, special networking hardware is required.

- *Do I have to learn a new programming or query language to use Cluster?*

  No. While some specialised commands are used to manage and configure the cluster itself, only standard (My)SQL queries and commands are required for:

  - creating, altering, and dropping tables
  - inserting, updating, and deleting table data
  - creating, changing, and dropping primary and unique indexes
  - configuring and managing SQL nodes (MySQL servers)

- *How do I find out what an error or warning message means when using Cluster?*

  There are two ways in which this can be done:

1. From within the MySQL Monitor, use **SHOW ERRORS** or **SHOW WARNINGS** immediately upon being notified of the error or warning condition. These can also be displayed in MySQL Query Browser.

2. From a system shell prompt, use **perror --ndb** ***`error-code`***.

- *Is MySQL Cluster transaction-safe? What table types does Cluster support?*

  Yes. MySQL Cluster is enabled for tables created with the NDB storage engine, which supports transactions. NDB is the only MySQL storage engine which supports clustering.

- *What does "NDB" mean?*

  This stands for "**N**etwork **D**ata**b**ase".

- *Which version(s) of the MySQL software support Cluster? Do I have to compile from source?*

  Cluster is supported in the MySQL-max binaries from version 4.1.3 onwards. You can determine whether or not your server binary has NDB support

  Linux users, please note that NDB is **not** included in the RPMs; you should use the binaries supplied as `.tar.gz` archives in the MySQL Downloads area [http://dev.mysql.com/downloads/] instead. You can also obtain NDB support by compiling the `-max` binaries from source, but it is not necessary to do so simply to use MySQL Cluster.

- *How much RAM do I need? Is it possible to use disk memory at all?*

  Currently, Cluster is in-memory only. This means that all table data (including indexes) is stored in RAM. Therefore, if your data takes up 1 gigabyte of space and you wish to replicate it once in the cluster, you'll need 2 gigabytes of memory to do so. This in addition to the memory required by the operating system and any applications running on the cluster computers.

  You can use the following formula for obtaining a rough estimate of how much RAM is needed for each storage node in the cluster:

  ```
  (SizeofDatabase * NumberOfReplicas * 1.1 ) / NumberOfStorageNodes
  ```

  To calculate the memory requirements more exactly requires determining, for each table in the cluster database, the storage space required per row (see Section 11.5, “Column Type Storage Requirements” for details), and multiplying this by the number of rows. You must also remember to allow for any column indexes. It is especially important to keep in mind that *every* MySQL Cluster table must have a primary key, and that the NDB storage engine will create one automatically if none is defined.

  We often see questions from users who report that, when they're trying to populate a Cluster database, the loading process terminates prematurely and an error message like this one is observed:

  ```
  ERROR 1114: The table 'my_cluster_table' is full
  ```

  When this occurs, the cause is very likely to be that your setup does not provide sufficient RAM for all table data and all indexes, *including the primary key required by NDB.*

  It is also worth noting that all storage nodes should have the same amount of RAM, as no storage node in a cluster can use more memory than the least amount available to any individual storage node. In other words, if there are three computers hosting Cluster storage nodes, with two of these

having three gigabytes of RAM available to store Cluster data, and one having only one GB RAM, then each storage node can devote only one GB for Cluster.

- *In the event of a catstrophic failure - say, for instance, the whole city lost power AND my UPS failed - would I lose all my data?*

  All committed transactions are logged. Therefore, while it is possible that some data could be lost in the event of a catastrophe, this should be quite limited. Data loss can be further reduced by minimising the number of operations per transaction.

- *Is it possible to use FULLTEXT indexes with Cluster?*

  `FULLTEXT` indexing is not currently (MySQL 4.1.9) supported by the NDB storage engine. We are working to add this capability in a future release.

- *Can I run multiple nodes on a single computer?*

  It is possible but not advisable. One of the chief reasons to run a cluster is to provide redundancy; in order to enjoy the full benefits of this redundancy, each node should reside on a separate machine. If you place multiple nodes on a single machine and that machine fails, you lose all of those nodes. Given that MySQL Cluster can be run on commodity hardware loaded with a low-cost or even no-cost operating system, it is well worth the expense of an extra machine or two in order to safeguard mission-critical data. It also worth noting that the requirements for a cluster host running a management node are minimal; this task can be accomplished with a 200 MHz Pentium CPU and sufficient RAM for the operating system plus a small amount of overhead for the **ndb_mgmd** and **ndb_mgm** processes.

- *Can I add nodes to a cluster without restarting it?*

  Not at present. A simple restart is all that is required for adding new MGM or SQL nodes to a Cluster. When adding storage nodes the process is more complex and requires the following steps:

  - Making a complete backup of all Cluster data
  - Complete shutting down the cluster and all cluster node processes
  - Restarting the cluster, using the **--initial** startup option
  - Restoring all cluster data from the backup

  In future, we hope to implement "hot" reconfiguration capability for MySQL Cluster in order to minimize if not eliminate requirements for restarting the cluster when adding new nodes.

- *Are there any limitations that I should be aware of when using Cluster?*

  NDB tables in MySQL 4.1 are subject to the following limitations:

  - Not all character sets and collations are supported. (For a complete listing of those that are supported, see Section D.10.5, “MySQL Cluster-4.1.6 (10 Oct 2004)”).
  - `FULLTEXT` indexes and prefix indexes are not supported. Only complete columns may be indexed.
  - Chapter 18, *Spatial Extensions in MySQL* are not supported.
  - Only complete rollbacks for transactions are supported. Partial rollbacks and rollbacks to save points are not supported.

- The maximum number of attributes allowed per table is 128, and attribute names cannot be any longer than 31 characters. For each table, the maximum combined length of the table and database names is 122 characters.
- The maximum size for a table row is 8 kilobytes, not counting BLOBs. There is no set limit for the number of rows per table; table size limits depend on a number of factors, in particular on the amount of RAM available to each data node.
- The NDB engine does not support foreign key constraints. As with MyISAM tables, these are ignored.
- Query caching is not supported.

We expect to lift many of these restrictions in MySQL 5.0. For additional information on current limitations, see Section 16.9, “Cluster Limitations in MySQL 4.1”.

- *How do I import an existing MySQL database into a cluster?*

  You can import databases into MySQL Cluster much as you would with any other version of MySQL. Other than the limitation mentioned in the previous question, the only other special requirement is that any tables to be included in the cluster must use the NDB storage engine. This means that the tables must be created with the option **ENGINE=NDB** or **ENGINE=NDBCLUSTER**.

- *How do cluster nodes communicate with one another?*

  Cluster nodes can communicate via any of three different protocols: TCP/IP, SHM (shared memory), and SCI (Scalable Coherent Interface). Where available, SHM is used by default between nodes residing on the same cluster host. SCI is a high-speed (1 gigabit per second and higher), high-availability protocol used in building scalable multi-processor systems; it requires special hardware and drivers. See Section 16.8, “Using High-Speed Interconnects with MySQL Cluster” for more about using SCI as a transport mechanism in MySQL Cluster.

- *What is an arbitrator?*

  If one or more nodes in a cluster fail, it is possible that not all cluster nodes will not be able to "see" one another. In fact, it is possible that two sets of nodes might become isolated from one another in a network partitioning, also known as a "split brain" scenario. This type of situation is undesirable because each set of nodes tries to behave as though it is the entire cluster.

  When cluster nodes go down, there are two possibilities. If more than 50% of the remaining nodes can communicate with each other, then we have what is sometimes called a "majority rules" situation, and this set of nodes is considered to be the cluster. The arbitrator comes into play when there is an even number of nodes: in such a case, the set of nodes to which the arbitrator belongs is considered to be the cluster, and nodes not belonging to this set are shut down.

  The above is somewhat simplified; a more complete explanation taking into account node groups follows below:

  When all nodes in at least one node group are alive, network partitioning is not an issue, because no one portion of the cluster can form a functional cluster. The real problem arises when no single node group has all its nodes alive, in which case network partitioning (the "split-brain" scenario) becomes possible. Then an arbitrator is required. All cluster nodes recognise the same node as the arbitrator, which is normally the management server; however, it is possible to configure any of the MySQL Servers in the cluster to act as the arbirtrator instead. The arbitrator accepts the first set of cluster nodes to contact it, and tells the remaining set to die. Arbitrator selection is controlled by the `ArbitrationRank` configuration parameter for MySQL Server and management server nodes. (See Section 16.5.4.4, “Defining the MySQL Cluster Management Server” for details.) It should also be noted that the role of arbitrator does not in and of itself impose any heavy demands upon the host so

designated, and thus the artitrator host does not need to be particularly fast or to have extra memory especially for this purpose.

- *What column types are supported by MySQL Cluster?*

  MySQL Cluster supports all of the usual MySQL column types, with the exception of those associated with MySQL's Chapter 18, *Spatial Extensions in MySQL*. In addition, there are some differences with regard to indexes when used with NDB tables. **Note**: In MySQL 4.1and 5.0.3, Cluster tables (that is, tables created with `ENGINE=NDBCLUSTER`) have only fixed-width rows. This means that (for example) each record containing a `VARCHAR(255)` column will will require 256 bytes of storage for that column, regardless of the size of the data stored therein. This issue is expected to be fixed in MySQL 5.1.

  See Section 16.9, “Cluster Limitations in MySQL 4.1” for more information about these issues.

- *How do I start and stop MySQL Cluster?*

  It is necessary to start each node in the cluster separately, in the following order:

  1. Start the management node with the **ndb_mgmd** command.
  2. Start each storage node with the **ndbd** command.
  3. Start each MySQL server (SQL node) using **mysqld_safe --user=mysql &**.

  Each of these commands must be run from a shell on the machine housing the affected node. You can verify the the cluster is running by starting the MGM management client **ndb_mgm** on the machine housing the MGM node.

- *What happens to cluster data when the cluster is shut down?*

  The data held in memory by the cluster's storage nodes is written to disk, and is reloaded in memory the next time that the cluster is started.

  To shut down the cluster, enter the following in a shell on the machine hosting the MGM node:

  ```
  shell> ndb_mgm -e shutdown
  ```

  This will cause the **ndb_mgm**, **ndb_mgm**, and any **ndbd** processes to terminate gracefully. MySQL servers running as Cluster SQL nodes can be stopped using **mysqladmin shutdown**.

  For more information, see Section 16.7.1, “Commands in the Management Client” and Section 16.4.6, “Safe Shutdown and Restart”.

- *Is it helpful to have more than one management node for a cluster?*

  It can be helpful as a fail-safe. Only one MGM node controls the cluster at any given time, but it is possible to configure one MGM as primary, and one or more additional management nodes to take over in the evnt that the primary MGM node fails.

- *Can I mix different kinds of hardware and operating systems in a Cluster?*

  Yes, so long as all machines and operating systems are the same endian. It is also possible to use different MySQL Cluster releases on different nodes (for example, 4.1.8 on some nodes and 4.1.9 on others); however, we recommend this be done only as part of a rolling upgrade procedure.

- *Can I run two storage nodes on a single host? Two SQL nodes?*

  Yes, it is possible to do this. In the case of multiple storage nodes, each node must use a different

data directory. If you want to run multiple SQL nodes on one machine, then each instance of **mysqld** must use a different TCP/IP port.

- *Can I use hostnames with MySQL Cluster?*

  Yes, it's possible to use DNS and DHCP for cluster hosts. However, if your application requires "five nines" availability, we recommend using fixed IP addresses. This is because making communication between Cluster hosts dependent on such services introduces additional points of failure, and the fewer of these, the better.

Chapter 17. Introduction to MaxDB

MaxDB is an enterprise-level database. MaxDB is the new name of a database management system formerly called SAP DB.

17.1. History of MaxDB

The history of SAP DB goes back to the early 1980s when it was developed as a commercial product (Adabas). The database has changed names several times since then. When SAP AG, a company based in Walldorf, Germany, took over the development of that database system, it was called SAP DB.

SAP developed that database system to serve as a storage system for all heavy-duty SAP applications, namely R/3. SAP DB was meant to provide an alternative to third-party database systems such as Oracle, Microsoft SQL Server, and DB2 by IBM. In October 2000, SAP AG released SAP DB under the GNU GPL license (see Appendix H, *GNU General Public License*), thus making it Open Source software. In October 2003, more than 2,000 customers of SAP AG were using SAP DB as their main database system, and more than another 2,000 customers were using it as a separate database system besides their main database, as part of the APO/LiveCache solution.

In May 2003, a technology partnership was formed between MySQL AB and SAP AG. That partnership entitles MySQL AB to further develop SAP DB, rename it, and sell commercial licenses of the renamed SAP DB to customers who do not want to be bound to the restrictions imposed on them when using that database system under the GNU GPL (see Appendix H, *GNU General Public License*). In August 2003, SAP DB was renamed MaxDB by MySQL AB.

17.2. Licensing and Support

MaxDB can be used under the same licenses available for the other products distributed by MySQL AB. Thus, MaxDB is available under the GNU General Public License, and a commercial license. For more information on licensing, see http://www.mysql.com/company/legal/licensing/.

MySQL offers MaxDB support to non-SAP customers.

The first rebranded version was MaxDB 7.5.00, which was released in November 2003.

17.3. MaxDB-Related Links

The main page for information about MaxDB is http://www.mysql.com/products/maxdb. Information formerly available at http://www.sapdb.org has been moved there.

17.4. Basic Concepts of MaxDB

MaxDB operates as a client/server product. It was developed to meet the demands of installations processing a high volume of online transactions. Both online backup and expansion of the database are supported. Microsoft Clustered Server is supported directly for multiple server implementations; other failover solutions must be scripted manually. Database management tools are provided in both Windows and browser-based implementations.

17.5. Feature Differences Between MaxDB and MySQL

The following list provides a short summary of the main differences between MaxDB and MySQL; it is not complete.

- MaxDB runs as a client/server system. MySQL can run as a client/server system or as an embedded system.
- MaxDB might not run on all platforms supported by MySQL. For example, MaxDB does not run on IBM's OS/2.
- MaxDB uses a proprietary network protocol for client/server communication. MySQL uses either TCP/IP (with or without SSL encryption), sockets (under Unix-like systems), or named pipes (under Windows NT-family systems).
- MaxDB supports stored procedures. For MySQL, stored procedures are implemented in version 5.0. MaxDB also supports programming of triggers through an SQL extension, which is scheduled for MySQL 5.1. MaxDB contains a debugger for stored procedure languages, can cascade nested triggers, and supports multiple triggers per action and row.
- MaxDB is distributed with user interfaces that are text-based, graphical, or Web-based. MySQL is distributed with text-based user interfaces only; graphical user interface (MySQL Control Center, MySQL Administrator) are shipped separately from the main distributions. Web-based user interfaces for MySQL are offered by third parties.
- MaxDB supports a number of programming interfaces that also are supported by MySQL. However, MaxDB does not support RDO, ADO, or .NET, all of which are supported by MySQL. MaxDB supports embedded SQL only with C/C++.
- MaxDB includes administrative features that MySQL does not have: job scheduling by time, event, and alert, and sending messages to a database administrator on alert thresholds.

17.6. Interoperability Features Between MaxDB and MySQL

As part of MaxDB 7.6, the MaxDB Synchronization Manager is released. The Synchronization Manager supports creation of asynchronous replication scenarios between several MaxDB instances. However, interoperability features also are planned, so that the Synchronization Manager supports replication to and from a MySQL server.

In the first release, the Synchronization Manager supports inserting data into MySQL. This means that initially only replication from MaxDB to MySQL is supported. In the course of 2005, exporting of data from a MySQL server to the Synchronization Manager will be added, thus adding support for MySQL to MaxDB replication scenarios.

MaxDB 7.6, with the Synchronization Manager, was released as a beta version in January 2005. The production release is planned for April 2005.

17.7. Reserved Words in MaxDB

Like MySQL, MaxDB has a number of reserved words that have special meanings. Normally, they cannot be used as names of identifiers, such as database or table names. The following table lists reserved words in MaxDB, indicates the context in which those words are used, and indicates whether or not they have counterparts in MySQL. If such a counterpart exists, the meaning in MySQL might be identical or differing in some aspects. The main purpose is to list in which respects MaxDB differs from MySQL;

therefore, this list is not complete.

For the list of reserved words in MySQL, see Section 9.6, “Treatment of Reserved Words in MySQL”.

Reserved in MaxDB	Context of usage in MaxDB	MySQL counterpart
@	Can prefix identifier, like ``@table"	Not allowed
ADDDATE()	SQL function	ADDDATE(); new in MySQL 4.1.1
ADDTIME()	SQL function	ADDTIME(); new in MySQL 4.1.1
ALPHA	SQL function	Nothing comparable
ARRAY	Data type	Not implemented
ASCII()	SQL function	ASCII(), but implemented with a different meaning
AUTOCOMMIT	Transactions; ON by default	Transactions; OFF by default
BOOLEAN	Column types; BOOLEAN accepts as values only TRUE, FALSE, and NULL	BOOLEAN was added in MySQL 4.1.0; it is a synonym for BOOL which is mapped to TINYINT(1). It accepts integer values in the same range as TINYINT as well as NULL. TRUE and FALSE can be used as aliases for 1 and 0.
CHECK	CHECK TABLE	CHECK TABLE; similar, but not identical usage
COLUMN	Column types	COLUMN; noise word
CHAR()	SQL function	CHAR(); identical syntax; similar, not identical usage
COMMIT	Implicit commits of transactions happen when data definition statements are issued	Implicit commits of transactions happen when data definition statements are issued, and also with a number of other statements
COSH()	SQL function	Nothing comparable
COT()	SQL function	COT(); identical syntax and implementation
CREATE	SQL, data definition language	CREATE
DATABASE	SQL function	DATABASE(); DATABASE is used in a different context; for example, CREATE DATABASE
DATE()	SQL function	CURRENT_DATE
DATEDIFF()	SQL function	DATEDIFF(); new in MySQL 4.1.1
DAY()	SQL function	Nothing comparable
DAYOFWEEK()	SQL function	DAYOFWEEK(); by default, 1 represents Monday in MaxDB and Sunday in MySQL
DISTINCT	SQL functions AVG, MAX, MIN, SUM	DISTINCT; but used in a different context: SELECT DISTINCT
DROP	DROP INDEX, for example	DROP INDEX; similar, but not identical usage
EBCDIC()	SQL function	Nothing comparable
EXPAND()	SQL function	Nothing comparable
EXPLAIN	Optimization	EXPLAIN; similar, but not identical usage
FIXED()	SQL function	Nothing comparable
FLOAT()	SQL function	Nothing comparable
HEX()	SQL function	HEX(); similar, but not identical usage

INDEX()	SQL function	INSTR() or LOCATE(); similar, but not identical syntaxes and meanings
INDEX	USE INDEX, IGNORE INDEX and similar hints are used right after SELECT; for example, SELECT ... USE INDEX	USE INDEX, IGNORE INDEX and similar hints are used in the FROM clause of a SELECT query; for example, in SELECT ... FROM ... USE INDEX
INITCAP()	SQL function	Nothing comparable
LENGTH()	SQL function	LENGTH(); identical syntax, but slightly different implementation
LFILL()	SQL function	Nothing comparable
LIKE	Comparisons	LIKE; but the extended LIKE MaxDB provides rather resembles the MySQL REGEX
LIKE wildcards	MaxDB supports ``%", ``_", ``Control-underline", ``Control-up arrow", ``*", and ``?" as wildcards in LIKE comparisons	MySQL supports ``%", and ``_" as wildcards in LIKE comparisons
LPAD()	SQL function	LPAD(); slightly different implementation
LTRIM()	SQL function	LTRIM(); slightly different implementation
MAKEDATE()	SQL function	MAKEDATE(); new in MySQL 4.1.1
MAKETIME()	SQL function	MAKETIME(); new in MySQL 4.1.1
MAPCHAR()	SQL function	Nothing comparable
MICROSECOND()	SQL function	MICROSECOND(); new in MySQL 4.1.1
NOROUND()	SQL function	Nothing comparable
NULL	Column types; comparisons	NULL; MaxDB supports special NULL values that are returned by arithmetic operations that lead to an overflow or a division by zero; MySQL does not support such special values
PI	SQL function	PI(); identical syntax and implementation, but parentheses are mandatory in MySQL
REF	Data type	Nothing comparable
RFILL()	SQL function	Nothing comparable
ROWNO	Predicate in WHERE clause	Similar to LIMIT clause
RPAD()	SQL function	RPAD(); slightly different implementation
RTRIM()	SQL function	RTRIM(); slightly different implementation
SEQUENCE	CREATE SEQUENCE, DROP SEQUENCE	AUTO_INCREMENT; similar concept, but different implementation
SINH()	SQL function	Nothing comparable
SOUNDS()	SQL function	SOUNDEX(); slightly different syntax
STATISTICS	UPDATE STATISTICS	ANALYZE TABLE; similar concept, but different implementation
SUBSTR()	SQL function	SUBSTRING(); slightly different implementation
SUBTIME()	SQL function	SUBTIME(); new in MySQL 4.1.1
SYNONYM	Data definition language: CREATE [PUBLIC] SYNONYM, RENAME SYNONYM,	Nothing comparable

	DROP SYNONYM	
TANH()	SQL function	Nothing comparable
TIME()	SQL function	CURRENT_TIME
TIMEDIFF()	SQL function	TIMEDIFF(); new in MySQL 4.1.1
TIMESTAMP()	SQL function	TIMESTAMP(); new in MySQL 4.1.1
TIMESTAMP() as argument to DAYOFMONTH() and DAYOFYEAR()	SQL function	Nothing comparable
TIMEZONE()	SQL function	Nothing comparable
TRANSACTION()	Returns the ID of the current transaction	Nothing comparable
TRANSLATE()	SQL function	REPLACE(); identical syntax and implementation
TRIM()	SQL function	TRIM(); slightly different implementation
TRUNC()	SQL function	TRUNCATE(); slightly different syntax and implementation
USE	Switches to a new database instance; terminates the connection to the current database instance; all subsequent commands are referred to this database instance	USE; identical syntax, but does not terminate the connection to the current database
USER	SQL function	USER(); identical syntax, but slightly different implementation, and parentheses are mandatory in MySQL
UTC_DIFF()	SQL function	UTC_DATE(); provides a means to calculate the same result as UTC_DIFF()
VALUE()	SQL function, alias for COALESCE()	COALESCE(); identical syntax and implementation
VARIANCE()	SQL function	VARIANCE(); new in MySQL 4.1.0
WEEKOFYEAR()	SQL function	WEEKOFYEAR(); new in MySQL 4.1.1

Chapter 18. Spatial Extensions in MySQL

MySQL 4.1 introduces spatial extensions to allow the generation, storage, and analysis of geographic features. Currently, these features are available for `MyISAM` tables only.

This chapter covers the following topics:

- The basis of these spatial extensions in the OpenGIS geometry model
- Data formats for representing spatial data
- How to use spatial data in MySQL
- Use of indexing for spatial data
- MySQL differences from the OpenGIS specification

18.1. Introduction

MySQL implements spatial extensions following the specification of the `Open GIS Consortium` (OGC). This is an international consortium of more than 250 companies, agencies, and universities participating in the development of publicly available conceptual solutions that can be useful with all kinds of applications that manage spatial data. The OGC maintains a Web site at http://www.opengis.org/.

In 1997, the Open GIS Consortium published the *OpenGIS (R) Simple Features Specifications For SQL*, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. This specification is available from the Open GIS Web site at http://www.opengis.org/docs/99-049.pdf. It contains additional information relevant to this chapter.

MySQL implements a subset of the **SQL with Geometry Types** environment proposed by OGC. This term refers to an SQL environment that has been extended with a set of geometry types. A geometry-valued SQL column is implemented as a column that has a geometry type. The specifications describe a set of SQL geometry types, as well as functions on those types to create and analyze geometry values.

A **geographic feature** is anything in the world that has a location. A feature can be:

- An entity. For example, a mountain, a pond, a city.
- A space. For example, a postcode area, the tropics.
- A definable location. For example, a crossroad, as a particular place where two streets intersect.

You can also find documents that use the term **geospatial feature** to refer to geographic features.

Geometry is another word that denotes a geographic feature. Originally the word **geometry** meant measurement of the earth. Another meaning comes from cartography, referring to the geometric features that cartographers use to map the world.

This chapter uses all of these terms synonymously: **geographic feature**, **geospatial feature**, **feature**, or **geometry**. The term most commonly used here is **geometry**.

Let's define a **geometry** as *a point or an aggregate of points representing anything in the world that has a location*.

18.2. The OpenGIS Geometry Model

The set of geometry types proposed by OGC's **SQL with Geometry Types** environment is based on the **OpenGIS Geometry Model**. In this model, each geometric object has the following general properties:

- It is associated with a Spatial Reference System, which describes the coordinate space in which the object is defined.
- It belongs to some geometry class.

18.2.1. The Geometry Class Hierarchy

The geometry classes define a hierarchy as follows:

- `Geometry` (non-instantiable)
  - `Point` (instantiable)
  - `Curve` (non-instantiable)
    - `LineString` (instantiable)
      - `Line`
      - `LinearRing`
  - `Surface` (non-instantiable)
    - `Polygon` (instantiable)
  - `GeometryCollection` (instantiable)
    - `MultiPoint` (instantiable)
    - `MultiCurve` (non-instantiable)
      - `MultiLineString` (instantiable)
    - `MultiSurface` (non-instantiable)
      - `MultiPolygon` (instantiable)

It is not possible to create objects in non-instantiable classes. It is possible to create objects in instantiable classes. All classes have properties, and instantiable classes may also have assertions (rules that define valid class instances).

`Geometry` is the base class. It's an abstract class. The instantiable subclasses of `Geometry` are restricted to zero-, one-, and two-dimensional geometric objects that exist in two-dimensional coordinate space. All instantiable geometry classes are defined so that valid instances of a geometry class are topologically closed (that is, all defined geometries include their boundary).

The base `Geometry` class has subclasses for `Point`, `Curve`, `Surface`, and `GeometryCollection`:

- `Point` represents zero-dimensional objects.
- `Curve` represents one-dimensional objects, and has subclass `LineString`, with sub-subclasses `Line` and `LinearRing`.
- `Surface` is designed for two-dimensional objects and has subclass `Polygon`.
- `GeometryCollection` has specialized zero-, one-, and two-dimensional collection classes named `MultiPoint`, `MultiLineString`, and `MultiPolygon` for modeling geometries corresponding to collections of `Points`, `LineStrings`, and `Polygons`, respectively. `MultiCurve` and `MultiSurface` are introduced as abstract superclasses that generalize the collection interfaces to handle `Curves` and `Surfaces`.

`Geometry`, `Curve`, `Surface`, `MultiCurve`, and `MultiSurface` are defined as non-instantiable classes. They define a common set of methods for their subclasses and are included for extensibility.

`Point`, `LineString`, `Polygon`, `GeometryCollection`, `MultiPoint`, `MultiLineString`, and `MultiPolygon` are instantiable classes.

18.2.2. Class `Geometry`

`Geometry` is the root class of the hierarchy. It is a non-instantiable class but has a number of properties that are common to all geometry values created from any of the `Geometry` subclasses. These properties are described in the following list. (Particular subclasses have their own specific properties, described later.)

Geometry Properties

A geometry value has the following properties:

- Its **type**. Each geometry belongs to one of the instantiable classes in the hierarchy.
- Its **SRID**, or Spatial Reference Identifier. This value identifies the geometry's associated Spatial Reference System that describes the coordinate space in which the geometry object is defined.

  In MySQL, the SRID value is just an integer associated with the geometry value. All calculations are done assuming Euclidean (planar) geometry.
- Its **coordinates** in its Spatial Reference System, represented as double-precision (eight-byte) numbers. All non-empty geometries include at least one pair of (X,Y) coordinates. Empty geometries contain no coordinates.

  Coordinates are related to the SRID. For example, in different coordinate systems, the distance between two objects may differ even when objects have the same coordinates, because the distance on the **planar** coordinate system and the distance on the **geocentric** system (coordinates on the Earth's surface) are different things.
- Its **interior**, **boundary**, and **exterior**.

  Every geometry occupies some position in space. The exterior of a geometry is all space not occupied by the geometry. The interior is the space occupied by the geometry. The boundary is the interface between the geometry's interior and exterior.

- Its **MBR** (Minimum Bounding Rectangle), or Envelope. This is the bounding geometry, formed by the minimum and maximum (X,Y) coordinates:

  ```
  ((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
  ```

- Whether the value is **simple** or **non-simple**. Geometry values of types (`LineString`, `MultiPoint`, `MultiLineString`) are either simple or non-simple. Each type determines its own assertions for being simple or non-simple.

- Whether the value is **closed** or **not closed**. Geometry values of types (`LineString`, `MultiString`) are either closed or not closed. Each type determines its own assertions for being closed or not closed.

- Whether the value is **empty** or **non-empty** A geometry is empty if it does not have any points. Exterior, interior, and boundary of an empty geometry are not defined (that is, they are represented by a `NULL` value). An empty geometry is defined to be always simple and has an area of 0.

- Its **dimension**. A geometry can have a dimension of #1, 0, 1, or 2:

  - #1 for an empty geometry.
  - 0 for a geometry with no length and no area.
  - 1 for a geometry with non-zero length and zero area.
  - 2 for a geometry with non-zero area.

  `Point` objects have a dimension of zero. `LineString` objects have a dimension of 1. `Polygon` objects have a dimension of 2. The dimensions of `MultiPoint`, `MultiLineString`, and `MultiPolygon` objects are the same as the dimensions of the elements they consist of.

18.2.3. Class `Point`

A `Point` is a geometry that represents a single location in coordinate space.

`Point` Examples

- Imagine a large-scale map of the world with many cities. A `Point` object could represent each city.
- On a city map, a `Point` object could represent a bus stop.

`Point` Properties

- X-coordinate value.
- Y-coordinate value.
- `Point` is defined as a zero-dimensional geometry.
- The boundary of a `Point` is the empty set.

18.2.4. Class `Curve`

A `Curve` is a one-dimensional geometry, usually represented by a sequence of points. Particular subclasses of `Curve` define the type of interpolation between points. `Curve` is a non-instantiable class.

`Curve` Properties

- A `Curve` has the coordinates of its points.
- A `Curve` is defined as a one-dimensional geometry.
- A `Curve` is simple if it does not pass through the same point twice.
- A `Curve` is closed if its start point is equal to its end point.
- The boundary of a closed `Curve` is empty.
- The boundary of a non-closed `Curve` consists of its two end points.
- A `Curve` that is simple and closed is a `LinearRing`.

18.2.5. Class `LineString`

A `LineString` is a `Curve` with linear interpolation between points.

`LineString` Examples

- On a world map, `LineString` objects could represent rivers.
- In a city map, `LineString` objects could represent streets.

`LineString` Properties

- A `LineString` has coordinates of segments, defined by each consecutive pair of points.
- A `LineString` is a `Line` if it consists of exactly two points.
- A `LineString` is a `LinearRing` if it is both closed and simple.

18.2.6. Class `Surface`

A `Surface` is a two-dimensional geometry. It is a non-instantiable class. Its only instantiable subclass is `Polygon`.

`Surface` Properties

- A `Surface` is defined as a two-dimensional geometry.
- The OpenGIS specification defines a simple `Surface` as a geometry that consists of a single ``patch'' that is associated with a single exterior boundary and zero or more interior boundaries.
- The boundary of a simple `Surface` is the set of closed curves corresponding to its exterior and interior boundaries.

18.2.7. Class Polygon

A Polygon is a planar Surface representing a multisided geometry. It is defined by a single exterior boundary and zero or more interior boundaries, where each interior boundary defines a hole in the Polygon.

Polygon Examples

- On a region map, Polygon objects could represent forests, districts, an so on.

Polygon Assertions

- The boundary of a Polygon consists of a set of LinearRing objects (that is, LineString objects that are both simple and closed) that make up its exterior and interior boundaries.
- A Polygon has no rings that cross. The rings in the boundary of a Polygon may intersect at a Point, but only as a tangent.
- A Polygon has no lines, spikes, or punctures.
- A Polygon has an interior that is a connected point set.
- A Polygon may have holes. The exterior of a Polygon with holes is not connected. Each hole defines a connected component of the exterior.

The preceding assertions make a Polygon a simple geometry.

18.2.8. Class GeometryCollection

A GeometryCollection is a geometry that is a collection of one or more geometries of any class.

All the elements in a GeometryCollection must be in the same Spatial Reference System (that is, in the same coordinate system). There are no other constraints on the elements of a GeometryCollection, although the subclasses of GeometryCollection described in the following sections may restrict membership. Restrictions may be based on:

- Element type (for example, a MultiPoint may contain only Point elements)
- Dimension
- Constraints on the degree of spatial overlap between elements

18.2.9. Class MultiPoint

A MultiPoint is a geometry collection composed of Point elements. The points are not connected or ordered in any way.

MultiPoint Examples

- On a world map, a MultiPoint could represent a chain of small islands.
- On a city map, a MultiPoint could represent the outlets for a ticket office.

`MultiPoint` Properties

- A `MultiPoint` is a zero-dimensional geometry.
- A `MultiPoint` is simple if no two of its `Point` values are equal (have identical coordinate values).
- The boundary of a `MultiPoint` is the empty set.

18.2.10. Class `MultiCurve`

A `MultiCurve` is a geometry collection composed of `Curve` elements. `MultiCurve` is a non-instantiable class.

`MultiCurve` Properties

- A `MultiCurve` is a one-dimensional geometry.
- A `MultiCurve` is simple if and only if all of its elements are simple; the only intersections between any two elements occur at points that are on the boundaries of both elements.
- A `MultiCurve` boundary is obtained by applying the ``mod 2 union rule'' (also known as the ``odd-even rule''): A point is in the boundary of a `MultiCurve` if it is in the boundaries of an odd number of `MultiCurve` elements.
- A `MultiCurve` is closed if all of its elements are closed.
- The boundary of a closed `MultiCurve` is always empty.

18.2.11. Class `MultiLineString`

A `MultiLineString` is a `MultiCurve` geometry collection composed of `LineString` elements.

`MultiLineString` Examples

- On a region map, a `MultiLineString` could represent a river system or a highway system.

18.2.12. Class `MultiSurface`

A `MultiSurface` is a geometry collection composed of surface elements. `MultiSurface` is a non-instantiable class. Its only instantiable subclass is `MultiPolygon`.

`MultiSurface` Assertions

- Two `MultiSurface` surfaces have no interiors that intersect.
- Two `MultiSurface` elements have boundaries that intersect at most at a finite number of points.

18.2.13. Class `MultiPolygon`

A `MultiPolygon` is a `MultiSurface` object composed of `Polygon` elements.

`MultiPolygon` Examples

- On a region map, a `MultiPolygon` could represent a system of lakes.

`MultiPolygon` Assertions

- A `MultiPolygon` has no two `Polygon` elements with interiors that intersect.
- A `MultiPolygon` has no two `Polygon` elements that cross (crossing is also forbidden by the previous assertion), or that touch at an infinite number of points.
- A `MultiPolygon` may not have cut lines, spikes, or punctures. A `MultiPolygon` is a regular, closed point set.
- A `MultiPolygon` that has more than one `Polygon` has an interior that is not connected. The number of connected components of the interior of a `MultiPolygon` is equal to the number of `Polygon` values in the `MultiPolygon`.

`MultiPolygon` Properties

- A `MultiPolygon` is a two-dimensional geometry.
- A `MultiPolygon` boundary is a set of closed curves (`LineString` values) corresponding to the boundaries of its `Polygon` elements.
- Each `Curve` in the boundary of the `MultiPolygon` is in the boundary of exactly one `Polygon` element.
- Every `Curve` in the boundary of an `Polygon` element is in the boundary of the `MultiPolygon`.

18.3. Supported Spatial Data Formats

This section describes the standard spatial data formats that are used to represent geometry objects in queries. They are:

- Well-Known Text (WKT) format
- Well-Known Binary (WKB) format

Internally, MySQL stores geometry values in a format that is not identical to either WKT or WKB format.

18.3.1. Well-Known Text (WKT) Format

The Well-Known Text (WKT) representation of Geometry is designed to exchange geometry data in ASCII form.

Examples of WKT representations of geometry objects are:

- A `Point`:

  ```
  POINT(15 20)
  ```

  Note that point coordinates are specified with no separating comma.

- A `LineString` with four points:

  ```
  LINESTRING(0 0, 10 10, 20 25, 50 60)
  ```

  Note that point coordinate pairs are separated by commas.

- A `Polygon` with one exterior ring and one interior ring:

  ```
  POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))
  ```

- A `MultiPoint` with three `Point` values:

  ```
  MULTIPOINT(0 0, 20 20, 60 60)
  ```

- A `MultiLineString` with two `LineString` values:

  ```
  MULTILINESTRING((10 10, 20 20), (15 15, 30 15))
  ```

- A `MultiPolygon` with two `Polygon` values:

  ```
  MULTIPOLYGON(((0 0,10 0,10 10,0 10,0 0)),((5 5,7 5,7 7,5 7, 5 5)))
  ```

- A `GeometryCollection` consisting of two `Point` values and one `LineString`:

  ```
  GEOMETRYCOLLECTION(POINT(10 10), POINT(30 30), LINESTRING(15 15, 20 20))
  ```

A Backus-Naur grammar that specifies the formal production rules for writing WKT values can be found in the OGC specification document referenced near the beginning of this chapter.

18.3.2. Well-Known Binary (WKB) Format

The Well-Known Binary (WKB) representation for geometric values is defined by the OpenGIS specifications. It is also defined in the ISO ``SQL/MM Part 3: Spatial'' standard.

WKB is used to exchange geometry data as binary streams represented by `BLOB` values containing geometric WKB information.

WKB uses one-byte unsigned integers, four-byte unsigned integers, and eight-byte double-precision numbers (IEEE 754 format). A byte is eight bits.

For example, a WKB value that corresponds to `POINT(1 1)` consists of this sequence of 21 bytes (each represented here by two hex digits):

```
0101000000000000000000F03F000000000000F03F
```

The sequence may be broken down into these components:

```
Byte order : 01
WKB type   : 01000000
X          : 000000000000F03F
Y          : 000000000000F03F
```

Component representation is as follows:

- The byte order may be either 0 or 1 to indicate little-endian or big-endian storage. The little-endian and big-endian byte orders are also known as Network Data Representation (NDR) and External Data Representation (XDR), respectively.
- The WKB type is a code that indicates the geometry type. Values from 1 through 7 indicate `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon`, and `GeometryCollection`.
- A `Point` value has X and Y coordinates, each represented as a double-precision value.

WKB values for more complex geometry values are represented by more complex data structures, as detailed in the OpenGIS specification.

18.4. Creating a Spatially Enabled MySQL Database

This section describes the data types you can use for representing spatial data in MySQL, and the functions available for creating and retrieving spatial values.

18.4.1. MySQL Spatial Data Types

MySQL has data types that correspond to OpenGIS classes. Some of these types hold single geometry values:

- `GEOMETRY`
- `POINT`
- `LINESTRING`
- `POLYGON`

`GEOMETRY` can store geometry values of any type. The other single-value types, `POINT` and `LINESTRING` and `POLYGON`, restrict their values to a particular geometry type.

The other data types hold collections of values:

- `MULTIPOINT`
- `MULTILINESTRING`
- `MULTIPOLYGON`

- `GEOMETRYCOLLECTION`

`GEOMETRYCOLLECTION` can store a collection of objects of any type. The other collection types, `MULTIPOINT` and `MULTILINESTRING` and `MULTIPOLYGON` and `GEOMETRYCOLLECTION`, restrict collection members to those having a particular geometry type.

18.4.2. Creating Spatial Values

This section describes how to create spatial values using Well-Known Text and Well-Known Binary functions that are defined in the OpenGIS standard, and using MySQL-specific functions.

18.4.2.1. Creating Geometry Values Using WKT Functions

MySQL provides a number of functions that take as input parameters a Well-Known Text representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry.

`GeomFromText()` accepts a WKT of any geometry type as its first argument. An implementation also provides type-specific construction functions for construction of geometry values of each geometry type.

- `GeomCollFromText(`*`wkt`*`[,`*`srid`*`])`, `GeometryCollectionFromText(`*`wkt`*`[,`*`srid`*`])`

  Constructs a `GEOMETRYCOLLECTION` value using its WKT representation and SRID.

- `GeomFromText(`*`wkt`*`[,`*`srid`*`])`, `GeometryFromText(`*`wkt`*`[,`*`srid`*`])`

  Constructs a geometry value of any type using its WKT representation and SRID.

- `LineFromText(`*`wkt`*`[,`*`srid`*`])`, `LineStringFromText(`*`wkt`*`[,`*`srid`*`])`

  Constructs a `LINESTRING` value using its WKT representation and SRID.

- `MLineFromText(`*`wkt`*`[,`*`srid`*`])`, `MultiLineStringFromText(`*`wkt`*`[,`*`srid`*`])`

  Constructs a `MULTILINESTRING` value using its WKT representation and SRID.

- `MPointFromText(`*`wkt`*`[,`*`srid`*`])`, `MultiPointFromText(`*`wkt`*`[,`*`srid`*`])`

  Constructs a `MULTIPOINT` value using its WKT representation and SRID.

- `MPolyFromText(`*`wkt`*`[,`*`srid`*`])`, `MultiPolygonFromText(`*`wkt`*`[,`*`srid`*`])`

  Constructs a `MULTIPOLYGON` value using its WKT representation and SRID.

- `PointFromText(`*`wkt`*`[,`*`srid`*`])`

Constructs a POINT value using its WKT representation and SRID.

- PolyFromText(*wkt*[,*srid*]), PolygonFromText(*wkt*[,*srid*])

  Constructs a POLYGON value using its WKT representation and SRID.

The OpenGIS specification also describes optional functions for constructing Polygon or MultiPolygon values based on the WKT representation of a collection of rings or closed LineString values. These values may intersect. MySQL does not implement these functions:

- BdMPolyFromText(*wkt*,*srid*)

  Constructs a MultiPolygon value from a MultiLineString value in WKT format containing an arbitrary collection of closed LineString values.

- BdPolyFromText(*wkt*,*srid*)

  Constructs a Polygon value from a MultiLineString value in WKT format containing an arbitrary collection of closed LineString values.

18.4.2.2. Creating Geometry Values Using WKB Functions

MySQL provides a number of functions that take as input parameters a BLOB containing a Well-Known Binary representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry.

GeomFromWKT() accepts a WKB of any geometry type as its first argument. An implementation also provides type-specific construction functions for construction of geometry values of each geometry type.

- GeomCollFromWKB(*wkb*[,*srid*]), GeometryCollectionFromWKB(*wkt*[,*srid*])

  Constructs a GEOMETRYCOLLECTION value using its WKB representation and SRID.

- GeomFromWKB(*wkb*[,*srid*]), GeometryFromWKB(*wkt*[,*srid*])

  Constructs a geometry value of any type using its WKB representation and SRID.

- LineFromWKB(*wkb*[,*srid*]), LineStringFromWKB(*wkb*[,*srid*])

  Constructs a LINESTRING value using its WKB representation and SRID.

- MLineFromWKB(*wkb*[,*srid*]), MultiLineStringFromWKB(*wkb*[,*srid*])

  Constructs a MULTILINESTRING value using its WKB representation and SRID.

- `MPointFromWKB(wkb[,srid])`, `MultiPointFromWKB(wkb[,srid])`

  Constructs a `MULTIPOINT` value using its WKB representation and SRID.

- `MPolyFromWKB(wkb[,srid])`, `MultiPolygonFromWKB(wkb[,srid])`

  Constructs a `MULTIPOLYGON` value using its WKB representation and SRID.

- `PointFromWKB(wkb[,srid])`

  Constructs a `POINT` value using its WKB representation and SRID.

- `PolyFromWKB(wkb[,srid])`, `PolygonFromWKB(wkb[,srid])`

  Constructs a `POLYGON` value using its WKB representation and SRID.

The OpenGIS specification also describes optional functions for constructing `Polygon` or `MultiPolygon` values based on the WKB representation of a collection of rings or closed `LineString` values. These values may intersect. MySQL does not implement these functions:

- `BdMPolyFromWKB(wkb,srid)`

  Constructs a `MultiPolygon` value from a `MultiLineString` value in WKB format containing an arbitrary collection of closed `LineString` values.

- `BdPolyFromWKB(wkb,srid)`

  Constructs a `Polygon` value from a `MultiLineString` value in WKB format containing an arbitrary collection of closed `LineString` values.

18.4.2.3. Creating Geometry Values Using MySQL-Specific Functions

Note: MySQL does not implement the functions listed in this section.

MySQL provides a set of useful functions for creating geometry WKB representations. The functions described in this section are MySQL extensions to the OpenGIS specifications. The results of these functions are `BLOB` values containing WKB representations of geometry values with no SRID. The results of these functions can be substituted as the first argument for any function in the `GeomFromWKB()` function family.

- `GeometryCollection(g1,g2,...)`

  Constructs a WKB `GeometryCollection`. If any argument is not a well-formed WKB representation of a geometry, the return value is `NULL`.

- `LineString(pt1,pt2,...)`

  Constructs a WKB `LineString` value from a number of WKB `Point` arguments. If any argument is not a WKB `Point`, the return value is `NULL`. If the number of `Point` arguments is less than two, the return value is `NULL`.

- `MultiLineString(ls1,ls2,...)`

  Constructs a WKB `MultiLineString` value using WKB `LineString` arguments. If any argument is not a WKB `LineString`, the return value is `NULL`.

- `MultiPoint(pt1,pt2,...)`

  Constructs a WKB `MultiPoint` value using WKB `Point` arguments. If any argument is not a WKB `Point`, the return value is `NULL`.

- `MultiPolygon(poly1,poly2,...)`

  Constructs a WKB `MultiPolygon` value from a set of WKB `Polygon` arguments. If any argument is not a WKB `Polygon`, the return value is `NULL`.

- `Point(x,y)`

  Constructs a WKB `Point` using its coordinates.

- `Polygon(ls1,ls2,...)`

  Constructs a WKB `Polygon` value from a number of WKB `LineString` arguments. If any argument does not represent the WKB of a `LinearRing` (that is, not a closed and simple `LineString`) the return value is `NULL`.

18.4.3. Creating Spatial Columns

MySQL provides a standard way of creating spatial columns for geometry types, for example, with `CREATE TABLE` or `ALTER TABLE`. Currently, spatial columns are supported only for `MyISAM` tables.

- Use the `CREATE TABLE` statement to create a table with a spatial column:

  ```
  mysql> CREATE TABLE geom (g GEOMETRY);
  Query OK, 0 rows affected (0.02 sec)
  ```

- Use the `ALTER TABLE` statement to add or drop a spatial column to or from an existing table:

```
mysql> ALTER TABLE geom ADD pt POINT;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql> ALTER TABLE geom DROP pt;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

18.4.4. Populating Spatial Columns

After you have created spatial columns, you can populate them with spatial data.

Values should be stored in internal geometry format, but you can convert them to that format from either Well-Known Text (WKT) or Well-Known Binary (WKB) format. The following examples demonstrate how to insert geometry values into a table by converting WKT values into internal geometry format.

You can perform the conversion directly in the `INSERT` statement:

```
INSERT INTO geom VALUES (GeomFromText('POINT(1 1)'));

SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (GeomFromText(@g));
```

Or you can perform the conversion prior to the `INSERT`:

```
SET @g = GeomFromText('POINT(1 1)');
INSERT INTO geom VALUES (@g);
```

The following examples insert more complex geometries into the table:

```
SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (GeomFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (GeomFromText(@g));

SET @g =
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (GeomFromText(@g));
```

The preceding examples all use `GeomFromText()` to create geometry values. You can also use type-specific functions:

```
SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (PointFromText(@g));

SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (LineStringFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (PolygonFromText(@g));

SET @g =
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (GeomCollFromText(@g));
```

Note that if a client application program wants to use WKB representations of geometry values, it is responsible for sending correctly formed WKB in queries to the server. However, there are several ways of satisfying this requirement. For example:

- Inserting a `POINT(1 1)` value with hex literal syntax:

  ```
  mysql> INSERT INTO geom VALUES
      -> (GeomFromWKB(0x0101000000000000000000F03F000000000000F03F));
  ```

- An ODBC application can send a WKB representation, binding it to a placeholder using an argument of `BLOB` type:

  ```
  INSERT INTO geom VALUES (GeomFromWKB(?))
  ```

  Other programming interfaces may support a similar placeholder mechanism.

- In a C program, you can escape a binary value using `mysql_real_escape_string()` and include the result in a query string that is sent to the server. See Section 22.2.3.47, "`mysql_real_escape_string()`".

18.4.5. Fetching Spatial Data

Geometry values stored in a table can be fetched in internal format. You can also convert them into WKT or WKB format.

18.4.5.1. Fetching Spatial Data in Internal Format

Fetching geometry values using internal format can be useful in table-to-table transfers:

```
CREATE TABLE geom2 (g GEOMETRY) SELECT g FROM geom;
```

18.4.5.2. Fetching Spatial Data in WKT Format

The `AsText()` function converts a geometry from internal format into a WKT string.

```
mysql> SELECT AsText(g) FROM geom;
+-------------------------+
| AsText(p1)              |
+-------------------------+
| POINT(1 1)              |
| LINESTRING(0 0,1 1,2 2) |
+-------------------------+
```

18.4.5.3. Fetching Spatial Data in WKB Format

The `AsBinary()` function converts a geometry from internal format into a `BLOB` containing the WKB value.

```
SELECT AsBinary(g) FROM geom;
```

18.5. Analyzing Spatial Information

After populating spatial columns with values, you are ready to query and analyze them. MySQL provides a set of functions to perform various operations on spatial data. These functions can be grouped into four major categories according to the type of operation they perform:

- Functions that convert geometries between various formats
- Functions that provide access to qualitative or quantitative properties of a geometry
- Functions that describe relations between two geometries
- Functions that create new geometries from existing ones

Spatial analysis functions can be used in many contexts, such as:

- Any interactive SQL program, such as **mysql** or **MySQLCC**
- Application programs written in any language that supports a MySQL client API

18.5.1. Geometry Format Conversion Functions

MySQL supports the following functions for converting geometry values between internal format and either WKT or WKB format:

- `AsBinary(g)`

  Converts a value in internal geometry format to its WKB representation and returns the binary result.

- `AsText(g)`

  Converts a value in internal geometry format to its WKT representation and returns the string result.

```
mysql> SET @g = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(GeomFromText(@g));
+--------------------------+
| AsText(GeomFromText(@G)) |
+--------------------------+
| LINESTRING(1 1,2 2,3 3)  |
+--------------------------+
```

- `GeomFromText(wkt[,srid])`

  Converts a string value from its WKT representation into internal geometry format and returns the result. A number of type-specific functions are also supported, such as `PointFromText()` and `LineFromText()`; see Section 18.4.2.1, “Creating Geometry Values Using WKT Functions”.

- `GeomFromWKB(wkb[,srid])`

  Converts a binary value from its WKB representation into internal geometry format and returns the

result. A number of type-specific functions are also supported, such as `PointFromWKB()` and `LineFromWKB()`; see Section 18.4.2.2, “Creating Geometry Values Using WKB Functions”.

18.5.2. `Geometry` Functions

Each function that belongs to this group takes a geometry value as its argument and returns some quantitative or qualitative property of the geometry. Some functions restrict their argument type. Such functions return `NULL` if the argument is of an incorrect geometry type. For example, `Area()` returns `NULL` if the object type is neither `Polygon` nor `MultiPolygon`.

18.5.2.1. General Geometry Functions

The functions listed in this section do not restrict their argument and accept a geometry value of any type.

- `Dimension(`*`g`*`)`

  Returns the inherent dimension of the geometry value *`g`*. The result can be #1, 0, 1, or 2. (The meaning of these values is given in Section 18.2.2, “Class `Geometry`”.)

```
mysql> SELECT Dimension(GeomFromText('LineString(1 1,2 2)'));
+-----------------------------------------------+
| Dimension(GeomFromText('LineString(1 1,2 2)')) |
+-----------------------------------------------+
|                                             1 |
+-----------------------------------------------+
```

- `Envelope(`*`g`*`)`

  Returns the Minimum Bounding Rectangle (MBR) for the geometry value *`g`*. The result is returned as a `Polygon` value.

```
mysql> SELECT AsText(Envelope(GeomFromText('LineString(1 1,2 2)')));
+-------------------------------------------------------+
| AsText(Envelope(GeomFromText('LineString(1 1,2 2)'))) |
+-------------------------------------------------------+
| POLYGON((1 1,2 1,2 2,1 2,1 1))                        |
+-------------------------------------------------------+
```

  The polygon is defined by the corner points of the bounding box:

```
POLYGON((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

- `GeometryType(`*`g`*`)`

Returns as a string the name of the geometry type of which the geometry instance *g* is a member. The name corresponds to one of the instantiable `Geometry` subclasses.

```
mysql> SELECT GeometryType(GeomFromText('POINT(1 1)'));
+-----------------------------------------+
| GeometryType(GeomFromText('POINT(1 1)')) |
+-----------------------------------------+
| POINT                                   |
+-----------------------------------------+
```

- `SRID(`*g*`)`

  Returns an integer indicating the Spatial Reference System ID for the geometry value *g*.

  In MySQL, the SRID value is just an integer associated with the geometry value. All calculations are done assuming Euclidean (planar) geometry.

```
mysql> SELECT SRID(GeomFromText('LineString(1 1,2 2)',101));
+-----------------------------------------------+
| SRID(GeomFromText('LineString(1 1,2 2)',101)) |
+-----------------------------------------------+
|                                           101 |
+-----------------------------------------------+
```

The OpenGIS specification also defines the following functions, which MySQL does not implement:

- `Boundary(`*g*`)`

  Returns a geometry that is the closure of the combinatorial boundary of the geometry value *g*.

- `IsEmpty(`*g*`)`

  Returns 1 if the geometry value *g* is the empty geometry, 0 if it is not empty, and #1 if the argument is `NULL`. If the geometry is empty, it represents the empty point set.

- `IsSimple(`*g*`)`

  Currently, this function is a placeholder and should not be used. If implemented, its behavior will be as described in the next paragraph.

  Returns 1 if the geometry value *g* has no anomalous geometric points, such as self-intersection or self-tangency. `IsSimple()` returns 0 if the argument is not simple, and #1 if it is `NULL`.

  The description of each instantiable geometric class given earlier in the chapter includes the specific conditions that cause an instance of that class to be classified as not simple.

18.5.2.2. `Point` Functions

A `Point` consists of X and Y coordinates, which may be obtained using the following functions:

- `X(p)`

  Returns the X-coordinate value for the point *p* as a double-precision number.

```
mysql> SELECT X(GeomFromText('Point(56.7 53.34)'));
+--------------------------------------+
| X(GeomFromText('Point(56.7 53.34)')) |
+--------------------------------------+
|                                 56.7 |
+--------------------------------------+
```

- `Y(p)`

  Returns the Y-coordinate value for the point *p* as a double-precision number.

```
mysql> SELECT Y(GeomFromText('Point(56.7 53.34)'));
+--------------------------------------+
| Y(GeomFromText('Point(56.7 53.34)')) |
+--------------------------------------+
|                                53.34 |
+--------------------------------------+
```

18.5.2.3. `LineString` Functions

A `LineString` consists of `Point` values. You can extract particular points of a `LineString`, count the number of points that it contains, or obtain its length.

- `EndPoint(ls)`

  Returns the `Point` that is the end point of the `LineString` value *ls*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(EndPoint(GeomFromText(@ls)));
+-------------------------------------+
| AsText(EndPoint(GeomFromText(@ls))) |
+-------------------------------------+
| POINT(3 3)                          |
+-------------------------------------+
```

- `GLength(ls)`

Returns as a double-precision number the length of the `LineString` value *`ls`* in its associated spatial reference.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT GLength(GeomFromText(@ls));
+----------------------------+
| GLength(GeomFromText(@ls)) |
+----------------------------+
|            2.8284271247462 |
+----------------------------+
```

- `IsClosed(`*`ls`*`)`

  Returns 1 if the `LineString` value *`ls`* is closed (that is, its `StartPoint()` and `EndPoint()` values are the same). Returns 0 if *`ls`* is not closed, and #1 if it is `NULL`.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT IsClosed(GeomFromText(@ls));
+------------------------------+
| IsClosed(GeomFromText(@ls)) |
+------------------------------+
|                            0 |
+------------------------------+
```

- `NumPoints(`*`ls`*`)`

  Returns the number of points in the `LineString` value *`ls`*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT NumPoints(GeomFromText(@ls));
+-------------------------------+
| NumPoints(GeomFromText(@ls)) |
+-------------------------------+
|                             3 |
+-------------------------------+
```

- `PointN(`*`ls`*`,`*`n`*`)`

  Returns the *`n`*-th point in the `Linestring` value *`ls`*. Point numbers begin at 1.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(PointN(GeomFromText(@ls),2));
+-------------------------------------+
| AsText(PointN(GeomFromText(@ls),2)) |
+-------------------------------------+
```

```
| POINT(2 2)                          |
+-------------------------------------+
```

- `StartPoint(ls)`

  Returns the `Point` that is the start point of the `LineString` value *`ls`*.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT AsText(StartPoint(GeomFromText(@ls)));
+---------------------------------------+
| AsText(StartPoint(GeomFromText(@ls))) |
+---------------------------------------+
| POINT(1 1)                            |
+---------------------------------------+
```

The OpenGIS specification also defines the following function, which MySQL does not implement:

- `IsRing(ls)`

  Returns 1 if the `LineString` value *`ls`* is closed (that is, its `StartPoint()` and `EndPoint()` values are the same) and is simple (does not pass through the same point more than once). Returns 0 if *`ls`* is not a ring, and #1 if it is `NULL`.

18.5.2.4. `MultiLineString` Functions

- `GLength(mls)`

  Returns as a double-precision number the length of the `MultiLineString` value *`mls`*. The length of *`mls`* is equal to the sum of the lengths of its elements.

```
mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
mysql> SELECT GLength(GeomFromText(@mls));
+-----------------------------+
| GLength(GeomFromText(@mls)) |
+-----------------------------+
|             4.2426406871193 |
+-----------------------------+
```

- `IsClosed(mls)`

  Returns 1 if the `MultiLineString` value *`mls`* is closed (that is, the `StartPoint()` and `EndPoint()` values are the same for each `LineString` in *`mls`*). Returns 0 if *`mls`* is not closed, and #1 if it is `NULL`.

```
mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
mysql> SELECT IsClosed(GeomFromText(@mls));
```

```
+------------------------------+
| IsClosed(GeomFromText(@mls)) |
+------------------------------+
|                            0 |
+------------------------------+
```

18.5.2.5. Polygon Functions

- `Area(poly)`

  Returns as a double-precision number the area of the `Polygon` value *`poly`*, as measured in its spatial reference system.

```
mysql> SET @poly = 'Polygon((0 0,0 3,3 0,0 0),(1 1,1 2,2 1,1 1))';
mysql> SELECT Area(GeomFromText(@poly));
+---------------------------+
| Area(GeomFromText(@poly)) |
+---------------------------+
|                         4 |
+---------------------------+
```

- `ExteriorRing(poly)`

  Returns the exterior ring of the `Polygon` value *`poly`* as a `LineString`.

```
mysql> SET @poly =
    -> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT AsText(ExteriorRing(GeomFromText(@poly)));
+-------------------------------------------+
| AsText(ExteriorRing(GeomFromText(@poly))) |
+-------------------------------------------+
| LINESTRING(0 0,0 3,3 3,3 0,0 0)           |
+-------------------------------------------+
```

- `InteriorRingN(poly,n)`

  Returns the *`n`*-th interior ring for the `Polygon` value *`poly`* as a `LineString`. Ring numbers begin at 1.

```
mysql> SET @poly =
    -> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT AsText(InteriorRingN(GeomFromText(@poly),1));
+---------------------------------------------+
| AsText(InteriorRingN(GeomFromText(@poly),1)) |
+---------------------------------------------+
| LINESTRING(1 1,1 2,2 2,2 1,1 1)             |
```

```
+---------------------------------------------+
```

- `NumInteriorRings(poly)`

  Returns the number of interior rings in the `Polygon` value *`poly`*.

```
mysql> SET @poly =
    -> 'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT NumInteriorRings(GeomFromText(@poly));
+----------------------------------------+
| NumInteriorRings(GeomFromText(@poly)) |
+----------------------------------------+
|                                      1 |
+----------------------------------------+
```

18.5.2.6. `MultiPolygon` Functions

- `Area(mpoly)`

  Returns as a double-precision number the area of the `MultiPolygon` value *`mpoly`*, as measured in its spatial reference system.

```
mysql> SET @mpoly =
    -> 'MultiPolygon(((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1)))';
mysql> SELECT Area(GeomFromText(@mpoly));
+----------------------------+
| Area(GeomFromText(@mpoly)) |
+----------------------------+
|                          8 |
+----------------------------+
```

The OpenGIS specification also defines the following functions, which MySQL does not implement:

- `Centroid(mpoly)`

  Returns the mathematical centroid for the `MultiPolygon` value *`mpoly`* as a `Point`. The result is not guaranteed to be on the `MultiPolygon`.

- `PointOnSurface(mpoly)`

  Returns a `Point` value that is guaranteed to be on the `MultiPolygon` value *`mpoly`*.

18.5.2.7. `GeometryCollection` Functions

- `GeometryN(gc,n)`

Returns the *n*-th geometry in the `GeometryCollection` value *gc*. Geometry numbers begin at 1.

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
mysql> SELECT AsText(GeometryN(GeomFromText(@gc),1));
+----------------------------------------+
| AsText(GeometryN(GeomFromText(@gc),1)) |
+----------------------------------------+
| POINT(1 1)                             |
+----------------------------------------+
```

- `NumGeometries(gc)`

  Returns the number of geometries in the `GeometryCollection` value *gc*.

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
mysql> SELECT NumGeometries(GeomFromText(@gc));
+-----------------------------------+
| NumGeometries(GeomFromText(@gc)) |
+-----------------------------------+
|                                 2 |
+-----------------------------------+
```

18.5.3. Functions That Create New Geometries from Existing Ones

18.5.3.1. Geometry Functions That Produce New Geometries

In the section Section 18.5.2, “`Geometry` Functions”, we've discussed some functions that can construct new geometries from the existing ones:

- `Envelope(g)`
- `StartPoint(ls)`
- `EndPoint(ls)`
- `PointN(ls,n)`
- `ExteriorRing(poly)`
- `InteriorRingN(poly,n)`
- `GeometryN(gc,n)`

18.5.3.2. Spatial Operators

OpenGIS proposes a number of other functions that can produce geometries. They are designed to implement spatial operators.

These functions are not implemented in MySQL. They may appear in future releases.

- `Buffer(g,d)`

  Returns a geometry that represents all points whose distance from the geometry value *g* is less than or equal to a distance of *d*.

- `ConvexHull(g)`

  Returns a geometry that represents the convex hull of the geometry value *g*.

- `Difference(g1,g2)`

  Returns a geometry that represents the point set difference of the geometry value *g1* with *g2*.

- `Intersection(g1,g2)`

  Returns a geometry that represents the point set intersection of the geometry values *g1* with *g2*.

- `SymDifference(g1,g2)`

  Returns a geometry that represents the point set symmetric difference of the geometry value *g1* with *g2*.

- `Union(g1,g2)`

  Returns a geometry that represents the point set union of the geometry values *g1* and *g2*.

18.5.4. Functions for Testing Spatial Relations Between Geometric Objects

The functions described in these sections take two geometries as input parameters and return a qualitative or quantitative relation between them.

18.5.5. Relations on Geometry Minimal Bounding Rectangles (MBRs)

MySQL provides some functions that can test relations between minimal bounding rectangles of two geometries `g1` and `g2`. They include:

- `MBRContains(g1,g2)`

  Returns 1 or 0 to indicate whether or not the Minimum Bounding Rectangle of *g1* contains the Minimum Bounding Rectangle of *g2*.

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Point(1 1)');
mysql> SELECT MBRContains(@g1,@g2), MBRContains(@g2,@g1);
----------------------+----------------------+
| MBRContains(@g1,@g2) | MBRContains(@g2,@g1) |
+----------------------+----------------------+
|                    1 |                    0 |
+----------------------+----------------------+
```

- `MBRDisjoint(g1,g2)`

  Returns 1 or 0 to indicate whether or not the Minimum Bounding Rectangles of the two geometries *g1* and *g2* are disjoint (do not intersect).

- `MBREqual(g1,g2)`

  Returns 1 or 0 to indicate whether or not the Minimum Bounding Rectangles of the two geometries *g1* and *g2* are the same.

- `MBRIntersects(g1,g2)`

  Returns 1 or 0 to indicate whether or not the Minimum Bounding Rectangles of the two geometries *g1* and *g2* intersect.

- `MBROverlaps(g1,g2)`

  Returns 1 or 0 to indicate whether or not the Minimum Bounding Rectangles of the two geometries *g1* and *g2* overlap.

- `MBRTouches(g1,g2)`

  Returns 1 or 0 to indicate whether or not the Minimum Bounding Rectangles of the two geometries *g1* and *g2* touch.

- `MBRWithin(g1,g2)`

  Returns 1 or 0 to indicate whether or not the Minimum Bounding Rectangle of *g1* is within the Minimum Bounding Rectangle of *g2*.

```
mysql> SET @g1 = GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = GeomFromText('Polygon((0 0,0 5,5 5,5 0,0 0))');
mysql> SELECT MBRWithin(@g1,@g2), MBRWithin(@g2,@g1);
+--------------------+--------------------+
| MBRWithin(@g1,@g2) | MBRWithin(@g2,@g1) |
```

```
+--------------------+--------------------+
|                  1 |                  0 |
+--------------------+--------------------+
```

18.5.6. Functions That Test Spatial Relationships Between Geometries

The OpenGIS specification defines the following functions. Currently, MySQL does not implement them according to the specification. Those that are implemented return the same result as the corresponding MBR-based functions. This includes functions in the following list other than `Distance()` and `Related()`.

These functions may be implemented in future releases with full support for spatial analysis, not just MBR-based support.

The functions operate on two geometry values `g1` and `g2`.

- `Contains(`*`g1`*`,`*`g2`*`)`

  Returns 1 or 0 to indicate whether or not *`g1`* completely contains *`g2`*.

- `Crosses(`*`g1`*`,`*`g2`*`)`

  Returns 1 if *`g1`* spatially crosses *`g2`*. Returns `NULL` if `g1` is a `Polygon` or a `MultiPolygon`, or if *`g2`* is a `Point` or a `MultiPoint`. Otherwise, returns 0.

  The term *spatially crosses* denotes a spatial relation between two given geometries that has the following properties:

  - The two geometries intersect
  - Their intersection results in a geometry that has a dimension that is one less than the maximum dimension of the two given geometries
  - Their intersection is not equal to either of the two given geometries

- `Disjoint(`*`g1`*`,`*`g2`*`)`

  Returns 1 or 0 to indicate whether or not *`g1`* is spatially disjoint from (does not intersect) *`g2`*.

- `Distance(`*`g1`*`,`*`g2`*`)`

  Returns as a double-precision number the shortest distance between any two points in the two geometries.

- `Equals(`*`g1`*`,`*`g2`*`)`

Returns 1 or 0 to indicate whether or not *g1* is spatially equal to *g2*.

- `Intersects(g1,g2)`

  Returns 1 or 0 to indicate whether or not *g1* spatially intersects *g2*.

- `Overlaps(g1,g2)`

  Returns 1 or 0 to indicate whether or not *g1* spatially overlaps *g2*. The term *spatially overlaps* is used if two geometries intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

- `Related(g1,g2,pattern_matrix)`

  Returns 1 or 0 to indicate whether or not the spatial relationship specified by *pattern_matrix* exists between *g1* and *g2*. Returns #1 if the arguments are `NULL`. The pattern matrix is a string. Its specification will be noted here if this function is implemented.

- `Touches(g1,g2)`

  Returns 1 or 0 to indicate whether or not *g1* spatially touches *g2*. Two geometries *spatially touch* if the interiors of the geometries do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

- `Within(g1,g2)`

  Returns 1 or 0 to indicate whether or not *g1* is spatially within *g2*.

18.6. Optimizing Spatial Analysis

Search operations in non-spatial databases can be optimized using indexes. This is true for spatial databases as well. With the help of a great variety of multi-dimensional indexing methods that have previously been designed, it is possible to optimize spatial searches. The most typical of these are:

- Point queries that search for all objects that contain a given point
- Region queries that search for all objects that overlap a given region

MySQL uses **R-Trees with quadratic splitting** to index spatial columns. A spatial index is built using the MBR of a geometry. For most geometries, the MBR is a minimum rectangle that surrounds the geometries. For a horizontal or a vertical linestring, the MBR is a rectangle degenerated into the linestring. For a point, the MBR is a rectangle degenerated into the point.

18.6.1. Creating Spatial Indexes

MySQL can create spatial indexes using syntax similar to that for creating regular indexes, but extended with the `SPATIAL` keyword. Spatial columns that are indexed currently must be declared `NOT NULL`. The following examples demonstrate how to create spatial indexes.

- With `CREATE TABLE`:

  ```
  mysql> CREATE TABLE geom (g GEOMETRY NOT NULL, SPATIAL INDEX(g));
  ```

- With `ALTER TABLE`:

  ```
  mysql> ALTER TABLE geom ADD SPATIAL INDEX(g);
  ```

- With `CREATE INDEX`:

  ```
  mysql> CREATE SPATIAL INDEX sp_index ON geom (g);
  ```

To drop spatial indexes, use `ALTER TABLE` or `DROP INDEX`:

- With `ALTER TABLE`:

  ```
  mysql> ALTER TABLE geom DROP INDEX g;
  ```

- With `DROP INDEX`:

  ```
  mysql> DROP INDEX sp_index ON geom;
  ```

Example: Suppose that a table `geom` contains more than 32,000 geometries, which are stored in the column `g` of type `GEOMETRY`. The table also has an `AUTO_INCREMENT` column `fid` for storing object ID values.

```
mysql> DESCRIBE geom;
+-------+----------+------+-----+---------+----------------+
| Field | Type     | Null | Key | Default | Extra          |
+-------+----------+------+-----+---------+----------------+
| fid   | int(11)  |      | PRI | NULL    | auto_increment |
| g     | geometry |      |     |         |                |
+-------+----------+------+-----+---------+----------------+
2 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM geom;
+----------+
| count(*) |
+----------+
|    32376 |
+----------+
1 row in set (0.00 sec)
```

To add a spatial index on the column `g`, use this statement:

```
mysql> ALTER TABLE geom ADD SPATIAL INDEX(g);
Query OK, 32376 rows affected (4.05 sec)
Records: 32376  Duplicates: 0  Warnings: 0
```

18.6.2. Using a Spatial Index

The optimizer investigates whether available spatial indexes can be involved in the search for queries that use a function such as `MBRContains()` or `MBRWithin()` in the `WHERE` clause. For example, let's say we want to find all objects that are in the given rectangle:

```
mysql> SELECT fid,AsText(g) FROM geom WHERE
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000 16000,3000
+-----+-------------------------------------------------------------------------
| fid | AsText(g)
+-----+-------------------------------------------------------------------------
|  21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30333.8 15828.8)
|  22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8,30334 15871.4)
|  23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4,30334 15914.2)
|  24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4,30273.4 15823)
|  25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882.4,30274.8 15866.2
|  26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4,30275 15918.2)
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946.8,30320.4 15938.4
|   1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136.4,30240 15127.2)
|   2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136,30210.4 15121)
|   3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,30169 15113)
|   4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30157 15111.6)
|   5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4,30194.2 15075.2)
|   6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,30244.6 15077)
|   7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8,30201.2 15049.4)
|  10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6,30189.6 15019)
|  11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2,30151.2 15009.8)
|  13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,30114.6 15067.8)
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30278 15134)
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30259 15083.4)
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4,30128.8 15001)
+-----+-------------------------------------------------------------------------
20 rows in set (0.00 sec)
```

Lets use `EXPLAIN` to check the way this query is executed (the `id` column has been removed so the output better fits the page):

```
mysql> EXPLAIN SELECT fid,AsText(g) FROM geom WHERE
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000 16000,3000
+-------------+-------+-------+---------------+------+---------+------+------+----
| select_type | table | type  | possible_keys | key  | key_len | ref  | rows | Ext
+-------------+-------+-------+---------------+------+---------+------+------+----
| SIMPLE      | geom  | range | g             | g    |      32 | NULL |   50 | Usi
+-------------+-------+-------+---------------+------+---------+------+------+----
1 row in set (0.00 sec)
```

Lets check what would happen without a spatial index:

```
mysql> EXPLAIN SELECT fid,AsText(g) FROM g IGNORE INDEX (g) WHERE
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000 16000,3000
+-------------+-------+------+---------------+------+---------+------+-------+----
| select_type | table | type | possible_keys | key  | key_len | ref  | rows  | Ext
+-------------+-------+------+---------------+------+---------+------+-------+----
| SIMPLE      | geom  | ALL  | NULL          | NULL |    NULL | NULL | 32376 | Usi
+-------------+-------+------+---------------+------+---------+------+-------+----
1 row in set (0.00 sec)
```

Let's execute the `SELECT` statement, ignoring the spatial key we have:

```
mysql> SELECT fid,AsText(g) FROM geom IGNORE INDEX (g) WHERE
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000 16000,3000
+-----+-------------------------------------------------------------------------
| fid | AsText(g)
+-----+-------------------------------------------------------------------------
|   1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136.4,30240 15127.2)
|   2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136,30210.4 15121)
|   3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,30169 15113)
|   4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30157 15111.6)
|   5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4,30194.2 15075.2)
|   6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,30244.6 15077)
|   7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8,30201.2 15049.4)
|  10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6,30189.6 15019)
|  11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2,30151.2 15009.8)
|  13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,30114.6 15067.8)
|  21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30333.8 15828.8)
|  22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8,30334 15871.4)
|  23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4,30334 15914.2)
|  24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4,30273.4 15823)
|  25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882.4,30274.8 15866.2
|  26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4,30275 15918.2)
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30278 15134)
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30259 15083.4)
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4,30128.8 15001)
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946.8,30320.4 15938.4
+-----+-------------------------------------------------------------------------
20 rows in set (0.46 sec)
```

When the index is not used, the execution time for this query rises from 0.00 seconds to 0.46 seconds.

In future releases, spatial indexes may also be used for optimizing other functions. See Section 18.5.4, “Functions for Testing Spatial Relations Between Geometric Objects”.

18.7. MySQL Conformance and Compatibility

18.7.1. GIS Features That Are Not Yet Implemented

- Additional Metadata Views

  OpenGIS specifications propose several additional metadata views. For example, a system view named `GEOMETRY_COLUMNS` contains a description of geometry columns, one row for each geometry column in the database.

- The OpenGIS function `Length()` on `LineString` and `MultiLineString` currently should be called in MySQL as `GLength()`

  The problem is that there is an existing SQL function `Length()` which calculates the length of string values, and sometimes it is not possible to distinguish whether the function is called in a textual or spatial context. We need either to solve this somehow, or decide on another function name.

Chapter 19. Stored Procedures and Functions

Stored procedures and functions are a new feature in MySQL version 5.0. A stored procedure is a set of SQL statements that can be stored in the server. Once this has been done, clients don't need to keep reissuing the individual statements but can refer to the stored procedure instead.

Some situations where stored procedures can be particularly useful:

- When multiple client applications are written in different languages or work on different platforms, but need to perform the same database operations.
- When security is paramount. Banks, for example, use stored procedures for all common operations. This provides a consistent and secure environment, and procedures can ensure that each operation is properly logged. In such a setup, applications and users would not get any access to the database tables directly, but can only execute specific stored procedures.

Stored procedures can provide improved performance because less information needs to be sent between the server and the client. The tradeoff is that this does increase the load on the database server system because more of the work is done on the server side and less is done on the client (application) side. Consider this if many client machines (such as Web servers) are serviced by only one or a few database servers.

Stored procedures also allow you to have libraries of functions in the database server. This is a feature shared by modern application languages that allow such design internally, for example, by using classes. Using these client application language features is beneficial for the programmer even outside the scope of database use.

MySQL follows the SQL:2003 syntax for stored procedures, which is also used by IBM's DB2.

The MySQL implementation of stored procedures is still in progress. All syntax described in this chapter is supported and any limitations and extensions are documented where appropriate.

19.1. Stored Procedures and the Grant Tables

Stored procedures require the `proc` table in the `mysql` database. This table is created during the MySQL 5.0 installation procedure. If you are upgrading to MySQL 5.0 from an earlier version, be sure to update your grant tables to make sure that the `proc` table exists. See Section 2.10.7, “Upgrading the Grant Tables”.

Beginning with MySQL 5.0.3, the grant system has been modified to take stored routines into account as follows:

- The `CREATE ROUTINE` privilege is needed to create stored routines.
- The `ALTER ROUTINE` privilege is needed to alter or drop stored routines. This privilege is granted automatically to the creator of a routine.
- The `EXECUTE` privilege is required to execute stored routines. However, this privilege is granted automatically to the creator of a routine. Also, the default `SQL SECURITY` characteristic for a routine is `DEFINER`, which allows users who have access to the database with which the routine is associated to execute the routine.

19.2. Stored Procedure Syntax

Stored procedures and functions are routines that are created with `CREATE PROCEDURE` and `CREATE FUNCTION` statements. A routine is either a procedure or a function. A procedure is invoked using a `CALL` statement, and can only pass back values using output variables. A function can be called from inside a statement just like any other function (that is, by invoking the function's name), and can return a scalar value. Stored routines may call other stored routines.

As of MySQL 5.0.1, a stored procedure or function is associated with a particular database. This has several implications:

- When the routine is invoked, an implicit `USE db_name` is performed (and undone when the routine terminates). `USE` statements within stored routines are disallowed.
- You can qualify routine names with the database name. This can be used to refer to a routine that is not in the current database. For example, to invoke a stored procedure `p` or function `f` that is associated with the `test` database, you can say `CALL test.p()` or `test.f()`.
- When a database is dropped, all stored routines associated with it are dropped as well.

(In MySQL 5.0.0, stored routines are global and not associated with a database. They inherit the default database from the caller. If a `USE db_name` is executed within the routine, the original default database is restored upon routine exit.)

MySQL supports the very useful extension that allows the use of regular `SELECT` statements (that is, without using cursors or local variables) inside a stored procedure. The result set of such a query is simply sent directly to the client. Multiple `SELECT` statements generate multiple result sets, so the client must use a MySQL client library that supports multiple result sets. This means the client must use a client library from a version of MySQL at least as recent as 4.1.

The following sections describe the syntax used to create, alter, drop, and query stored procedures and functions.

19.2.1. CREATE PROCEDURE and CREATE FUNCTION

```
CREATE PROCEDURE sp_name ([parameter[,...]])
    [characteristic ...] routine_body

CREATE FUNCTION sp_name ([parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body

parameter:
    [ IN | OUT | INOUT ] param_name type

type:
    Any valid MySQL data type

characteristic:
    LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | SQL SECURITY {DEFINER | INVOKER}
  | COMMENT 'string'

routine_body:
    Valid SQL procedure statements or statements
```

These statements create a stored routine. As of MySQL 5.0.3, to create a routine, it is necessary to have the `CREATE ROUTINE` privilege for it, and the `ALTER ROUTINE` and `EXECUTE` privileges are granted automatically to its creator.

By default, the routine is associated with the current database. To associate the routine explicitly with a given database, specify the name as *`db_name.sp_name`* when you create it.

If the routine name is the same as the name of a built-in SQL function, you need to use a space between the name and the following parenthesis when defining the routine, or a syntax error occurs. This is also true when you invoke the routine later.

The `RETURNS` clause may be specified only for a `FUNCTION`, for which it is mandatory. It is used to indicate the return type of the function, and the function body must contain a `RETURN value` statement.

The parameter list enclosed within parentheses must always be present. If there are no parameters, an empty parameter list of `()` should be used. Each parameter is an `IN` parameter by default. To specify otherwise for a parameter, use the keyword `OUT` or `INOUT` before the parameter name. Specifying `IN`, `OUT`, or `INOUT` is only valid for a `PROCEDURE`.

The `CREATE FUNCTION` statement is used in earlier versions of MySQL to support UDFs (User Defined Functions). See Section 25.2, “Adding New Functions to MySQL”. UDFs continue to be supported, even with the existence of stored functions. A UDF can be regarded as an external stored function. However, do note that stored functions share their namespace with UDFs.

A framework for external stored procedures will be introduced in the near future. This will allow you to write stored procedures in languages other than SQL. Most likely, one of the first languages to be supported is PHP because the core PHP engine is small, thread-safe, and can easily be embedded. Because the framework is public, it is expected that many other languages can also be supported.

A procedure or function is considered ``deterministic'' if it always produces the same result for the same input parameters, and ``not deterministic'' otherwise. Currently, the `DETERMINISTIC` characteristic is accepted, but not yet used by the optimizer.

The `SQL SECURITY` characteristic can be used to specify whether the routine should be executed using the permissions of the user who creates the routine or the user who invokes it. The default value is `DEFINER`. This feature is new in SQL:2003. The creator or invoker must have permission to access the database with which the routine is associated. As of MySQL 5.0.3, it is necessary to have the `EXECUTE` privilege to be able to execute the routine. The user that must have this privilege is either the definer or invoker, depending on how the `SQL SECURITY` characteristic is set.

MySQL stores the `sql_mode` system variable setting that is in effect at the time a routine is created, and always executes the routine with this setting in force.

The `COMMENT` clause is a MySQL extension, and may be used to describe the stored procedure. This information is displayed by the `SHOW CREATE PROCEDURE` and `SHOW CREATE FUNCTION` statements.

MySQL allows routines to contain DDL statements (such as `CREATE` and `DROP`) and SQL transaction statements (such as `COMMIT`). This is not required by the standard and therefore is implementation-specific.

Note: Currently, stored functions created with `CREATE FUNCTION` may not contain references to tables. Please note that this includes some `SET` statements, but excludes some `SELECT` statements. This limitation will be lifted as soon as possible.

The following is an example of a simple stored procedure that uses an `OUT` parameter. The example uses the **mysql** client `delimiter` command to change the statement delimiter from `;` to `//` while the procedure is being defined. This allows the `;` delimiter used in the procedure body to be passed through to

the server rather than being interpreted by **mysql** itself.

```
mysql> delimiter //

mysql> CREATE PROCEDURE simpleproc (OUT param1 INT)
    -> BEGIN
    ->   SELECT COUNT(*) INTO param1 FROM t;
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> CALL simpleproc(@a);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @a;
+------+
| @a   |
+------+
| 3    |
+------+
1 row in set (0.00 sec)
```

The following is an example of a function that takes a parameter, performs an operation using an SQL function, and returns the result:

```
mysql> delimiter //

mysql> CREATE FUNCTION hello (s CHAR(20)) RETURNS CHAR(50)
    -> RETURN CONCAT('Hello, ',s,'!');
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> SELECT hello('world');
+----------------+
| hello('world') |
+----------------+
| Hello, world!  |
+----------------+
1 row in set (0.00 sec)
```

19.2.2. ALTER PROCEDURE and ALTER FUNCTION

```
ALTER {PROCEDURE | FUNCTION} sp_name [characteristic ...]

characteristic:
    SQL SECURITY {DEFINER | INVOKER}
  | COMMENT 'string'
```

This statement can be used to change the characteristics of a stored procedure or function. You must have the `ALTER ROUTINE` privilege for the routine as of MySQL 5.0.3. This privilege is granted automatically to the routine creator.

More than one change may be specified in an `ALTER PROCEDURE` or `ALTER FUNCTION` statement.

19.2.3. DROP PROCEDURE and DROP FUNCTION

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

This statement is used to drop a stored procedure or function. That is, the specified routine is removed from the server. You must have the `ALTER ROUTINE` privilege for the routine as of MySQL 5.0.3. This privilege is granted automatically to the routine creator.

The `IF EXISTS` clause is a MySQL extension. It prevents an error from occurring if the procedure or function does not exist. A warning is produced that can be viewed with `SHOW WARNINGS`.

19.2.4. SHOW CREATE PROCEDURE and SHOW CREATE FUNCTION

```
SHOW CREATE {PROCEDURE | FUNCTION} sp_name
```

This statement is a MySQL extension. Similar to `SHOW CREATE TABLE`, it returns the exact string that can be used to re-create the named routine.

```
mysql> SHOW CREATE FUNCTION test.hello\G
*************************** 1. row ***************************
       Function: hello
       sql_mode:
Create Function: CREATE FUNCTION `test`.`hello`(s CHAR(20)) RETURNS CHAR(50)
RETURN CONCAT('Hello, ',s,'!')
```

19.2.5. SHOW PROCEDURE STATUS and SHOW FUNCTION STATUS

```
SHOW {PROCEDURE | FUNCTION} STATUS [LIKE 'pattern']
```

This statement is a MySQL extension. It returns characteristics of routines, such as the database, name, type, creator, and creation and modification dates. If no pattern is specified, the information for all stored procedures or all stored functions is listed, depending on which statement you use.

```
mysql> SHOW FUNCTION STATUS LIKE 'hello'\G
*************************** 1. row ***************************
           Db: test
         Name: hello
         Type: FUNCTION
      Definer: testuser@localhost
     Modified: 2004-08-03 15:29:37
      Created: 2004-08-03 15:29:37
Security_type: DEFINER
      Comment:
```

You can also get information about stored routines from the `ROUTINES` table in `INFORMATION_SCHEMA`. See Section 21.1.14, “The `INFORMATION_SCHEMA ROUTINES` Table”.

19.2.6. CALL Statement

```
CALL sp_name([parameter[,...]])
```

The CALL statement invokes a procedure that was defined previously with CREATE PROCEDURE.

CALL can pass back values to its caller using parameters that are declared as OUT or INOUT parameters. It also ``returns'' the number of rows affected, which a client program can obtain at the SQL level by calling the ROW_COUNT() function and from C by calling the mysql_affected_rows() C API function.

19.2.7. BEGIN ... END Compound Statement

```
[begin_label:] BEGIN
    [statement_list]
END [end_label]
```

Stored routines may contain multiple statements, using a BEGIN ... END compound statement.

A compound statement can be labeled. *end_label* cannot be given unless *begin_label* also is present, and if both are present, they must be the same.

Please note that the optional [NOT] ATOMIC clause is not yet supported. This means that no transactional savepoint is set at the start of the instruction block and the BEGIN clause used in this context has no effect on the current transaction.

Using multiple statements requires that a client is able to send query strings containing the ; statement delimiter. This is handled in the **mysql** command-line client with the delimiter command. Changing the ; end-of-query delimiter (for example, to //) allows ; to be used in a routine body.

19.2.8. DECLARE Statement

The DECLARE statement is used to define various items local to a routine: local variables (see Section 19.2.9, “Variables in Stored Procedures”), conditions and handlers (see Section 19.2.10, “Conditions and Handlers”) and cursors (see Section 19.2.11, “Cursors”). SIGNAL and RESIGNAL statements are not currently supported.

DECLARE may be used only inside a BEGIN ... END compound statement and must be at its start, before any other statements.

Cursors must be declared before declaring handlers, and variables and conditions must be declared before declaring either cursors or handlers.

19.2.9. Variables in Stored Procedures

You may declare and use variables within a routine.

19.2.9.1. DECLARE Local Variables

```
DECLARE var_name[,...] type [DEFAULT value]
```

This statement is used to declare local variables. To provide a default value for the variable, include a DEFAULT clause. The value can be specified as an expression; it need not be a constant.

The scope of a local variable is within the BEGIN ... END block where it is declared. It can be used in nested blocks those that declare a variable with the same name.

19.2.9.2. Variable SET Statement

```
SET var_name = expr [, var_name = expr] ...
```

The SET statement in stored procedures is an extended version of the general SET statement. Referenced variables may be ones declared inside a routine, or global server variables.

The SET statement in stored procedures is implemented as part of the pre-existing SET syntax. This allows an extended syntax of SET a=x, b=y, ... where different variable types (locally declared variables and global and session server variables) can be mixed. This also allows combinations of local variables and some options that make sense only for system variables; in that case, the options are recognized but ignored.

19.2.9.3. SELECT ... INTO Statement

```
SELECT col_name[,...] INTO var_name[,...] table_expr
```

This SELECT syntax stores selected columns directly into variables. Therefore, only a single row may be retrieved. This statement is also extremely useful when used in combination with cursors.

```
SELECT id,data INTO x,y FROM test.t1 LIMIT 1;
```

19.2.10. Conditions and Handlers

Certain conditions may require specific handling. These conditions can relate to errors, as well as general flow control inside a routine.

19.2.10.1. DECLARE Conditions

```
DECLARE condition_name CONDITION FOR condition_value

condition_value:
    SQLSTATE [VALUE] sqlstate_value
  | mysql_error_code
```

This statement specifies conditions that need specific handling. It associates a name with a specified error condition. The name can subsequently be used in a DECLARE HANDLER statement. See Section 19.2.10.2, “DECLARE Handlers”.

In addition to SQLSTATE values, MySQL error codes are also supported.

19.2.10.2. DECLARE Handlers

```
DECLARE handler_type HANDLER FOR condition_value[,...] sp_statement

handler_type:
    CONTINUE
  | EXIT
  | UNDO

condition_value:
    SQLSTATE [VALUE] sqlstate_value
  | condition_name
  | SQLWARNING
  | NOT FOUND
  | SQLEXCEPTION
  | mysql_error_code
```

This statement specifies handlers that each may deal with one or more conditions. If one of these conditions occurs, the specified statement is executed.

For a `CONTINUE` handler, execution of the current routine continues after execution of the handler statement. For an `EXIT` handler, execution of the current `BEGIN...END` compound statement is terminated. The `UNDO` handler type statement is not yet supported.

- `SQLWARNING` is shorthand for all SQLSTATE codes that begin with `01`.
- `NOT FOUND` is shorthand for all SQLSTATE codes that begin with `02`.
- `SQLEXCEPTION` is shorthand for all SQLSTATE codes not caught by `SQLWARNING` or `NOT FOUND`.

In addition to SQLSTATE values, MySQL error codes are also supported.

For example:

```
mysql> CREATE TABLE test.t (s1 int,primary key (s1));
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter //

mysql> CREATE PROCEDURE handlerdemo ()
    -> BEGIN
    ->   DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;
    ->   SET @x = 1;
    ->   INSERT INTO test.t VALUES (1);
    ->   SET @x = 2;
    ->   INSERT INTO test.t VALUES (1);
    ->   SET @x = 3;
    -> END;
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL handlerdemo()//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
    +------+
    | @x   |
    +------+
    | 3    |
    +------+
    1 row in set (0.00 sec)
```

Notice that `@x` is `3`, which shows that MySQL executed to the end of the procedure. If the line `DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;` had not been present, MySQL would have taken the default (`EXIT`) path after the second `INSERT` failed due to the `PRIMARY KEY` constraint, and `SELECT @x` would have returned `2`.

19.2.11. Cursors

Simple cursors are supported inside stored procedures and functions. The syntax is as in embedded SQL. Cursors are currently asensitive, read-only, and non-scrolling. Asensitive means that the server may or may not make a copy of its result table.

Cursors must be declared before declaring handlers, and variables must be declared before declaring either cursors or handlers.

For example:

```
CREATE PROCEDURE curdemo()
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE a CHAR(16);
  DECLARE b,c INT;
  DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
  DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;

  OPEN cur1;
  OPEN cur2;

  REPEAT
    FETCH cur1 INTO a, b;
    FETCH cur2 INTO c;
    IF NOT done THEN
       IF b < c THEN
          INSERT INTO test.t3 VALUES (a,b);
       ELSE
          INSERT INTO test.t3 VALUES (a,c);
       END IF;
    END IF;
  UNTIL done END REPEAT;

  CLOSE cur1;
  CLOSE cur2;
END
```

19.2.11.1. Declaring Cursors

```
DECLARE cursor_name CURSOR FOR select_statement
```

This statement declares a cursor. Multiple cursors may be defined in a routine, but each must have a unique name.

The `SELECT` statement cannot have an `INTO` clause.

19.2.11.2. Cursor `OPEN` Statement

```
OPEN cursor_name
```

This statement opens a previously declared cursor.

19.2.11.3. Cursor `FETCH` Statement

```
FETCH cursor_name INTO var_name [, var_name] ...
```

This statement fetches the next row (if a row exists) using the specified open cursor, and advances the cursor pointer.

19.2.11.4. Cursor `CLOSE` Statement

```
CLOSE cursor_name
```

This statement closes a previously opened cursor.

If not closed explicitly, a cursor is closed at the end of the compound statement in which it was declared.

19.2.12. Flow Control Constructs

The `IF`, `CASE`, `LOOP`, `WHILE`, `ITERATE`, and `LEAVE` constructs are fully implemented.

These constructs may each contain either a single statement, or a block of statements using the `BEGIN ... END` compound statement. Constructs may be nested.

`FOR` loops are not currently supported.

19.2.12.1. IF Statement

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list]
    ...
    [ELSE statement_list]
END IF
```

`IF` implements a basic conditional construct. If the *`search_condition`* evaluates to true, the corresponding SQL statement list is executed. If no *`search_condition`* matches, the statement list in the `ELSE` clause is executed. *`statement_list`* can consist of one or more statements.

Please note that there is also an `IF()` *function*, which differs from the `IF` *statement* described here.. See Section 12.2, “Control Flow Functions”.

19.2.12.2. CASE Statement

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list ...]
    [ELSE statement_list]
END CASE
```

Or:

```
CASE
    WHEN search_condition THEN statement_list
    [WHEN search_condition THEN statement_list ...]
    [ELSE statement_list]
END CASE
```

The `CASE` statement for stored procedures implements a complex conditional construct. If a *`search_condition`* evaluates to true, the corresponding SQL statement is executed. If no search condition matches, the statement in the `ELSE` clause is executed.

Note: The syntax of a `CASE` *statement* shown here for use inside a stored procedure differs slightly from that of the SQL `CASE` *expression* described in Section 12.2, “Control Flow Functions”. The `CASE` statement cannot have an `ELSE NULL` clause, and it is terminated with `END CASE` instead of `END`.

19.2.12.3. LOOP Statement

```
[begin_label:] LOOP
    statement_list
END LOOP [end_label]
```

`LOOP` implements a simple loop construct, enabling repeated execution of a particular statement or statements. The statements within the loop are repeated until the loop is exited; usually this is accomplished with a `LEAVE` statement.

A `LOOP` statement can be labeled. *`end_label`* cannot be given unless *`begin_label`* also is present, and if both are present, they must be the same.

19.2.12.4. LEAVE Statement

```
LEAVE label
```

This statement is used to exit any labeled flow control construct. It can be used with `BEGIN ... END` or loops.

19.2.12.5. ITERATE Statement

```
ITERATE label
```

`ITERATE` can only appear within `LOOP`, `REPEAT`, and `WHILE` statements. `ITERATE` means ``do the loop again."

For example:

```
CREATE PROCEDURE doiterate(p1 INT)
BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN ITERATE label1; END IF;
    LEAVE label1;
  END LOOP label1;
  SET @x = p1;
END
```

19.2.12.6. REPEAT Statement

```
[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]
```

The statement or statements within a `REPEAT` statement are repeated until the *`search_condition`* is true.

A `REPEAT` statement can be labeled. *`end_label`* cannot be given unless *`begin_label`* also is present, and if both are present, they must be the same.

For example:

```
mysql> delimiter //

mysql> CREATE PROCEDURE dorepeat(p1 INT)
```

```
    -> BEGIN
    ->   SET @x = 0;
    ->   REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL dorepeat(1000)//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
+------+
| @x   |
+------+
| 1001 |
+------+
1 row in set (0.00 sec)
```

19.2.12.7. WHILE Statement

```
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

The statement or statements within a `WHILE` statement are repeated as long as the *`search_condition`* is true.

A `WHILE` statement can be labeled. *`end_label`* cannot be given unless *`begin_label`* also is present, and if both are present, they must be the same.

For example:

```
CREATE PROCEDURE dowhile()
BEGIN
  DECLARE v1 INT DEFAULT 5;

  WHILE v1 > 0 DO
    ...
    SET v1 = v1 - 1;
  END WHILE;
END
```

Chapter 20. Triggers

Rudimentary support for triggers is included beginning with MySQL 5.0.2. A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. For example, the following statements set up a table, as well as a trigger for INSERT statements into the table. The trigger sums the values inserted into one of the table's columns:

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
    -> FOR EACH ROW SET @sum = @sum + NEW.amount;
```

This chapter describes the syntax for creating and dropping triggers, and show some examples of how to use them.

20.1. CREATE TRIGGER Syntax

```
CREATE TRIGGER trigger_name trigger_time trigger_event
    ON tbl_name FOR EACH ROW trigger_stmt
```

A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table.

The trigger becomes associated with the table named *tbl_name*. *tbl_name* must refer to a permanent table. You cannot associate a trigger with a TEMPORARY table or a view.

trigger_time is the trigger action time. It can be BEFORE or AFTER to indicate that the trigger activates before or after the statement that activated it.

trigger_event indicates the kind of statement that activates the trigger. It can be INSERT, UPDATE, or DELETE. For example, a BEFORE trigger for INSERT statements could be used to check the values to be inserted into new rows.

There cannot be two triggers for a given table that have the same trigger action time and event. For example, you cannot have two BEFORE UPDATE triggers for a table. But you can have a BEFORE UPDATE and a BEFORE INSERT trigger, or a BEFORE UPDATE and an AFTER UPDATE trigger.

trigger_stmt is the statement to execute when the trigger activates. If you want to execute multiple statements, use the BEGIN ... END compound statement construct. This also enables you to use the same statements that are allowable within stored routines. See Section 19.2.7, "BEGIN ... END Compound Statement".

Note: Currently, triggers have the same limitation as stored functions that they may not contain direct references to tables by name. This limitation will be lifted as soon as possible.

However, in the triggered statement, you can refer to columns in the table associated with the trigger by using the names OLD and NEW. OLD.*col_name* refers to a column of a an existing row before it is updated or deleted. NEW.*col_name* refers to the column of a new row to be inserted or an existing row after it is updated.

Use of SET NEW.*col_name* = *value* requires the UPDATE privilege on the column. Use of SET *value* = NEW.*col_name* requires the SELECT privilege on the column.

The CREATE TRIGGER statement requires the SUPER privilege. It was added in MySQL 5.0.2.

20.2. DROP TRIGGER Syntax

DROP TRIGGER *tbl_name.trigger_name*

Drops a trigger. The name of the trigger to drop must include the table name because each trigger is associated with a particular table.

The DROP TRIGGER statement requires the SUPER privilege. It was added in MySQL 5.0.2.

20.3. Using Triggers

Support for triggers is included beginning with MySQL 5.0.2. Currently, trigger support is rudimentary, so there are certain limitations on what you can do with them. This section discusses how to use triggers and what the current limitations are.

A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. Some uses for triggers are to perform checks of values to be inserted into a table or to perform calculations on values involved in an update.

A trigger is associated with a table and is defined to activate when an INSERT, DELETE, or UPDATE statement for the table executes. A trigger can be set to activate either before or after the triggering statement. For example, you can have a trigger activate before each row that is deleted from a table or after each row that is updated.

To create a trigger or drop a trigger, use the CREATE TRIGGER or DROP TRIGGER statement. The syntax for these statements is described in Section 20.1, "CREATE TRIGGER Syntax" and Section 20.2, "DROP TRIGGER Syntax".

Here is a simple example that associates a trigger with a table for INSERT statements. It acts as an accumulator to sum the values inserted into one of the columns of the table.

The following statements create a table and a trigger for it:

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
    -> FOR EACH ROW SET @sum = @sum + NEW.amount;
```

The CREATE TRIGGER statement creates a trigger named ins_sum that is associated with the account table. It also includes clauses that specify the trigger activation time, the triggering event, and what to do with the trigger activates:

- The keyword BEFORE indicates the trigger action time. In this case, the trigger should activate before each row inserted into the table. The other allowable keyword here is AFTER.
- The keyword INSERT indicates the event that activates the trigger. In the example, INSERT statements cause trigger activation. You can also create triggers for DELETE and UPDATE statements.
- The statement following FOR EACH ROW defines the statement to execute each time the trigger activates, which occurs once for each row affected by the triggering statement In the example, the triggered statement is a simple SET that accumulates the values inserted into the amount column. The statement refers to the column as NEW.amount which means ``the value of the amount column to be inserted into the new row."

To use the trigger, set the accumulator variable to zero, execute an INSERT statement, and then see

what value the variable has afterward:

```
mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
+-----------------------+
| Total amount inserted |
+-----------------------+
| 1852.48               |
+-----------------------+
```

In this case, the value of `@sum` after the `INSERT` statement has executed is `14.98 + 1937.50 - 100` or `1852.48`.

To destroy the trigger, use a `DROP TRIGGER` statement. The trigger name must include the table name:

```
mysql> DROP TRIGGER account.ins_sum;
```

Because a trigger is associated with a particular table, you cannot have multiple triggers for a table that have the same name. You should also be aware that the namespace for triggers might change in the future from table to database. That is, the requirement that trigger names be unique for a given table might be strengthened to the requirement that they be unique within the database. For better forward compatibility with future development, try to use trigger names that do not overlap within a database.

In addition to the requirement that trigger names be unique for a table, there are other limitations on the types of triggers you can create. In particular, you cannot have two triggers for a table that have the same activate time and activation event. For example, you cannot define two `BEFORE INSERT` triggers or two `AFTER UPDATE` triggers for a table. This should rarely be a significant limitation, because it is possible to define a trigger that executes multiple statements by using the `BEGIN … END` compound statement construct after `FOR EACH ROW`. (An example appears later in this section.)

There are also limitations on what can appear in the statement that the trigger executes when activated:

- The trigger cannot refer directly to tables by name, including the table with which the table is associated. You can, however, use the keywords `OLD` and `NEW`. `OLD` refers to an existing row to be deleted or a row to be updated before the update occurs. `NEW` refers to a new row to be inserted or an updated row after the update occurs.
- The trigger cannot invoke stored procedures using the `CALL` statement. (This means, for example, that you cannot get around the prohibition on referring to tables by name by invoking a stored procedure that refers to the tables.)
- The trigger cannot use statements that begin or end a transaction such as `START TRANSACTION`, `COMMIT`, or `ROLLBACK`.

The `OLD` and `NEW` keywords enable you to access columns in the rows affected by a trigger. (`OLD` and `NEW` are not case sensitive.) In an `INSERT` trigger, only `NEW.`*`col_name`* can be used; there is no old row. In a `DELETE` trigger, only `OLD.`*`col_name`* can be used; there is no new row. In an `UPDATE` trigger, you can use `OLD.`*`col_name`* to refer to the columns of a row before it is updated and `NEW.`*`col_name`* to refer to the columns of the row after it is updated.

A column named with `OLD` is read-only. You can refer to it but not modify it. A column named with `NEW` can be referred to if you have the `SELECT` privilege for it. In a `BEFORE` trigger, you can also change its value with `SET NEW.`*`col_name = value`* if you have the `UPDATE` privilege for it. This means you can use a trigger to modify the values to be inserted into a new row or that are used to update a row.

OLD and NEW are MySQL extensions to triggers.

By using the BEGIN ... END construct, you can define a trigger that executes multiple statements. Within the BEGIN block, you also can use other syntax that is allowed within stored routines such as conditionals and loops. However, just as for stored routines, when you define a trigger that executes multiple statements, it becomes necessary to redefine the statement delimiter if you are entering the trigger with the **mysql** program so that you can use the ';' character within the trigger definition. The following example illustrates these points. It defines an UPDATE trigger that checks the new value to be used for updating each row, and modifies the value to be within the range from 0 to 100. This must be a BEFORE trigger because the value needs to be checked before it is used to update the row:

```
mysql> delimiter //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account
    -> FOR EACH ROW
    -> BEGIN
    ->     IF NEW.amount < 0 THEN
    ->         SET NEW.amount = 0;
    ->     ELSEIF NEW.amount > 100 THEN
    ->         SET NEW.amount = 100;
    ->     END IF;
    -> END//
mysql> delimiter ;
```

It might occur to you that it would be easier to define a stored procedure separately and then invoke it from the trigger using a simple CALL statement. That would also be advantageous if you wanted to invoke the same routine from within several triggers. However, a limitation on triggers is that CALL cannot be used. You have to write out the compound statement in each CREATE TRIGGER statement where you want to use it.

Chapter 21. The INFORMATION_SCHEMA Information Database

INFORMATION_SCHEMA support is available in MySQL 5.0.2 and later. It provides access to database metadata.

``Metadata'' is data about the data, such as the name of a database or table, the data type of a column, or access privileges. Other terms that sometimes are used for this information are ``data dictionary'' or ``system catalog.''

Here is an example:

```
mysql> SELECT table_name, table_type, engine
    -> FROM information_schema.tables
    -> WHERE table_schema = 'db5'
    -> ORDER BY table_name DESC;
+------------+------------+--------+
| table_name | table_type | engine |
+------------+------------+--------+
| v56        | VIEW       | NULL   |
| v3         | VIEW       | NULL   |
| v2         | VIEW       | NULL   |
| v          | VIEW       | NULL   |
| tables     | BASE TABLE | MyISAM |
| t7         | BASE TABLE | MyISAM |
| t3         | BASE TABLE | MyISAM |
| t2         | BASE TABLE | MyISAM |
| t          | BASE TABLE | MyISAM |
| pk         | BASE TABLE | InnoDB |
| loop       | BASE TABLE | MyISAM |
| kurs       | BASE TABLE | MyISAM |
| k          | BASE TABLE | MyISAM |
| into       | BASE TABLE | MyISAM |
| goto       | BASE TABLE | MyISAM |
| fk2        | BASE TABLE | InnoDB |
| fk         | BASE TABLE | InnoDB |
+------------+------------+--------+
17 rows in set (0.01 sec)
```

Explanation: The statement requests a list of all the tables in database db5, in reverse alphabetical order, showing just three pieces of information: the name of the table, its type, and its engine.

INFORMATION_SCHEMA is the ``information database'', the place that stores information about all the other databases that the MySQL server maintains. Inside INFORMATION_SCHEMA there are several read-only tables. They are actually views, not base tables, so you won't actually see any file associated with them.

Each MySQL user has the right to access these tables, but only the rows in the tables that correspond to objects for which the user has the proper access privileges.

Advantages of SELECT

The SELECT ... FROM INFORMATION_SCHEMA statement is intended as a more consistent way to provide access to the information provided by the various SHOW statements that MySQL supports (SHOW DATABASES, SHOW TABLES, and so forth). Using SELECT has these advantages, compared to SHOW:

- It conforms to Codd's rules. That is, all access is done on tables.
- Nobody needs to learn a new statement syntax. Because they already know how SELECT works, they only need to learn the object names.
- The implementor need not worry about adding keywords.
- There are millions of possible output variations, instead of just one. This provides more flexibility for applications that have varying requirements about what metadata they need.
- Migration is easier because every other DBMS does it this way.

However, because SHOW is popular with MySQL employees and users, and because it might be confusing were it to disappear, the advantages of conventional syntax are not a sufficient reason to eliminate SHOW. In fact, there are enhancements to SHOW in MySQL 5.0, too. These are described in Section 21.2, “Extensions to SHOW Statements”.

Standards

The implementation for the INFORMATION_SCHEMA table structures in MySQL follows the ANSI/ISO SQL:2003 standard Part 11 ``Schemata." Our intent is approximate compliance with SQL:2003 core feature F021 ``Basic information schema."

Users of SQL Server 2000 (which also follows the standard) may notice a strong similarity. However, MySQL has omitted many columns that are not relevant for our implementation, and added columns that are MySQL-specific. One such column is the engine column in the INFORMATION_SCHEMA.TABLES table.

Although other DBMSs use a variety of names, like syscat or system, the standard name is INFORMATION_SCHEMA.

In effect, we have a new ``database" named information_schema, though there is never a need to make a file by that name. It is not possible to issue USE INFORMATION_SCHEMA. The only thing you can do with these tables is SELECT. You cannot update, insert, delete, or even reference them.

Privileges

There is no difference between the current (SHOW) privilege requirement and the SELECT requirement. In either case, you have to have some privilege on an object in order to see information about it.

21.1. INFORMATION_SCHEMA Tables

Explanation of following sections

In the following sections, we take the tables and columns that are in INFORMATION_SCHEMA. For each column, there are three pieces of information:

- ``Standard Name" indicates the standard SQL name for the column.
- ``SHOW name" indicates what the equivalent field name is in the closest SHOW statement, if any.
- ``Remarks" provides additional information where applicable. We have marked ``omit" those columns for which MySQL has no present use. We have omitted these columns, that is, they appear in the standard but not in MySQL. So their presence here is a matter of record only.

To avoid using any name that's reserved in the standard or in DB2 or in SQL Server or in Oracle we changed the names of columns marked ``MySQL extension.'' (For example, we changed COLLATION to TABLE_COLLATION in the TABLES table.) See the list of reserved words near the end of this article: http://www.dbazine.com/gulutzan5.shtml.

The definition for character columns (for example, TABLES.TABLE_NAME), is generally VARCHAR(*N*) CHARACTER SET utf8 where *N* is at least 64.

Each section indicates what SHOW statement is equivalent to a SELECT that retrieves information from INFORMATION_SCHEMA, or else that there is no such equivalent statement.

Note: At present, there are some missing columns and some columns out of order. We are working on this and intend to update the documentation as changes are made.

21.1.1. The INFORMATION_SCHEMA SCHEMATA Table

A schema is a database, so the SCHEMATA table provides information about databases.

Standard Name	SHOW name	Remarks
CATALOG_NAME	-	NULL
SCHEMA_NAME		Database
SCHEMA_OWNER		omit
DEFAULT_CHARACTER_SET_CATALOG		omit
DEFAULT_CHARACTER_SET_SCHEMA		omit
DEFAULT_CHARACTER_SET_NAME		
SQL_PATH		NULL

Notes:

- For SQL_PATH, we might eventually support something in MySQL 5.x. For now, it's always NULL.

The following statements are equivalent:

```
SELECT SCHEMA_NAME AS `Database`
  FROM INFORMATION_SCHEMA.SCHEMATA
  [WHERE SCHEMA_NAME LIKE 'wild']

SHOW DATABASES
  [LIKE 'wild']
```

21.1.2. The INFORMATION_SCHEMA TABLES Table

The TABLES table provides information about tables in databases.

Standard Name	SHOW name	Remarks
TABLE_CATALOG		NULL
TABLE_SCHEMA	Table_...	
TABLE_NAME	Table_...	

TABLE_TYPE		
SELF_REFERENCING_COLUMN_NAME		omit
REFERENCE_GENERATION		omit
USER_DEFINED_TYPE_NAME		omit
IS_INSERTABLE_INTO		omit
IS_TYPED		omit
COMMIT_ACTION		omit
ENGINE	Engine	MySQL extension
VERSION	Version	MySQL extension
ROW_FORMAT	Row_format	MySQL extension
TABLE_ROWS	Rows	MySQL extension
AVG_ROW_LENGTH	Avg_row_length	MySQL extension
DATA_LENGTH	Data_length	MySQL extension
MAX_DATA_LENGTH	Max_data_length	MySQL extension
INDEX_LENGTH	Index_length	MySQL extension
DATA_FREE	Data_free	MySQL extension
AUTO_INCREMENT	Auto_increment	MySQL extension
CREATE_TIME	Create_time	MySQL extension
UPDATE_TIME	Update_time	MySQL extension
CHECK_TIME	Check_time	MySQL extension
TABLE_COLLATION	Collation	MySQL extension
CHECKSUM	Checksum	MySQL extension
CREATE_OPTIONS	Create_options	MySQL extension
TABLE_COMMENT	Comment	MySQL extension

Notes:

- TABLE_SCHEMA and TABLE_NAME are a single field in a SHOW display, for example Table_in_db1.
- TABLE_TYPE should be BASE TABLE or VIEW. If table is temporary, then TABLE_TYPE = TEMPORARY. (There are no temporary views, so this is not ambiguous.)
- We have nothing for the table's default character set. TABLE_COLLATION is close, because collation names begin with a character set name.

The following statements are equivalent:

```
SELECT table_name FROM INFORMATION_SCHEMA.TABLES
  [WHERE table_schema = 'db_name']
  [WHERE|AND table_name LIKE 'wild']

SHOW TABLES
  [FROM db_name]
  [LIKE 'wild']
```

21.1.3. The INFORMATION_SCHEMA COLUMNS Table

The COLUMNS table provides information about columns in tables.

Standard Name	SHOW name	Remarks
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME	Field	
ORDINAL_POSITION		see notes
COLUMN_DEFAULT	Default	
IS_NULLABLE	Null	
DATA_TYPE	Type	
CHARACTER_MAXIMUM_LENGTH	Type	
CHARACTER_OCTET_LENGTH		
NUMERIC_PRECISION	Type	
NUMERIC_PRECISION_RADIX		omit
NUMERIC_SCALE	Type	
DATETIME_PRECISION		omit
INTERVAL_TYPE		omit
INTERVAL_PRECISION		omit
CHARACTER_SET_CATALOG		omit
CHARACTER_SET_SCHEMA		omit
CHARACTER_SET_NAME		
COLLATION_CATALOG		omit
COLLATION_SCHEMA		omit
COLLATION_NAME	Collation	
DOMAIN_NAME		omit
UDT_CATALOG		omit
UDT_SCHEMA		omit
UDT_NAME		omit
SCOPE_CATALOG		omit
SCOPE_SCHEMA		omit
SCOPE_NAME		omit
MAXIMUM_CARDINALITY		omit
DTD_IDENTIFIER		omit
IS_SELF_REFERENCING		omit
IS_IDENTITY		omit
IDENTITY_GENERATION		omit
IDENTITY_START		omit
IDENTITY_INCREMENT		omit
IDENTITY_MAXIMUM		omit
IDENTITY_MINIMUM		omit

IDENTITY_CYCLE		omit
IS_GENERATED		omit
GENERATION_EXPRESSION		omit
COLUMN_KEY	Key	MySQL extension
EXTRA	Extra	MySQL extension
COLUMN_COMMENT	Comment	MySQL extension

Notes:

- In SHOW, the Type display includes values from several different COLUMNS columns.
- ORDINAL_POSITION is necessary because you might someday want to say ORDER BY ORDINAL_POSITION. Unlike SHOW, SELECT does not have automatic ordering.
- CHARACTER_OCTET_LENGTH should be the same as CHARACTER_MAXIMUM_LENGTH, except for multi-byte character sets.
- CHARACTER_SET_NAME can be derived from Collation. For example, if you say SHOW FULL COLUMNS FROM t, and you see in the Collation column a value of latin1_swedish_ci, the character set is what's before the first underscore: latin1.

The following statements are nearly equivalent:

```
SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_DEFAULT
  FROM INFORMATION_SCHEMA.COLUMNS
  WHERE table_name = 'tbl_name'
  [AND schema_name = 'db_name']
  [AND column_name LIKE 'wild']

SHOW COLUMNS
  FROM tbl_name
  [FROM db_name]
  [LIKE wild]
```

21.1.4. The INFORMATION_SCHEMA STATISTICS Table

The STATISTICS table provides information about table indexes.

Standard Name	SHOW name	Remarks
TABLE_CATALOG		NULL
TABLE_SCHEMA		= Database
TABLE_NAME	Table	
NON_UNIQUE	Non_unique	
INDEX_SCHEMA		= Database
INDEX_NAME	Key_name	
TYPE		omit
SEQ_IN_INDEX	Seq_in_index	
COLUMN_NAME	Column_name	
COLLATION	Collation	

CARDINALITY	Cardinality	
PAGES		omit
FILTER_CONDITION		omit
SUB_PART	Sub_part	MySQL extension
PACKED	Packed	MySQL extension
NULLABLE	Null	MySQL extension
INDEX_TYPE	Index_type	MySQL extension
COMMENT	Comment	MySQL extension

Notes:

- There is no standard table for indexes. The preceding list is similar to what SQL Server 2000 returns for sp_statistics, except that we replaced the name QUALIFIER with CATALOG and we replaced the name OWNER with SCHEMA.

  Clearly, the preceding table and the output from SHOW INDEX are derived from the same parent. So the correlation is already close.

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS
  WHERE table_name = 'tbl_name'
  [AND schema_name = 'db_name']

SHOW INDEX
  FROM tbl_name
  [FROM db_name]
```

21.1.5. The INFORMATION_SCHEMA USER_PRIVILEGES Table

The USER_PRIVILEGES table provides information about global privileges. This information comes from the mysql.user grant table.

Standard Name	SHOW name	Remarks
GRANTEE		e.g. 'user'@'host'
TABLE_CATALOG		NULL
PRIVILEGE_TYPE		
IS_GRANTABLE		

Notes:

- This is a non-standard table. It takes its values from the mysql.user table.

21.1.6. The INFORMATION_SCHEMA SCHEMA_PRIVILEGES

Table

The SCHEMA_PRIVILEGES table provides information about schema (database) privileges. This information comes from the mysql.db grant table.

Standard Name	SHOW name	Remarks
GRANTEE		e.g. 'user'@'host'
TABLE_CATALOG		NULL
TABLE_SCHEMA		
PRIVILEGE_TYPE		
IS_GRANTABLE		

Notes:

- This is a non-standard table. It takes its values from the mysql.db table.

21.1.7. The INFORMATION_SCHEMA TABLE_PRIVILEGES Table

The TABLE_PRIVILEGES table provides information about table privileges. This information comes from the mysql.tables_priv grant table.

Standard Name	SHOW name	Remarks
GRANTOR		omit
GRANTEE		e.g. 'user'@'host'
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
PRIVILEGE_TYPE		
IS_GRANTABLE		
WITH_HIERARCHY		omit

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES
```

```
SHOW GRANTS ...
```

PRIVILEGE_TYPE can contain one (and only one) of these values: SELECT, INSERT, UPDATE, REFERENCES, ALTER, INDEX, DROP, CREATE VIEW.

21.1.8. The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table

The COLUMN_PRIVILEGES table provides information about column privileges. This information

comes from the mysql.columns_priv grant table.

Standard Name	SHOW name	Remarks
GRANTOR		omit
GRANTEE		e.g. 'user'@'host'
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME		
PRIVILEGE_TYPE		
IS_GRANTABLE		

Notes:

- In the output from SHOW FULL COLUMNS, the privileges are all in one field and in lowercase, for example, select,insert,update,references. In COLUMN_PRIVILEGES, there is one row per privilege, and it's uppercase.
- PRIVILEGE_TYPE can contain one (and only one) of these values: SELECT, INSERT, UPDATE, REFERENCES.
- If the user has GRANT OPTION privilege, then IS_GRANTABLE should be YES. Otherwise, IS_GRANTABLE should be NO. The output does not list GRANT OPTION as a separate privilege.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.COLUMN_PRIVILEGES
```

```
SHOW GRANTS ...
```

21.1.9. The INFORMATION_SCHEMA CHARACTER_SETS Table

The CHARACTER_SETS table provides information about available character sets.

Standard Name	SHOW name	Remarks
CHARACTER_SET_CATALOG		omit
CHARACTER_SET_SCHEMA		omit
CHARACTER_SET_NAME	Charset	
CHARACTER_REPERTOIRE		omit
FORM_OF_USE		omit
NUMBER_OF_CHARACTERS		omit
DEFAULT_COLLATE_CATALOG		omit
DEFAULT_COLLATE_SCHEMA		omit
DEFAULT_COLLATE_NAME	Default collation	
DESCRIPION	Description	MySQL extension
MAXLEN	Maxlen	MySQL extension

Notes:

- We have added two non-standard columns corresponding to the `Description` and `Maxlen` columns in the output from `SHOW CHARACTER SET`.

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.CHARACTER_SETS
  [WHERE name LIKE 'wild']

SHOW CHARACTER SET
  [LIKE 'wild']
```

21.1.10. The INFORMATION_SCHEMA COLLATIONS Table

The `COLLATIONS` table provides information about collations for each character set.

Standard Name	SHOW name	Remarks
COLLATION_CATALOG		omit
COLLATION_SCHEMA		omit
COLLATION_NAME	Collation	
PAD_ATTRIBUTE		omit
COLLATION_TYPE		omit
COLLATION_DEFINITION		omit
COLLATION_DICTIONARY		omit
CHARACTER_SET_NAME		omit MySQL extension
ID		omit MySQL extension
IS_DEFAULT		omit MySQL extension
IS_COMPILED		omit MySQL extension
SORTLEN		omit MySQL extension

Notes:

- We have added five non-standard columns corresponding to the `Charset`, `Id`, `Default`, `Compiled`, and `Sortlen` columns in the output from `SHOW COLLATION`.

The following statements are equivalent:

```
SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLLATIONS
  [WHERE collation_name LIKE 'wild']

SHOW COLLATION
  [LIKE 'wild']
```

21.1.11. The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table

The COLLATION_CHARACTER_SET_APPLICABILITY table indicates what character set is applicable for what collation. The columns are equivalent to the first two display fields that we get from SHOW COLLATION.

Standard Name	SHOW name	Remarks
COLLATION_CATALOG		omit
COLLATION_SCHEMA		omit
COLLATION_NAME	Collation	
CHARACTER_SET_CATALOG		omit
CHARACTER_SET_SCHEMA		omit
CHARACTER_SET_NAME	Charset	

21.1.12. The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table

The TABLE_CONSTRAINTS table describes which tables have constraints.

Standard Name	SHOW name	Remarks
CONSTRAINT_CATALOG		NULL
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
TABLE_CATALOG		omit
TABLE_SCHEMA		
TABLE_NAME		
CONSTRAINT_TYPE		
IS_DEFERRABLE		omit
INITIALLY_DEFERRED		omit

Notes:

- The CONSTRAINT_TYPE value can be UNIQUE, PRIMARY KEY, or FOREIGN KEY.
- The UNIQUE and PRIMARY KEY information is about the same as what you get from the Key_name field in the output from SHOW INDEX when the Non_unique field is 0.
- The CONSTRAINT_TYPE column can contain one of these values: UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK. This is a CHAR (not ENUM) column. The CHECK value is not available until we support CHECK.

21.1.13. The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table

The KEY_COLUMN_USAGE table describes which key columns have constraints.

Standard Name	SHOW name	Remarks

CONSTRAINT_CATALOG		NULL
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
TABLE_CATALOG		
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME		
ORDINAL_POSITION		
POSI-TION_IN_UNIQUE_CONSTRAINT		

Notes:

- If the constraint is a foreign key, then this is the column of the foreign key, not the column that the foreign key references.
- The value of ORDINAL_POSITION is the column's position within the constraint, not the column's position within the table. Column positions are numbered beginning with 1.
- The value of POSITION_IN_UNIQUE_CONSTRAINT is NULL for unique and primary-key constraints. For foreign-key constraints, it is the ordinal position in key of the table that is being referenced.

  For example, suppose that there are two tables name t1 and t3 that have the following definitions:

  ```
  CREATE TABLE t1
  (
      s1 INT,
      s2 INT,
      s3 INT,
      PRIMARY KEY(s3)
  ) ENGINE=InnoDB;

  CREATE TABLE t3
  (
      s1 INT,
      s2 INT,
      s3 INT,
      KEY(s1),
      CONSTRAINT CO FOREIGN KEY (s2) REFERENCES t1(s3)
  ) ENGINE=InnoDB;
  ```

  For those two tables, the KEY_COLUMN_USAGE table has two rows:

  - One row with CONSTRAINT_NAME='PRIMARY', TABLE_NAME='t1', COLUMN_NAME='s3', ORDINAL_POSITION=1, POSITION_IN_UNIQUE_CONSTRAINT=NULL.
  - One row with CONSTRAINT_NAME='CO', TABLE_NAME='t3', COLUMN_NAME='s2', ORDINAL_POSITION=1, POSITION_IN_UNIQUE_CONSTRAINT=1.

21.1.14. The INFORMATION_SCHEMA ROUTINES Table

The ROUTINES table provides information about stored routines (both procedures and functions). The ROUTINES table does not include user-defined functions (UDFs) at this time.

The column named ``mysql.proc name'' indicates the mysql.proc table column that corresponds to the INFORMATION_SCHEMA.ROUTINES table column, if any.

Standard Name	mysql.proc name	Remarks
SPECIFIC_CATALOG		omit
SPECIFIC_SCHEMA	db	omit
SPECIFIC_NAME	specific_name	
ROUTINE_CATALOG		NULL
ROUTINE_SCHEMA	db	
ROUTINE_NAME	name	
MODULE_CATALOG		omit
MODULE_SCHEMA		omit
MODULE_NAME		omit
USER_DEFINED_TYPE_CATALOG		omit
USER_DEFINED_TYPE_SCHEMA		omit
USER_DEFINED_TYPE_NAME		omit
ROUTINE_TYPE	type	{PROCEDURE|FUNCTION}
DTD_IDENTIFIER		(data type descriptor)
ROUTINE_BODY		SQL
ROUTINE_DEFINITION	body	
EXTERNAL_NAME		NULL
EXTERNAL_LANGUAGE	language	NULL
PARAMETER_STYLE		SQL
IS_DETERMINISTIC	is_deterministic	
SQL_DATA_ACCESS	sql_data_access	
IS_NULL_CALL		omit
SQL_PATH		NULL
SCHEMA_LEVEL_ROUTINE		omit
MAX_DYNAMIC_RESULT_SETS		omit
IS_USER_DEFINED_CAST		omit
IS_IMPLICITLY_INVOCABLE		omit
SECURITY_TYPE	security_type	
TO_SQL_SPECIFIC_CATALOG		omit
TO_SQL_SPECIFIC_SCHEMA		omit
TO_SQL_SPECIFIC_NAME		omit
AS_LOCATOR		omit
CREATED	created	
LAST_ALTERED	modified	
NEW_SAVEPOINT_LEVEL		omit
IS_UDT_DEPENDENT		omit

RES-ULT_CAST_FROM_DTD_IDENTIFIE R		omit
RESULT_CAST_AS_LOCATOR		omit
SQL_MODE	sql_mode	MySQL extension
ROUTINE_COMMENT	comment	MySQL extension
DEFINER	definer	MySQL extension

Notes:

- MySQL calculates EXTERNAL_LANGUAGE thus:
  - If mysql.proc.language='SQL', then EXTERNAL_LANGUAGE is NULL
  - Otherwise, EXTERNAL_LANGUAGE is what's in mysql.proc.language. However, we don't have external languages yet, so it's always NULL.

21.1.15. The INFORMATION_SCHEMA VIEWS Table

The VIEWS table provides information about views in databases.

Standard Name	SHOW name	Remarks
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
VIEW_DEFINITION		
CHECK_OPTION		
IS_UPDATABLE		
INSERTABLE_INTO		omit

Notes:

- There is a new privilege, SHOW VIEW, without which you cannot see the VIEWS table.
- The VIEW_DEFINITION column has most of what you see in the Create Table field that SHOW CREATE VIEW produces. Skip the words before SELECT and skip the words WITH CHECK OPTION. For example, if the original statement was:

  ```
  CREATE VIEW v AS
    SELECT s2,s1 FROM t
    WHERE s1 > 5
    ORDER BY s1
    WITH CHECK OPTION;
  ```

  then the view definition is:

  ```
  SELECT s2,s1 FROM t WHERE s1 > 5 ORDER BY s1
  ```

- The CHECK_OPTION column always has a value of NONE.
- The IS_UPDATABLE column is YES if the view is updatable, NO if the view is not updatable.

21.1.16. Other INFORMATION_SCHEMA Tables

We will add more INFORMATION_SCHEMA tables soon. Particularly, we acknowledge the need for INFORMATION_SCHEMA.PARAMETERS and for INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS and for INFORMATION_SCHEMA.TRIGGERS.

21.2. Extensions to SHOW Statements

Some extensions to SHOW statements accompany the implementation of INFORMATION_SCHEMA:

- SHOW can be used to get information about the structure of INFORMATION_SCHEMA itself.
- Several SHOW statements accept a WHERE clause that provides more flexibility in specifying which rows to display.

These extensions are available beginning with MySQL 5.0.3.

INFORMATION_SCHEMA is an information database, so its name is included in the output from SHOW DATABASES. Similarly, SHOW TABLES can be used with INFORMATION_SCHEMA to obtain a list of its tables:

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA;
+---------------------------------------+
| Tables_in_information_schema          |
+---------------------------------------+
| SCHEMATA                              |
| TABLES                                |
| COLUMNS                               |
| CHARACTER_SETS                        |
| COLLATIONS                            |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| ROUTINES                              |
| STATISTICS                            |
| VIEWS                                 |
| USER_PRIVILEGES                       |
| SCHEMA_PRIVILEGES                     |
| TABLE_PRIVILEGES                      |
| COLUMN_PRIVILEGES                     |
| TABLE_CONSTRAINTS                     |
| KEY_COLUMN_USAGE                      |
+---------------------------------------+
```

SHOW COLUMNS and DESCRIBE can display information about the columns in individual INFORMATION_SCHEMA tables.

Several SHOW statement have been extended to allow a WHERE clause:

```
SHOW CHARACTER SET
SHOW COLLATION
SHOW COLUMNS
SHOW DATABASES
SHOW FUNCTION STATUS
```

```
SHOW KEYS
SHOW OPEN TABLES
SHOW PROCEDURE STATUS
SHOW STATUS
SHOW TABLE STATUS
SHOW TABLES
SHOW VARIABLES
```

The WHERE clause, if present, is evaluated against the column names displayed by the SHOW statement. For example, the SHOW COLLATION statement produces these output columns:

For example, the SHOW CHARACTER SET statement produces these output columns:

```
mysql> SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |      2 |
| dec8     | DEC West European           | dec8_swedish_ci     |      1 |
| cp850    | DOS West European           | cp850_general_ci    |      1 |
| hp8      | HP West European            | hp8_english_ci      |      1 |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |      1 |
| latin1   | ISO 8859-1 West European    | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |      1 |

...
```

To use a WHERE clause with SHOW CHARACTER SET, you would refer to those column names. As an example, the following statement displays information about character sets for which the default collation contains the string "japanese":

```
mysql> SHOW CHARACTER SET WHERE `Default collation` LIKE '%japanese%';
+---------+---------------------------+---------------------+--------+
| Charset | Description               | Default collation   | Maxlen |
+---------+---------------------------+---------------------+--------+
| ujis    | EUC-JP Japanese           | ujis_japanese_ci    |      3 |
| sjis    | Shift-JIS Japanese        | sjis_japanese_ci    |      2 |
| cp932   | SJIS for Windows Japanese | cp932_japanese_ci   |      2 |
| eucjpms | UJIS for Windows Japanese | eucjpms_japanese_ci |      3 |
+---------+---------------------------+---------------------+--------+
```

This statement displays the multi-byte character sets:

```
mysql> SHOW CHARACTER SET WHERE Maxlen > 1;
+---------+---------------------------+---------------------+--------+
| Charset | Description               | Default collation   | Maxlen |
+---------+---------------------------+---------------------+--------+
| big5    | Big5 Traditional Chinese  | big5_chinese_ci     |      2 |
| ujis    | EUC-JP Japanese           | ujis_japanese_ci    |      3 |
| sjis    | Shift-JIS Japanese        | sjis_japanese_ci    |      2 |
| euckr   | EUC-KR Korean             | euckr_korean_ci     |      2 |
| gb2312  | GB2312 Simplified Chinese | gb2312_chinese_ci   |      2 |
| gbk     | GBK Simplified Chinese    | gbk_chinese_ci      |      2 |
| utf8    | UTF-8 Unicode             | utf8_general_ci     |      3 |
| ucs2    | UCS-2 Unicode             | ucs2_general_ci     |      2 |
| cp932   | SJIS for Windows Japanese | cp932_japanese_ci   |      2 |
| eucjpms | UJIS for Windows Japanese | eucjpms_japanese_ci |      3 |
+---------+---------------------------+---------------------+--------+
```

Chapter 22. MySQL APIs

This chapter describes the APIs available for MySQL, where to get them, and how to use them. The C API is the most extensively covered, because it was developed by the MySQL team, and is the basis for most of the other APIs.

22.1. MySQL Program Development Utilities

This section describes some utilities that you may find useful when developing MySQL programs.

- `msql2mysql`

  A shell script that converts `mSQL` programs to MySQL. It doesn't handle every case, but it gives a good start when converting.

- `mysql_config`

  A shell script that produces the option values needed when compiling MySQL programs.

22.1.1. msql2mysql, Convert mSQL Programs for Use with MySQL

Initially, the MySQL C API was developed to be very similar to that for the mSQL database system. Because of this, mSQL programs often can be converted relatively easily for use with MySQL by changing the names of the C API functions.

The **msql2mysql** utility performs the conversion of mSQL C API function calls to their MySQL equivalents. **msql2mysql** converts the input file in place, so make a copy of the original before converting it. For example, use **msql2mysql** like this:

```
shell> cp client-prog.c client-prog.c.orig
shell> msql2mysql client-prog.c
client-prog.c converted
```

Then examine `client-prog.c` and make any post-conversion revisions that may be necessary.

msql2mysql uses the **replace** utility to make the function name substitutions. See Section 8.13, “The **replace** String-Replacement Utility”.

22.1.2. mysql_config, Get compile options for compiling clients

mysql_config provides you with useful information for compiling your MySQL client and connecting it to MySQL.

mysql_config supports the following options:

- `--cflags`

Compiler flags to find include files and critical compiler flags and defines used when compiling the `libmysqlclient` library.

- `--include`

  Compiler options to find MySQL include files. (Note that normally you would use `--cflags` instead of this option.)

- `--libmysqld-libs, --embedded`

  Libraries and options required to link with the MySQL embedded server.

- `--libs`

  Libraries and options required to link with the MySQL client library.

- `--libs_r`

  Libraries and options required to link with the thread-safe MySQL client library.

- `--port`

  The default TCP/IP port number, defined when configuring MySQL.

- `--socket`

  The default Unix socket file, defined when configuring MySQL.

- `--version`

  Version number and version for the MySQL distribution.

If you invoke **mysql_config** with no options, it displays a list of all options that it supports, and their values:

```
shell> mysql_config
Usage: /usr/local/mysql/bin/mysql_config [options]
Options:
  --cflags         [-I/usr/local/mysql/include/mysql -mcpu=pentiumpro]
  --include        [-I/usr/local/mysql/include/mysql]
  --libs           [-L/usr/local/mysql/lib/mysql -lmysqlclient -lz
                    -lcrypt -lnsl -lm -L/usr/lib -lssl -lcrypto]
  --libs_r         [-L/usr/local/mysql/lib/mysql -lmysqlclient_r
                    -lpthread -lz -lcrypt -lnsl -lm -lpthread]
  --socket         [/tmp/mysql.sock]
  --port           [3306]
  --version        [4.0.16]
  --libmysqld-libs [-L/usr/local/mysql/lib/mysql -lmysqld -lpthread -lz
                    -lcrypt -lnsl -lm -lpthread -lrt]
```

You can use **mysql_config** within a command line to include the value that it displays for a particular option. For example, to compile a MySQL client program, use **mysql_config** as follows:

```
CFG=/usr/local/mysql/bin/mysql_config
sh -c "gcc -o progname `$CFG --cflags` progname.c `$CFG --libs`"
```

When you use **mysql_config** this way, be sure to invoke it within backtick ('`') characters. That tells the

shell to execute it and substitute its output into the surrounding command.

22.2. MySQL C API

The C API code is distributed with MySQL. It is included in the `mysqlclient` library and allows C programs to access a database.

Many of the clients in the MySQL source distribution are written in C. If you are looking for examples that demonstrate how to use the C API, take a look at these clients. You can find these in the `clients` directory in the MySQL source distribution.

Most of the other client APIs (all except Connector/J) use the `mysqlclient` library to communicate with the MySQL server. This means that, for example, you can take advantage of many of the same environment variables that are used by other client programs, because they are referenced from the library. See Chapter 8, *MySQL Client and Utility Programs*, for a list of these variables.

The client has a maximum communication buffer size. The size of the buffer that is allocated initially (16KB) is automatically increased up to the maximum size (the maximum is 16MB). Because buffer sizes are increased only as demand warrants, simply increasing the default maximum limit does not in itself cause more resources to be used. This size check is mostly a check for erroneous queries and communication packets.

The communication buffer must be large enough to contain a single SQL statement (for client-to-server traffic) and one row of returned data (for server-to-client traffic). Each thread's communication buffer is dynamically enlarged to handle any query or row up to the maximum limit. For example, if you have `BLOB` values that contain up to 16MB of data, you must have a communication buffer limit of at least 16MB (in both server and client). The client's default maximum is 16MB, but the default maximum in the server is 1MB. You can increase this by changing the value of the `max_allowed_packet` parameter when the server is started. See Section 7.5.2, “Tuning Server Parameters”.

The MySQL server shrinks each communication buffer to `net_buffer_length` bytes after each query. For clients, the size of the buffer associated with a connection is not decreased until the connection is closed, at which time client memory is reclaimed.

For programming with threads, see Section 22.2.15, “How to Make a Threaded Client”. For creating a standalone application which includes the "server" and "client" in the same program (and does not communicate with an external MySQL server), see Section 22.2.16, “libmysqld, the Embedded MySQL Server Library”.

22.2.1. C API Data types

- `MYSQL`

  This structure represents a handle to one database connection. It is used for almost all MySQL functions.

- `MYSQL_RES`

  This structure represents the result of a query that returns rows (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`). The information returned from a query is called the *result set* in the remainder of this section.

- `MYSQL_ROW`

  This is a type-safe representation of one row of data. It is currently implemented as an array of coun-

ted byte strings. (You cannot treat these as null-terminated strings if field values may contain binary data, because such values may contain null bytes internally.) Rows are obtained by calling `mysql_fetch_row()`.

- `MYSQL_FIELD`

  This structure contains information about a field, such as the field's name, type, and size. Its members are described in more detail here. You may obtain the `MYSQL_FIELD` structures for each field by calling `mysql_fetch_field()` repeatedly. Field values are not part of this structure; they are contained in a `MYSQL_ROW` structure.

- `MYSQL_FIELD_OFFSET`

  This is a type-safe representation of an offset into a MySQL field list. (Used by `mysql_field_seek()`.) Offsets are field numbers within a row, beginning at zero.

- `my_ulonglong`

  The type used for the number of rows and for `mysql_affected_rows()`, `mysql_num_rows()`, and `mysql_insert_id()`. This type provides a range of `0` to `1.84e19`.

  On some systems, attempting to print a value of type `my_ulonglong` does not work. To print such a value, convert it to `unsigned long` and use a `%lu` print format. Example:

  ```
  printf ("Number of rows: %lu\n", (unsigned long) mysql_num_rows(result));
  ```

The `MYSQL_FIELD` structure contains the members listed here:

- `char * name`

  The name of the field, as a null-terminated string.

- `char * table`

  The name of the table containing this field, if it isn't a calculated field. For calculated fields, the `table` value is an empty string.

- `char * def`

  The default value of this field, as a null-terminated string. This is set only if you use `mysql_list_fields()`.

- `enum enum_field_types type`

  The type of the field. The `type` value may be one of the following:

Type Value	Type Description
`FIELD_TYPE_TINY`	`TINYINT` field
`FIELD_TYPE_SHORT`	`SMALLINT` field
`FIELD_TYPE_LONG`	`INTEGER` field

FIELD_TYPE_INT24	MEDIUMINT field
FIELD_TYPE_LONGLONG	BIGINT field
FIELD_TYPE_DECIMAL	DECIMAL or NUMERIC field
FIELD_TYPE_FLOAT	FLOAT field
FIELD_TYPE_DOUBLE	DOUBLE or REAL field
FIELD_TYPE_TIMESTAMP	TIMESTAMP field
FIELD_TYPE_DATE	DATE field
FIELD_TYPE_TIME	TIME field
FIELD_TYPE_DATETIME	DATETIME field
FIELD_TYPE_YEAR	YEAR field
FIELD_TYPE_STRING	CHAR field
FIELD_TYPE_VAR_STRING	VARCHAR field
FIELD_TYPE_BLOB	BLOB or TEXT field (use max_length to determine the maximum length)
FIELD_TYPE_SET	SET field
FIELD_TYPE_ENUM	ENUM field
FIELD_TYPE_NULL	NULL-type field
FIELD_TYPE_CHAR	Deprecated; use FIELD_TYPE_TINY instead

You can use the IS_NUM() macro to test whether a field has a numeric type. Pass the type value to IS_NUM() and it evaluates to TRUE if the field is numeric:

```
if (IS_NUM(field->type))
    printf("Field is numeric\n");
```

- unsigned int length

  The width of the field, as specified in the table definition.

- unsigned int max_length

  The maximum width of the field for the result set (the length of the longest field value for the rows actually in the result set). If you use mysql_store_result() or mysql_list_fields(), this contains the maximum length for the field. If you use mysql_use_result(), the value of this variable is zero.

- unsigned int flags

  Different bit-flags for the field. The flags value may have zero or more of the following bits set:

Flag Value	Flag Description
NOT_NULL_FLAG	Field can't be NULL
PRI_KEY_FLAG	Field is part of a primary key
UNIQUE_KEY_FLAG	Field is part of a unique key
MULTIPLE_KEY_FLAG	Field is part of a non-unique key
UNSIGNED_FLAG	Field has the UNSIGNED attribute

ZEROFILL_FLAG	Field has the ZEROFILL attribute
BINARY_FLAG	Field has the BINARY attribute
AUTO_INCREMENT_FLAG	Field has the AUTO_INCREMENT attribute
ENUM_FLAG	Field is an ENUM (deprecated)
SET_FLAG	Field is a SET (deprecated)
BLOB_FLAG	Field is a BLOB or TEXT (deprecated)
TIMESTAMP_FLAG	Field is a TIMESTAMP (deprecated)

Use of the BLOB_FLAG, ENUM_FLAG, SET_FLAG, and TIMESTAMP_FLAG flags is deprecated because they indicate the type of a field rather than an attribute of its type. It is preferable to test field->type against FIELD_TYPE_BLOB, FIELD_TYPE_ENUM, FIELD_TYPE_SET, or FIELD_TYPE_TIMESTAMP instead.

The following example illustrates a typical use of the flags value:

```
if (field->flags & NOT_NULL_FLAG)
    printf("Field can't be null\n");
```

You may use the following convenience macros to determine the boolean status of the flags value:

Flag Status	Description
IS_NOT_NULL(flags)	True if this field is defined as NOT NULL
IS_PRI_KEY(flags)	True if this field is a primary key
IS_BLOB(flags)	True if this field is a BLOB or TEXT (deprecated; test field->type instead)

- unsigned int decimals

  The number of decimals for numeric fields.

22.2.2. C API Function Overview

The functions available in the C API are summarized here and described in greater detail in a later section. See Section 22.2.3, “C API Function Descriptions”.

Function	Description
mysql_affected_rows()	Returns the number of rows changed/deleted/inserted by the last UPDATE, DELETE, or INSERT query.
mysql_change_user()	Changes user and database on an open connection.
mysql_charset_name()	Returns the name of the default character set for the connection.
mysql_close()	Closes a server connection.
mysql_connect()	Connects to a MySQL server. This function is deprecated; use mysql_real_connect() instead.
mysql_create_db()	Creates a database. This function is deprecated; use the SQL statement CREATE DATABASE instead.

mysql_data_seek()	Seeks to an arbitrary row number in a query result set.
mysql_debug()	Does a `DBUG_PUSH` with the given string.
mysql_drop_db()	Drops a database. This function is deprecated; use the SQL statement `DROP DATABASE` instead.
mysql_dump_debug_info()	Makes the server write debug information to the log.
mysql_eof()	Determines whether the last row of a result set has been read. This function is deprecated; `mysql_errno()` or `mysql_error()` may be used instead.
mysql_errno()	Returns the error number for the most recently invoked MySQL function.
mysql_error()	Returns the error message for the most recently invoked MySQL function.
mysql_escape_string()	Escapes special characters in a string for use in an SQL statement.
mysql_fetch_field()	Returns the type of the next table field.
mysql_fetch_field_direct()	Returns the type of a table field, given a field number.
mysql_fetch_fields()	Returns an array of all field structures.
mysql_fetch_lengths()	Returns the lengths of all columns in the current row.
mysql_fetch_row()	Fetches the next row from the result set.
mysql_field_seek()	Puts the column cursor on a specified column.
mysql_field_count()	Returns the number of result columns for the most recent statement.
mysql_field_tell()	Returns the position of the field cursor used for the last `mysql_fetch_field()`.
mysql_free_result()	Frees memory used by a result set.
mysql_get_client_info()	Returns client version information as a string.
mysql_get_client_version()	Returns client version information as an integer.
mysql_get_host_info()	Returns a string describing the connection.
mysql_get_server_version()	Returns version number of server as an integer (new in 4.1).
mysql_get_proto_info()	Returns the protocol version used by the connection.
mysql_get_server_info()	Returns the server version number.
mysql_info()	Returns information about the most recently executed query.
mysql_init()	Gets or initializes a `MYSQL` structure.
mysql_insert_id()	Returns the ID generated for an `AUTO_INCREMENT` column by the previous query.
mysql_kill()	Kills a given thread.
mysql_library_end()	Finalize MySQL C API library.
mysql_library_init()	Initialize MySQL C API library.
mysql_list_dbs()	Returns database names matching a simple regular expression.
mysql_list_fields()	Returns field names matching a simple regular expression.
mysql_list_processes()	Returns a list of the current server threads.
mysql_list_tables()	Returns table names matching a simple regular expression.
mysql_num_fields()	Returns the number of columns in a result set.
mysql_num_rows()	Returns the number of rows in a result set.
mysql_options()	Sets connect options for `mysql_connect()`.
mysql_ping()	Checks whether the connection to the server is working, reconnecting

	as necessary.
mysql_query()	Executes an SQL query specified as a null-terminated string.
mysql_real_connect()	Connects to a MySQL server.
mysql_real_escape_string()	Escapes special characters in a string for use in an SQL statement, taking into account the current charset of the connection.
mysql_real_query()	Executes an SQL query specified as a counted string.
mysql_reload()	Tells the server to reload the grant tables.
mysql_row_seek()	Seeks to a row offset in a result set, using value returned from `mysql_row_tell()`.
mysql_row_tell()	Returns the row cursor position.
mysql_select_db()	Selects a database.
mysql_server_end()	Finalize embedded server library.
mysql_server_init()	Initialize embedded server library.
mysql_set_server_option()	Sets an option for the connection (like `multi-statements`).
mysql_sqlstate()	Returns the SQLSTATE error code for the last error.
mysql_shutdown()	Shuts down the database server.
mysql_stat()	Returns the server status as a string.
mysql_store_result()	Retrieves a complete result set to the client.
mysql_thread_id()	Returns the current thread ID.
mysql_thread_safe()	Returns 1 if the clients are compiled as thread-safe.
mysql_use_result()	Initiates a row-by-row result set retrieval.
mysql_warning_count()	Returns the warning count for the previous SQL statement.
mysql_commit()	Commits the transaction.
mysql_rollback()	Rolls back the transaction.
mysql_autocommit()	Toggles autocommit mode on/off.
mysql_more_results()	Checks whether any more results exist.
mysql_next_result()	Returns/initiates the next result in multiple-statement executions.

Application programs should use this general outline for interacting with MySQL:

1. Initialize the MySQL library by calling `mysql_library_init()`. The library can be either the `mysqlclient` C client library or the `mysqld` embedded server library, depending on whether the application was linked with the `-libmysqlclient` or `-libmysqld` flag.

2. Initialize a connection handler by calling `mysql_init()` and connect to the server by calling `mysql_real_connect()`.

3. Issue SQL statements and process their results. (The following discussion provides more information about how to do this.)

4. Close the connection to the MySQL server by calling `mysql_close()`.

5. End use of the MySQL library by calling `mysql_library_end()`.

The purpose of calling `mysql_library_init()` and `mysql_library_end()` is to provide proper initialization and finalization of the MySQL library. For applications that are linked with the cli-

ent library, they provide improved memory management. If you don't call `mysql_library_end()`, a block of memory remains allocated. (This does not increase the amount of memory used by the application, but some memory leak detectors will complain about it.) For applications that are linked with the embedded server, these calls start and stop the server.

`mysql_library_init()` and `mysql_library_end()` are available as of MySQL 4.1.10 and 5.0.3. These actually are `#define` symbols that make them equivalent to `mysql_server_init()` and `mysql_server_end()`, but the names more clearly indicate that they should be called when beginning and ending use of a MySQL library no matter whether the application uses the `mysqlclient` or `mysqld` library. For older versions of MySQL, you can call `mysql_server_init()` and `mysql_server_end()` instead.

If you like, the call to `mysql_library_init()` may be omitted, because `mysql_init()` will invoke it automatically as necessary.

To connect to the server, call `mysql_init()` to initialize a connection handler, then call `mysql_real_connect()` with that handler (along with other information such as the hostname, username, and password). Upon connection, `mysql_real_connect()` sets the `reconnect` flag (part of the `MYSQL` structure) to a value of `1` in versions of the API strictly older than 5.0.3, of `0` in newer versions. A value of `1` for this flag indicates, in the event that a query cannot be performed because of a lost connection, to try reconnecting to the server before giving up. When you are done with the connection, call `mysql_close()` to terminate it.

While a connection is active, the client may send SQL queries to the server using `mysql_query()` or `mysql_real_query()`. The difference between the two is that `mysql_query()` expects the query to be specified as a null-terminated string whereas `mysql_real_query()` expects a counted string. If the string contains binary data (which may include null bytes), you must use `mysql_real_query()`.

For each non-`SELECT` query (for example, `INSERT`, `UPDATE`, `DELETE`), you can find out how many rows were changed (affected) by calling `mysql_affected_rows()`.

For `SELECT` queries, you retrieve the selected rows as a result set. (Note that some statements are `SELECT`-like in that they return rows. These include `SHOW`, `DESCRIBE`, and `EXPLAIN`. They should be treated the same way as `SELECT` statements.)

There are two ways for a client to process result sets. One way is to retrieve the entire result set all at once by calling `mysql_store_result()`. This function acquires from the server all the rows returned by the query and stores them in the client. The second way is for the client to initiate a row-by-row result set retrieval by calling `mysql_use_result()`. This function initializes the retrieval, but does not actually get any rows from the server.

In both cases, you access rows by calling `mysql_fetch_row()`. With `mysql_store_result()`, `mysql_fetch_row()` accesses rows that have previously been fetched from the server. With `mysql_use_result()`, `mysql_fetch_row()` actually retrieves the row from the server. Information about the size of the data in each row is available by calling `mysql_fetch_lengths()`.

After you are done with a result set, call `mysql_free_result()` to free the memory used for it.

The two retrieval mechanisms are complementary. Client programs should choose the approach that is most appropriate for their requirements. In practice, clients tend to use `mysql_store_result()` more commonly.

An advantage of `mysql_store_result()` is that because the rows have all been fetched to the client, you not only can access rows sequentially, you can move back and forth in the result set using `mysql_data_seek()` or `mysql_row_seek()` to change the current row position within the result set. You can also find out how many rows there are by calling `mysql_num_rows()`. On the other hand, the memory requirements for `mysql_store_result()` may be very high for large result sets and you are more likely to encounter out-of-memory conditions.

An advantage of `mysql_use_result()` is that the client requires less memory for the result set because it maintains only one row at a time (and because there is less allocation overhead, `mysql_use_result()` can be faster). Disadvantages are that you must process each row quickly to avoid tying up the server, you don't have random access to rows within the result set (you can only access rows sequentially), and you don't know how many rows are in the result set until you have retrieved them all. Furthermore, you **must** retrieve all the rows even if you determine in mid-retrieval that you've found the information you were looking for.

The API makes it possible for clients to respond appropriately to queries (retrieving rows only as necessary) without knowing whether or not the query is a `SELECT`. You can do this by calling `mysql_store_result()` after each `mysql_query()` (or `mysql_real_query()`). If the result set call succeeds, the query was a `SELECT` and you can read the rows. If the result set call fails, call `mysql_field_count()` to determine whether a result was actually to be expected. If `mysql_field_count()` returns zero, the query returned no data (indicating that it was an `INSERT`, `UPDATE`, `DELETE`, etc.), and was not expected to return rows. If `mysql_field_count()` is non-zero, the query should have returned rows, but didn't. This indicates that the query was a `SELECT` that failed. See the description for `mysql_field_count()` for an example of how this can be done.

Both `mysql_store_result()` and `mysql_use_result()` allow you to obtain information about the fields that make up the result set (the number of fields, their names and types, etc.). You can access field information sequentially within the row by calling `mysql_fetch_field()` repeatedly, or by field number within the row by calling `mysql_fetch_field_direct()`. The current field cursor position may be changed by calling `mysql_field_seek()`. Setting the field cursor affects subsequent calls to `mysql_fetch_field()`. You can also get information for fields all at once by calling `mysql_fetch_fields()`.

For detecting and reporting errors, MySQL provides access to error information by means of the `mysql_errno()` and `mysql_error()` functions. These return the error code or error message for the most recently invoked function that can succeed or fail, allowing you to determine when an error occurred and what it was.

22.2.3. C API Function Descriptions

In the descriptions here, a parameter or return value of `NULL` means `NULL` in the sense of the C programming language, not a MySQL `NULL` value.

Functions that return a value generally return a pointer or an integer. Unless specified otherwise, functions returning a pointer return a non-`NULL` value to indicate success or a `NULL` value to indicate an error, and functions returning an integer return zero to indicate success or non-zero to indicate an error. Note that ``non-zero'' means just that. Unless the function description says otherwise, do not test against a value other than zero:

```
if (result)                    /* correct */
    ... error ...

if (result < 0)                /* incorrect */
    ... error ...

if (result == -1)              /* incorrect */
    ... error ...
```

When a function returns an error, the **Errors** subsection of the function description lists the possible types of errors. You can find out which of these occurred by calling `mysql_errno()`. A string representation of the error may be obtained by calling `mysql_error()`.

22.2.3.1. `mysql_affected_rows()`

```
my_ulonglong mysql_affected_rows(MYSQL *mysql)
```

Description

Returns the number of rows changed by the last `UPDATE`, deleted by the last `DELETE` or inserted by the last `INSERT` statement. May be called immediately after `mysql_query()` for `UPDATE`, `DELETE`, or `INSERT` statements. For `SELECT` statements, `mysql_affected_rows()` works like `mysql_num_rows()`.

Return Values

An integer greater than zero indicates the number of rows affected or retrieved. Zero indicates that no records were updated for an `UPDATE` statement, no rows matched the `WHERE` clause in the query or that no query has yet been executed. #1 indicates that the query returned an error or that, for a `SELECT` query, `mysql_affected_rows()` was called prior to calling `mysql_store_result()`. Because `mysql_affected_rows()` returns an unsigned value, you can check for #1 by comparing the return value to `(my_ulonglong)-1` (or to `(my_ulonglong)~0`, which is equivalent).

Errors

None.

Example

```
mysql_query(&mysql,"UPDATE products SET cost=cost*1.25 WHERE group=10");
printf("%ld products updated",(long) mysql_affected_rows(&mysql));
```

If you specify the flag `CLIENT_FOUND_ROWS` when connecting to **mysqld**, `mysql_affected_rows()` returns the number of rows matched by the `WHERE` statement for `UPDATE` statements.

Note that when you use a `REPLACE` command, `mysql_affected_rows()` returns 2 if the new row replaced an old row. This is because in this case one row was inserted after the duplicate was deleted.

If you use `INSERT ... ON DUPLICATE KEY UPDATE` to insert a row, `mysql_affected_rows()` returns 1 if the row is inserted as a new row and 2 if an existing row is updated.

22.2.3.2. `mysql_change_user()`

```
my_bool mysql_change_user(MYSQL *mysql, const char *user, const char
*password, const char *db)
```

Description

Changes the user and causes the database specified by `db` to become the default (current) database on the connection specified by `mysql`. In subsequent queries, this database is the default for table references that do not include an explicit database specifier.

This function was introduced in MySQL 3.23.3.

`mysql_change_user()` fails if the connected user cannot be authenticated or doesn't have permission to use the database. In this case the user and database are not changed

The `db` parameter may be set to `NULL` if you don't want to have a default database.

Starting from MySQL 4.0.6 this command always performs a `ROLLBACK` of any active transactions, closes all temporary tables, unlocks all locked tables and resets the state as if one had done a new con-

nect. This happens even if the user didn't change.

Return Values

Zero for success. Non-zero if an error occurred.

Errors

The same that you can get from `mysql_real_connect()`.

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

- `ER_UNKNOWN_COM_ERROR`

  The MySQL server doesn't implement this command (probably an old server).

- `ER_ACCESS_DENIED_ERROR`

  The user or password was wrong.

- `ER_BAD_DB_ERROR`

  The database didn't exist.

- `ER_DBACCESS_DENIED_ERROR`

  The user did not have access rights to the database.

- `ER_WRONG_DB_NAME`

  The database name was too long.

Example

```
if (mysql_change_user(&mysql, "user", "password", "new_database"))
{
   fprintf(stderr, "Failed to change user.  Error: %s\n",
           mysql_error(&mysql));
}
```

22.2.3.3. `mysql_character_set_name()`

```
const char *mysql_character_set_name(MYSQL *mysql)
```

Description

Returns the default character set for the current connection.

Return Values

The default character set

Errors

None.

22.2.3.4. mysql_close()

```
void mysql_close(MYSQL *mysql)
```

Description

Closes a previously opened connection. `mysql_close()` also deallocates the connection handle pointed to by `mysql` if the handle was allocated automatically by `mysql_init()` or `mysql_connect()`.

Return Values

None.

Errors

None.

22.2.3.5. mysql_connect()

```
MYSQL *mysql_connect(MYSQL *mysql, const char *host, const char *user,
const char *passwd)
```

Description

This function is deprecated. It is preferable to use `mysql_real_connect()` instead.

`mysql_connect()` attempts to establish a connection to a MySQL database engine running on `host`. `mysql_connect()` must complete successfully before you can execute any of the other API functions, with the exception of `mysql_get_client_info()`.

The meanings of the parameters are the same as for the corresponding parameters for `mysql_real_connect()` with the difference that the connection parameter may be `NULL`. In this case the C API allocates memory for the connection structure automatically and frees it when you call `mysql_close()`. The disadvantage of this approach is that you can't retrieve an error message if the connection fails. (To get error information from `mysql_errno()` or `mysql_error()`, you must provide a valid `MYSQL` pointer.)

Return Values

Same as for `mysql_real_connect()`.

Errors

Same as for `mysql_real_connect()`.

22.2.3.6. mysql_create_db()

```
int mysql_create_db(MYSQL *mysql, const char *db)
```

Description

Creates the database named by the `db` parameter.

This function is deprecated. It is preferable to use `mysql_query()` to issue an SQL `CREATE DATABASE` statement instead.

Return Values

Zero if the database was created successfully. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

```
if(mysql_create_db(&mysql, "my_database"))
{
   fprintf(stderr, "Failed to create new database.  Error: %s\n",
           mysql_error(&mysql));
}
```

22.2.3.7. mysql_data_seek()

```
void mysql_data_seek(MYSQL_RES *result, my_ulonglong offset)
```

Description

Seeks to an arbitrary row in a query result set. The `offset` value is a row number and should be in the range from `0` to `mysql_num_rows(result)-1`.

This function requires that the result set structure contains the entire result of the query, so `mysql_data_seek()` may be used only in conjunction with `mysql_store_result()`, not with `mysql_use_result()`.

Return Values

None.

Errors

None.

22.2.3.8. mysql_debug()

```
void mysql_debug(const char *debug)
```

Description

Does a DBUG_PUSH with the given string. mysql_debug() uses the Fred Fish debug library. To use this function, you must compile the client library to support debugging. See Section E.1, “Debugging a MySQL Server”. See Section E.2, “Debugging a MySQL Client”.

Return Values

None.

Errors

None.

Example

The call shown here causes the client library to generate a trace file in /tmp/client.trace on the client machine:

```
mysql_debug("d:t:O,/tmp/client.trace");
```

22.2.3.9. mysql_drop_db()

```
int mysql_drop_db(MYSQL *mysql, const char *db)
```

Description

Drops the database named by the db parameter.

This function is deprecated. It is preferable to use mysql_query() to issue an SQL DROP DATABASE statement instead.

Return Values

Zero if the database was dropped successfully. Non-zero if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

  Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

  The MySQL server has gone away.

- CR_SERVER_LOST

The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

Example

```
if(mysql_drop_db(&mysql, "my_database"))
  fprintf(stderr, "Failed to drop the database: Error: %s\n",
          mysql_error(&mysql));
```

22.2.3.10. mysql_dump_debug_info()

```
int mysql_dump_debug_info(MYSQL *mysql)
```

Description

Instructs the server to write some debug information to the log. For this to work, the connected user must have the SUPER privilege.

Return Values

Zero if the command was successful. Non-zero if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

  Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

  The MySQL server has gone away.

- CR_SERVER_LOST

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

22.2.3.11. mysql_eof()

```
my_bool mysql_eof(MYSQL_RES *result)
```

Description

This function is deprecated. mysql_errno() or mysql_error() may be used instead.

mysql_eof() determines whether the last row of a result set has been read.

If you acquire a result set from a successful call to mysql_store_result(), the client receives the entire set in one operation. In this case, a NULL return from mysql_fetch_row() always means the

end of the result set has been reached and it is unnecessary to call `mysql_eof()`. When used with `mysql_store_result()`, `mysql_eof()` always returns true.

On the other hand, if you use `mysql_use_result()` to initiate a result set retrieval, the rows of the set are obtained from the server one by one as you call `mysql_fetch_row()` repeatedly. Because an error may occur on the connection during this process, a `NULL` return value from `mysql_fetch_row()` does not necessarily mean the end of the result set was reached normally. In this case, you can use `mysql_eof()` to determine what happened. `mysql_eof()` returns a non-zero value if the end of the result set was reached and zero if an error occurred.

Historically, `mysql_eof()` predates the standard MySQL error functions `mysql_errno()` and `mysql_error()`. Because those error functions provide the same information, their use is preferred over `mysql_eof()`, which is deprecated. (In fact, they provide more information, because `mysql_eof()` returns only a boolean value whereas the error functions indicate a reason for the error when one occurs.)

Return Values

Zero if no error occurred. Non-zero if the end of the result set has been reached.

Errors

None.

Example

The following example shows how you might use `mysql_eof()`:

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(!mysql_eof(result))  // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

However, you can achieve the same effect with the standard MySQL error functions:

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(mysql_errno(&mysql))  // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

22.2.3.12. `mysql_errno()`

```
unsigned int mysql_errno(MYSQL *mysql)
```

Description

For the connection specified by `mysql`, `mysql_errno()` returns the error code for the most recently

invoked API function that can succeed or fail. A return value of zero means that no error occurred. Client error message numbers are listed in the MySQL `errmsg.h` header file. Server error message numbers are listed in `mysqld_error.h`. In the MySQL source distribution you can find a complete list of error messages and error numbers in the file `Docs/mysqld_error.txt`. The server error codes also are listed at Chapter 24, *Error Handling in MySQL*.

Note that some functions like `mysql_fetch_row()` don't set `mysql_errno()` if they succeed.

A rule of thumb is that all functions that have to ask the server for information reset `mysql_errno()` if they succeed.

Return Values

An error code value for the last `mysql_xxx()` call, if it failed. zero means no error occurred.

Errors

None.

22.2.3.13. `mysql_error()`

```
const char *mysql_error(MYSQL *mysql)
```

Description

For the connection specified by `mysql`, `mysql_error()` returns a null-terminated string containing the error message for the most recently invoked API function that failed. If a function didn't fail, the return value of `mysql_error()` may be the previous error or an empty string to indicate no error.

A rule of thumb is that all functions that have to ask the server for information reset `mysql_error()` if they succeed.

For functions that reset `mysql_errno()`, the following two tests are equivalent:

```
if(mysql_errno(&mysql))
{
    // an error occurred
}

if(mysql_error(&mysql)[0] != '\0')
{
    // an error occurred
}
```

The language of the client error messages may be changed by recompiling the MySQL client library. Currently you can choose error messages in several different languages. See Section 5.8.2, “Setting the Error Message Language”.

Return Values

A null-terminated character string that describes the error. An empty string if no error occurred.

Errors

None.

22.2.3.14. `mysql_escape_string()`

You should use `mysql_real_escape_string()` instead!

This function is identical to `mysql_real_escape_string()` except that `mysql_real_escape_string()` takes a connection handler as its first argument and escapes the string according to the current character set. `mysql_escape_string()` does not take a connection argument and does not respect the current charset setting.

22.2.3.15. mysql_fetch_field()

```
MYSQL_FIELD *mysql_fetch_field(MYSQL_RES *result)
```

Description

Returns the definition of one column of a result set as a `MYSQL_FIELD` structure. Call this function repeatedly to retrieve information about all columns in the result set. `mysql_fetch_field()` returns `NULL` when no more fields are left.

`mysql_fetch_field()` is reset to return information about the first field each time you execute a new `SELECT` query. The field returned by `mysql_fetch_field()` is also affected by calls to `mysql_field_seek()`.

If you've called `mysql_query()` to perform a `SELECT` on a table but have not called `mysql_store_result()`, MySQL returns the default blob length (8KB) if you call `mysql_fetch_field()` to ask for the length of a `BLOB` field. (The 8KB size is chosen because MySQL doesn't know the maximum length for the `BLOB`. This should be made configurable sometime.) Once you've retrieved the result set, `field->max_length` contains the length of the largest value for this column in the specific query.

Return Values

The `MYSQL_FIELD` structure for the current column. `NULL` if no columns are left.

Errors

None.

Example

```
MYSQL_FIELD *field;

while((field = mysql_fetch_field(result)))
{
    printf("field name %s\n", field->name);
}
```

22.2.3.16. mysql_fetch_fields()

```
MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *result)
```

Description

Returns an array of all `MYSQL_FIELD` structures for a result set. Each structure provides the field definition for one column of the result set.

Return Values

An array of `MYSQL_FIELD` structures for all columns of a result set.

Errors

None.

Example

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *fields;

num_fields = mysql_num_fields(result);
fields = mysql_fetch_fields(result);
for(i = 0; i < num_fields; i++)
{
   printf("Field %u is %s\n", i, fields[i].name);
}
```

22.2.3.17. mysql_fetch_field_direct()

```
MYSQL_FIELD *mysql_fetch_field_direct(MYSQL_RES *result, unsigned int
fieldnr)
```

Description

Given a field number `fieldnr` for a column within a result set, returns that column's field definition as a `MYSQL_FIELD` structure. You may use this function to retrieve the definition for an arbitrary column. The value of `fieldnr` should be in the range from 0 to `mysql_num_fields(result)-1`.

Return Values

The `MYSQL_FIELD` structure for the specified column.

Errors

None.

Example

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *field;

num_fields = mysql_num_fields(result);
for(i = 0; i < num_fields; i++)
{
    field = mysql_fetch_field_direct(result, i);
    printf("Field %u is %s\n", i, field->name);
}
```

22.2.3.18. mysql_fetch_lengths()

```
unsigned long *mysql_fetch_lengths(MYSQL_RES *result)
```

Description

Returns the lengths of the columns of the current row within a result set. If you plan to copy field values, this length information is also useful for optimization, because you can avoid calling `strlen()`. In addition, if the result set contains binary data, you **must** use this function to determine the size of the data, because `strlen()` returns incorrect results for any field containing null characters.

The length for empty columns and for columns containing NULL values is zero. To see how to distinguish these two cases, see the description for mysql_fetch_row().

Return Values

An array of unsigned long integers representing the size of each column (not including any terminating null characters). NULL if an error occurred.

Errors

mysql_fetch_lengths() is valid only for the current row of the result set. It returns NULL if you call it before calling mysql_fetch_row() or after retrieving all rows in the result.

Example

```
MYSQL_ROW row;
unsigned long *lengths;
unsigned int num_fields;
unsigned int i;

row = mysql_fetch_row(result);
if (row)
{
    num_fields = mysql_num_fields(result);
    lengths = mysql_fetch_lengths(result);
    for(i = 0; i < num_fields; i++)
    {
         printf("Column %u is %lu bytes in length.\n", i, lengths[i]);
    }
}
```

22.2.3.19. mysql_fetch_row()

```
MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)
```

Description

Retrieves the next row of a result set. When used after mysql_store_result(), mysql_fetch_row() returns NULL when there are no more rows to retrieve. When used after mysql_use_result(), mysql_fetch_row() returns NULL when there are no more rows to retrieve or if an error occurred.

The number of values in the row is given by mysql_num_fields(result). If row holds the return value from a call to mysql_fetch_row(), pointers to the values are accessed as row[0] to row[mysql_num_fields(result)-1]. NULL values in the row are indicated by NULL pointers.

The lengths of the field values in the row may be obtained by calling mysql_fetch_lengths(). Empty fields and fields containing NULL both have length 0; you can distinguish these by checking the pointer for the field value. If the pointer is NULL, the field is NULL; otherwise, the field is empty.

Return Values

A MYSQL_ROW structure for the next row. NULL if there are no more rows to retrieve or if an error occurred.

Errors

Note that error is not reset between calls to mysql_fetch_row()

- CR_SERVER_LOST

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

Example

```
MYSQL_ROW row;
unsigned int num_fields;
unsigned int i;

num_fields = mysql_num_fields(result);
while ((row = mysql_fetch_row(result)))
{
   unsigned long *lengths;
   lengths = mysql_fetch_lengths(result);
   for(i = 0; i < num_fields; i++)
   {
       printf("[%.*s] ", (int) lengths[i], row[i] ? row[i] : "NULL");
   }
   printf("\n");
}
```

22.2.3.20. mysql_field_count()

```
unsigned int mysql_field_count(MYSQL *mysql)
```

If you are using a version of MySQL earlier than Version 3.22.24, you should use `unsigned int mysql_num_fields(MYSQL *mysql)` instead.

Description

Returns the number of columns for the most recent query on the connection.

The normal use of this function is when `mysql_store_result()` returned `NULL` (and thus you have no result set pointer). In this case, you can call `mysql_field_count()` to determine whether `mysql_store_result()` should have produced a non-empty result. This allows the client program to take proper action without knowing whether the query was a `SELECT` (or `SELECT`-like) statement. The example shown here illustrates how this may be done.

See Section 22.2.13.1, “Why `mysql_store_result()` Sometimes Returns `NULL` After `mysql_query()` Returns Success”.

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

Example

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;
```

```
if (mysql_query(&mysql,query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result)  // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else  // mysql_store_result() returned nothing; should it have?
    {
        if(mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
        else // mysql_store_result() should have returned data
        {
            fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
        }
    }
}
```

An alternative is to replace the `mysql_field_count(&mysql)` call with `mysql_errno(&mysql)`. In this case, you are checking directly for an error from `mysql_store_result()` rather than inferring from the value of `mysql_field_count()` whether the statement was a `SELECT`.

22.2.3.21. mysql_field_seek()

```
MYSQL_FIELD_OFFSET         mysql_field_seek(MYSQL_RES         *result,
MYSQL_FIELD_OFFSET offset)
```

Description

Sets the field cursor to the given offset. The next call to `mysql_fetch_field()` retrieves the field definition of the column associated with that offset.

To seek to the beginning of a row, pass an `offset` value of zero.

Return Values

The previous value of the field cursor.

Errors

None.

22.2.3.22. mysql_field_tell()

```
MYSQL_FIELD_OFFSET mysql_field_tell(MYSQL_RES *result)
```

Description

Returns the position of the field cursor used for the last `mysql_fetch_field()`. This value can be

used as an argument to `mysql_field_seek()`.

Return Values

The current offset of the field cursor.

Errors

None.

22.2.3.23. `mysql_free_result()`

```
void mysql_free_result(MYSQL_RES *result)
```

Description

Frees the memory allocated for a result set by `mysql_store_result()`, `mysql_use_result()`, `mysql_list_dbs()`, etc. When you are done with a result set, you must free the memory it uses by calling `mysql_free_result()`.

Do not attempt to access a result set after freeing it.

Return Values

None.

Errors

None.

22.2.3.24. `mysql_get_client_info()`

```
char *mysql_get_client_info(void)
```

Description

Returns a string that represents the client library version.

Return Values

A character string that represents the MySQL client library version.

Errors

None.

22.2.3.25. `mysql_get_client_version()`

```
unsigned long mysql_get_client_version(void)
```

Description

Returns an integer that represents the client library version. The value has the format `XYYZZ` where `X` is the major version, `YY` is the release level, and `ZZ` is the version number within the release level. For example, a value of `40102` represents a client library version of `4.1.2`.

This function was added in MySQL 4.0.16.

Return Values

An integer that represents the MySQL client library version.

Errors

None.

22.2.3.26. mysql_get_host_info()

```
char *mysql_get_host_info(MYSQL *mysql)
```

Description

Returns a string describing the type of connection in use, including the server hostname.

Return Values

A character string representing the server hostname and the connection type.

Errors

None.

22.2.3.27. mysql_get_proto_info()

```
unsigned int mysql_get_proto_info(MYSQL *mysql)
```

Description

Returns the protocol version used by current connection.

Return Values

An unsigned integer representing the protocol version used by the current connection.

Errors

None.

22.2.3.28. mysql_get_server_info()

```
char *mysql_get_server_info(MYSQL *mysql)
```

Description

Returns a string that represents the server version number.

Return Values

A character string that represents the server version number.

Errors

None.

22.2.3.29. mysql_get_server_version()

```
unsigned long mysql_get_server_version(MYSQL *mysql)
```

Description

Returns the version number of the server as an integer.

This function was added in MySQL 4.1.0.

Return Values

A number that represents the MySQL server version in this format:

```
major_version*10000 + minor_version *100 + sub_version
```

For example, 4.1.2 is returned as 40102.

This function is useful in client programs for quickly determining whether some version-specific server capability exists.

Errors

None.

22.2.3.30. mysql_hex_string()

```
unsigned long mysql_hex_string(char *to, const char *from, unsigned long length)
```

Description

This function is used to create a legal SQL string that you can use in a SQL statement. See Section 9.1.1, “Strings”.

The string in `from` is encoded to hexadecimal format, with each character encoded as two hexadecimal digits. The result is placed in `to` and a terminating null byte is appended.

The string pointed to by `from` must be `length` bytes long. You must allocate the `to` buffer to be at least `length*2+1` bytes long. When `mysql_hex_string()` returns, the contents of `to` is a null-terminated string. The return value is the length of the encoded string, not including the terminating null character.

The return value can be placed into an SQL statement using either `0x`*`value`* or `X'`*`value`*`'` format. However, the return value does not include the `0x` or `X'...'`. The caller must supply whichever of those is desired.

`mysql_hex_string()` was added in MySQL 4.0.23 and 4.1.8.

Example

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table values(");
end = strmov(end,"0x");
end += mysql_hex_string(end,"What's this",11);
end = strmov(end,",0x");
end += mysql_hex_string(end,"binary data: \0\r\n",16);
*end++ = ')';

if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
   fprintf(stderr, "Failed to insert row, Error: %s\n",
```

```
            mysql_error(&mysql));
}
```

The `strmov()` function used in the example is included in the `mysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

Return Values

The length of the value placed into `to`, not including the terminating null character.

Errors

None.

22.2.3.31. `mysql_info()`

```
char *mysql_info(MYSQL *mysql)
```

Description

Retrieves a string providing information about the most recently executed query, but only for the statements listed here. For other statements, `mysql_info()` returns `NULL`. The format of the string varies depending on the type of query, as described here. The numbers are illustrative only; the string contains values appropriate for the query.

- `INSERT INTO ... SELECT ...`

  String format: `Records: 100 Duplicates: 0 Warnings: 0`

- `INSERT INTO ... VALUES (...),(...),(...)...`

  String format: `Records: 3 Duplicates: 0 Warnings: 0`

- `LOAD DATA INFILE ...`

  String format: `Records: 1 Deleted: 0 Skipped: 0 Warnings: 0`

- `ALTER TABLE`

  String format: `Records: 3 Duplicates: 0 Warnings: 0`

- `UPDATE`

  String format: `Rows matched: 40 Changed: 40 Warnings: 0`

Note that `mysql_info()` returns a non-`NULL` value for `INSERT ... VALUES` only for the multiple-row form of the statement (that is, only if multiple value lists are specified).

Return Values

A character string representing additional information about the most recently executed query. `NULL` if no information is available for the query.

Errors

None.

22.2.3.32. mysql_init()

```
MYSQL *mysql_init(MYSQL *mysql)
```

Description

Allocates or initializes a MYSQL object suitable for mysql_real_connect(). If mysql is a NULL pointer, the function allocates, initializes, and returns a new object. Otherwise, the object is initialized and the address of the object is returned. If mysql_init() allocates a new object, it is freed when mysql_close() is called to close the connection.

Return Values

An initialized MYSQL* handle. NULL if there was insufficient memory to allocate a new object.

Errors

In case of insufficient memory, NULL is returned.

22.2.3.33. mysql_insert_id()

```
my_ulonglong mysql_insert_id(MYSQL *mysql)
```

Description

Returns the value generated for an AUTO_INCREMENT column by the previous INSERT or UPDATE statement. Use this function after you have performed an INSERT statement into a table that contains an AUTO_INCREMENT field.

More precisely, mysql_insert_id() is updated under these conditions:

- INSERT statements that store a value into an AUTO_INCREMENT column. This is true whether the value is automatically generated by storing the special values NULL or 0 into the column, or is an explicit non-special value.
- In the case of a multiple-row INSERT statement, mysql_insert_id() returns the **first** automatically generated AUTO_INCREMENT value; if no such value is generated, it returns the last **last** explicit value inserted into the AUTO_INCREMENT column.
- INSERT statements that generate an AUTO_INCREMENT value by inserting LAST_INSERT_ID(*expr*) into any column.
- INSERT statements that generate an AUTO_INCREMENT value by updating any column to LAST_INSERT_ID(*expr*).
- The value of mysql_insert_id() is not affected by statements such as SELECT that return a result set.
- If the previous statement returned an error, the value of mysql_insert_id() is undefined.

Note that mysql_insert_id() returns 0 if the previous statement does not use an AUTO_INCREMENT value. If you need to save the value for later, be sure to call mysql_insert_id() immediately after the statement that generates the value.

The value of mysql_insert_id() is affected only by statements issued within the current client connection. It is not affected by statements issued by other clients.

See Section 12.8.3, “Information Functions”.

Also note that the value of the SQL `LAST_INSERT_ID()` function always contains the most recently generated `AUTO_INCREMENT` value, and is not reset between statements because the value of that function is maintained in the server. Another difference is that `LAST_INSERT_ID()` is not updated if you set an `AUTO_INCREMENT` column to a specific non-special value.

The reason for the difference between `LAST_INSERT_ID()` and `mysql_insert_id()` is that `LAST_INSERT_ID()` is made easy to use in scripts while `mysql_insert_id()` tries to provide a little more exact information of what happens to the `AUTO_INCREMENT` column.

Return Values

Described in the preceding discussion.

Errors

None.

22.2.3.34. `mysql_kill()`

```
int mysql_kill(MYSQL *mysql, unsigned long pid)
```

Description

Asks the server to kill the thread specified by `pid`.

Return Values

Zero for success. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

22.2.3.35. `mysql_library_init()`

```
int mysql_library_init(int argc, char **argv, char **groups)
```

Description

This is a synonym for the `mysql_server_init()` function. It was added in MySQL 4.1.10 and 5.0.3.

See Section 22.2.2, “C API Function Overview” for usage information.

22.2.3.36. mysql_library_end()

```
void mysql_library_end(void)
```

Description

This is a synonym for the `mysql_server_end()` function. It was added in MySQL 4.1.10 and 5.0.3.

See Section 22.2.2, “C API Function Overview” for usage information.

22.2.3.37. mysql_list_dbs()

```
MYSQL_RES *mysql_list_dbs(MYSQL *mysql, const char *wild)
```

Description

Returns a result set consisting of database names on the server that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters '%' or '_', or may be a `NULL` pointer to match all databases. Calling `mysql_list_dbs()` is similar to executing the query `SHOW databases [LIKE wild]`.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

22.2.3.38. mysql_list_fields()

```
MYSQL_RES *mysql_list_fields(MYSQL *mysql, const char *table, const
char *wild)
```

Description

Returns a result set consisting of field names in the given table that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters '%' or '_', or may be a `NULL` pointer to match all fields. Calling `mysql_list_fields()` is similar to executing the query `SHOW COLUMNS FROM tbl_name [LIKE wild]`.

Note that it's recommended that you use `SHOW COLUMNS FROM tbl_name` instead of `mysql_list_fields()`.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

22.2.3.39. `mysql_list_processes()`

```
MYSQL_RES *mysql_list_processes(MYSQL *mysql)
```

Description

Returns a result set describing the current server threads. This is the same kind of information as that reported by **mysqladmin processlist** or a `SHOW PROCESSLIST` query.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- CR_SERVER_LOST

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

22.2.3.40. mysql_list_tables()

```
MYSQL_RES *mysql_list_tables(MYSQL *mysql, const char *wild)
```

Description

Returns a result set consisting of table names in the current database that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters '%' or '_', or may be a `NULL` pointer to match all tables. Calling `mysql_list_tables()` is similar to executing the query `SHOW tables [LIKE wild]`.

You must free the result set with `mysql_free_result()`.

Return Values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

  Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

  The MySQL server has gone away.

- CR_SERVER_LOST

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

22.2.3.41. mysql_num_fields()

```
unsigned int mysql_num_fields(MYSQL_RES *result)
```

Or:

```
unsigned int mysql_num_fields(MYSQL *mysql)
```

The second form doesn't work on MySQL 3.22.24 or newer. To pass a `MYSQL*` argument, you must use `unsigned int mysql_field_count(MYSQL *mysql)` instead.

Description

Returns the number of columns in a result set.

Note that you can get the number of columns either from a pointer to a result set or to a connection handle. You would use the connection handle if `mysql_store_result()` or `mysql_use_result()` returned `NULL` (and thus you have no result set pointer). In this case, you can call `mysql_field_count()` to determine whether `mysql_store_result()` should have produced a non-empty result. This allows the client program to take proper action without knowing whether or not the query was a `SELECT` (or `SELECT`-like) statement. The example shown here illustrates how this may be done.

See Section 22.2.13.1, "Why `mysql_store_result()` Sometimes Returns `NULL` After `mysql_query()` Returns Success".

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

Example

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql,query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result)  // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else  // mysql_store_result() returned nothing; should it have?
    {
        if (mysql_errno(&mysql))
        {
           fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
        }
        else if (mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
    }
}
```

An alternative (if you know that your query should have returned a result set) is to replace the `mysql_errno(&mysql)` call with a check whether `mysql_field_count(&mysql)` is = 0. This happens only if something went wrong.

22.2.3.42. `mysql_num_rows()`

```
my_ulonglong mysql_num_rows(MYSQL_RES *result)
```

Description

Returns the number of rows in the result set.

The use of `mysql_num_rows()` depends on whether you use `mysql_store_result()` or `mysql_use_result()` to return the result set. If you use `mysql_store_result()`, `mysql_num_rows()` may be called immediately. If you use `mysql_use_result()`, `mysql_num_rows()` does not return the correct value until all the rows in the result set have been retrieved.

Return Values

The number of rows in the result set.

Errors

None.

22.2.3.43. `mysql_options()`

```
int mysql_options(MYSQL *mysql, enum mysql_option option, const char
*arg)
```

Description

Can be used to set extra connect options and affect behavior for a connection. This function may be called multiple times to set several options.

`mysql_options()` should be called after `mysql_init()` and before `mysql_connect()` or `mysql_real_connect()`.

The `option` argument is the option that you want to set; the `arg` argument is the value for the option. If the option is an integer, then `arg` should point to the value of the integer.

Possible option values:

Option	Argument Type	Function
`MYSQL_INIT_COMMAND`	`char *`	Command to execute when connecting to the MySQL server. Will automatically be re-executed when reconnecting.
`MYSQL_OPT_COMPRESS`	Not used	Use the compressed client/server protocol.
`MYSQL_OPT_CONNECT_TIMEOUT`	`unsigned int *`	Connect timeout in seconds.
`MYSQL_OPT_LOCAL_INFILE`	optional pointer to uint	If no pointer is given or if pointer points to an `unsigned int != 0` the command `LOAD LOCAL INFILE` is enabled.
`MYSQL_OPT_NAMED_PIPE`	Not used	Use named pipes to connect to a MySQL server on NT.
`MYSQL_OPT_PROTOCOL`	`unsigned int *`	Type of protocol to use. Should be one of the enum values of `mysql_protocol_type` defined

		in mysql.h. New in 4.1.0.
MYSQL_OPT_READ_TIMEOUT	unsigned int *	Timeout for reads from server (works currently only on Windows on TCP/IP connections). New in 4.1.1.
MYSQL_OPT_WRITE_TIMEOUT	unsigned int *	Timeout for writes to server (works currently only on Windows on TCP/IP connections). New in 4.1.1.
MYSQL_READ_DEFAULT_FILE	char *	Read options from the named option file instead of from my.cnf.
MYSQL_READ_DEFAULT_GROUP	char *	Read options from the named group from my.cnf or the file specified with MYSQL_READ_DEFAULT_FILE.
MYSQL_REPORT_DATA_TRUNCATION	my_bool *	Enable or disable reporting of data truncation errors for prepared statements via MYSQL_BIND.error. (Default: disabled) New in 5.0.3.
MYSQL_SECURE_AUTH	my_bool*	Whether to connect to a server that does not support the new 4.1.1 password hashing. New in 4.1.1.
MYSQL_SET_CHARSET_DIR	char*	The pathname to the directory that contains character set definition files.
MYSQL_SET_CHARSET_NAME	char*	The name of the character set to use as the default character set.
MYSQL_SHARED_MEMORY_BASE_NAME	char*	Named of shared memory object for communication to server. Should be same as the option -shared-memory-base-name used for the mysqld server you want's to connect to. New in 4.1.0.

Note that the client group is always read if you use MYSQL_READ_DEFAULT_FILE or MYSQL_READ_DEFAULT_GROUP.

The specified group in the option file may contain the following options:

Option	Description
connect-timeout	Connect timeout in seconds. On Linux this timeout is also used for waiting for the first answer from the server.
compress	Use the compressed client/server protocol.
database	Connect to this database if no database was specified in the connect command.
debug	Debug options.
disable-local-infile	Disable use of LOAD DATA LOCAL.
host	Default hostname.
init-command	Command to execute when connecting to MySQL server. Will automatically be re-executed when reconnecting.
interactive-timeout	Same as specifying CLIENT_INTERACTIVE to mysql_real_connect(). See Section 22.2.3.46,

	`mysql_real_connect()`”.
`local-in-file[=(0|1)]`	If no argument or argument != 0 then enable use of `LOAD DATA LOCAL`.
`max_allowed_packet`	Max size of packet client can read from server.
`multi-results`	Allow multiple result sets from multiple-statement executions or stored procedures. New in 4.1.1.
`multi-statements`	Allow the client to send multiple statements in a single string (separated by `';'`). New in 4.1.9.
`password`	Default password.
`pipe`	Use named pipes to connect to a MySQL server on NT.
`protocol={TCP | SOCKET | PIPE | MEMORY}`	The protocol to use when connecting to server (New in 4.1)
`port`	Default port number.
`return-found-rows`	Tell `mysql_info()` to return found rows instead of updated rows when using `UPDATE`.
`shared-memory-base-name=`*`me`*	Shared memory name to use to connect to server (default is "MYSQL"). New in MySQL 4.1.
`socket`	Default socket file.
`user`	Default user.

Note that `timeout` has been replaced by `connect-timeout`, but `timeout` still works for a while.

For more information about option files, see Section 4.3.2, “Using Option Files”.

Return Values

Zero for success. Non-zero if you used an unknown option.

Example

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql,MYSQL_OPT_COMPRESS,0);
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"odbc");
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
          mysql_error(&mysql));
}
```

This code requests the client to use the compressed client/server protocol and read the additional options from the `odbc` section in the `my.cnf` file.

22.2.3.44. `mysql_ping()`

```
int mysql_ping(MYSQL *mysql)
```

Description

Checks whether the connection to the server is working. If it has gone down, an automatic reconnection is attempted.

This function can be used by clients that remain idle for a long while, to check whether the server has closed the connection and reconnect if necessary.

Return Values

Zero if the server is alive. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

22.2.3.45. `mysql_query()`

```
int mysql_query(MYSQL *mysql, const char *query)
```

Description

Executes the SQL query pointed to by the null-terminated string `query`. Normally, the string must consist of a single SQL statement and you should not add a terminating semicolon (`';'`) or `\g` to the statement. If multiple-statement execution has been enabled, the string can contain several statements separated by semicolons. See Section 22.2.9, “C API Handling of Multiple Query Execution”.

`mysql_query()` cannot be used for queries that contain binary data; you should use `mysql_real_query()` instead. (Binary data may contain the `'\0'` character, which `mysql_query()` interprets as the end of the query string.)

If you want to know whether the query should return a result set, you can use `mysql_field_count()` to check for this. See Section 22.2.3.20, “`mysql_field_count()`”.

Return Values

Zero if the query was successful. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- CR_SERVER_LOST

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

22.2.3.46. mysql_real_connect()

```
MYSQL *mysql_real_connect(MYSQL *mysql, const char *host, const char
*user, const char *passwd, const char *db, unsigned int port, const
char *unix_socket, unsigned long client_flag)
```

Description

mysql_real_connect() attempts to establish a connection to a MySQL database engine running on host. mysql_real_connect() must complete successfully before you can execute any other API functions that require a valid MYSQL connection handle structure.

The parameters are specified as follows:

- The first parameter should be the address of an existing MYSQL structure. Before calling mysql_real_connect() you must call mysql_init() to initialize the MYSQL structure. You can change a lot of connect options with the mysql_options() call. See Section 22.2.3.43, “mysql_options()”.

- The value of host may be either a hostname or an IP address. If host is NULL or the string "localhost", a connection to the local host is assumed. If the OS supports sockets (Unix) or named pipes (Windows), they are used instead of TCP/IP to connect to the server.

- The user parameter contains the user's MySQL login ID. If user is NULL or the empty string "", the current user is assumed. Under Unix, this is the current login name. Under Windows ODBC, the current username must be specified explicitly. See Section 23.1.9.2, “Configuring a MyODBC DSN on Windows”.

- The passwd parameter contains the password for user. If passwd is NULL, only entries in the user table for the user that have a blank (empty) password field are checked for a match. This allows the database administrator to set up the MySQL privilege system in such a way that users get different privileges depending on whether or not they have specified a password.

  Note: Do not attempt to encrypt the password before calling mysql_real_connect(); password encryption is handled automatically by the client API.

- db is the database name. If db is not NULL, the connection sets the default database to this value.

- If port is not 0, the value is used as the port number for the TCP/IP connection. Note that the host parameter determines the type of the connection.

- If unix_socket is not NULL, the string specifies the socket or named pipe that should be used. Note that the host parameter determines the type of the connection.

- The value of client_flag is usually 0, but can be set to a combination of the following flags in very special circumstances:

Flag Name	Flag Description

CLIENT_COMPRESS	Use compression protocol.
CLIENT_FOUND_ROWS	Return the number of found (matched) rows, not the number of affected rows.
CLIENT_IGNORE_SPACE	Allow spaces after function names. Makes all functions names reserved words.
CLIENT_INTERACTIVE	Allow interactive_timeout seconds (instead of wait_timeout seconds) of inactivity before closing the connection. The client's session wait_timeout variable is set to the value of the session interactive_timeout variable.
CLIENT_LOCAL_FILES	Enable LOAD DATA LOCAL handling.
CLIENT_MULTI_STATEMENTS	Tell the server that the client may send multiple statements in a single string (separated by ';'). If this flag is not set, multiple-statement execution is disabled. New in 4.1.
CLIENT_MULTI_RESULTS	Tell the server that the client can handle multiple result sets from multiple-statement executions or stored procedures. This is automatically set if CLIENT_MULTI_STATEMENTS is set. New in 4.1.
CLIENT_NO_SCHEMA	Don't allow the *db_name.tbl_name.col_name* syntax. This is for ODBC. It causes the parser to generate an error if you use that syntax, which is useful for trapping bugs in some ODBC programs.
CLIENT_ODBC	The client is an ODBC client. This changes **mysqld** to be more ODBC-friendly.
CLIENT_SSL	Use SSL (encrypted protocol). This option should not be set by application programs; it is set internally in the client library.

Return Values

A MYSQL* connection handle if the connection was successful, NULL if the connection was unsuccessful. For a successful connection, the return value is the same as the value of the first parameter.

Errors

- CR_CONN_HOST_ERROR

  Failed to connect to the MySQL server.

- CR_CONNECTION_ERROR

  Failed to connect to the local MySQL server.

- CR_IPSOCK_ERROR

  Failed to create an IP socket.

- CR_OUT_OF_MEMORY

  Out of memory.

- CR_SOCKET_CREATE_ERROR

Failed to create a Unix socket.

- CR_UNKNOWN_HOST

  Failed to find the IP address for the hostname.

- CR_VERSION_ERROR

  A protocol mismatch resulted from attempting to connect to a server with a client library that uses a different protocol version. This can happen if you use a very old client library to connect to a new server that wasn't started with the --old-protocol option.

- CR_NAMEDPIPEOPEN_ERROR

  Failed to create a named pipe on Windows.

- CR_NAMEDPIPEWAIT_ERROR

  Failed to wait for a named pipe on Windows.

- CR_NAMEDPIPESETSTATE_ERROR

  Failed to get a pipe handler on Windows.

- CR_SERVER_LOST

  If connect_timeout > 0 and it took longer than connect_timeout seconds to connect to the server or if the server died while executing the init-command.

Example

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"your_prog_name");
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
          mysql_error(&mysql));
}
```

By using mysql_options() the MySQL library reads the [client] and [your_prog_name] sections in the my.cnf file which ensures that your program works, even if someone has set up MySQL in some non-standard way.

Note that upon connection, mysql_real_connect() sets the reconnect flag (part of the MYSQL structure) to a value of 1 in versions of the API strictly older than 5.0.3, of 0 in newer versions. A value of 1 for this flag indicates, in the event that a query cannot be performed because of a lost connection, to try reconnecting to the server before giving up.

22.2.3.47. mysql_real_escape_string()

```
unsigned long mysql_real_escape_string(MYSQL *mysql, char *to, const
char *from, unsigned long length)
```

Note that mysql must be a valid, open connection. This is needed because the escaping depends on the character-set in use by the server.

Description

This function is used to create a legal SQL string that you can use in a SQL statement. See Section 9.1.1, “Strings”.

The string in `from` is encoded to an escaped SQL string, taking into account the current character set of the connection. The result is placed in `to` and a terminating null byte is appended. Characters encoded are `NUL` (ASCII 0), '`\n`', '`\r`', '`\`', '`'`', '`"`', and Control-Z (see Section 9.1, “Literal Values”). (Strictly speaking, MySQL requires only that backslash and the quote character used to quote the string in the query be escaped. This function quotes the other characters to make them easier to read in log files.)

The string pointed to by `from` must be `length` bytes long. You must allocate the `to` buffer to be at least `length*2+1` bytes long. (In the worst case, each character may need to be encoded as using two bytes, and you need room for the terminating null byte.) When `mysql_real_escape_string()` returns, the contents of `to` is a null-terminated string. The return value is the length of the encoded string, not including the terminating null character.

Example

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table values(");
*end++ = '\'';
end += mysql_real_escape_string(&mysql, end,"What's this",11);
*end++ = '\'';
*end++ = ',';
*end++ = '\'';
end += mysql_real_escape_string(&mysql, end,"binary data: \0\r\n",16);
*end++ = '\'';
*end++ = ')';

if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
   fprintf(stderr, "Failed to insert row, Error: %s\n",
           mysql_error(&mysql));
}
```

The `strmov()` function used in the example is included in the `mysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

Return Values

The length of the value placed into `to`, not including the terminating null character.

Errors

None.

22.2.3.48. `mysql_real_query()`

`int mysql_real_query(MYSQL *mysql, const char *query, unsigned long length)`

Description

Executes the SQL query pointed to by `query`, which should be a string `length` bytes long. Normally, the string must consist of a single SQL statement and you should not add a terminating semicolon ('`;`') or `\g` to the statement. If multiple-statement execution has been enabled, the string can contain several statements separated by semicolons. See Section 22.2.9, “C API Handling of Multiple Query

.

You **must** use `mysql_real_query()` rather than `mysql_query()` for queries that contain binary data, because binary data may contain the '`\0`' character. In addition, `mysql_real_query()` is faster than `mysql_query()` because it does not call `strlen()` on the query string.

If you want to know whether the query should return a result set, you can use `mysql_field_count()` to check for this. See Section 22.2.3.20, "`mysql_field_count()`".

Return Values

Zero if the query was successful. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

22.2.3.49. `mysql_reload()`

```
int mysql_reload(MYSQL *mysql)
```

Description

Asks the MySQL server to reload the grant tables. The connected user must have the `RELOAD` privilege.

This function is deprecated. It is preferable to use `mysql_query()` to issue an SQL `FLUSH PRIVILEGES` statement instead.

Return Values

Zero for success. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

22.2.3.50. mysql_row_seek()

```
MYSQL_ROW_OFFSET mysql_row_seek(MYSQL_RES *result, MYSQL_ROW_OFFSET offset)
```

Description

Sets the row cursor to an arbitrary row in a query result set. The `offset` value is a row offset that should be a value returned from `mysql_row_tell()` or from `mysql_row_seek()`. This value is not a row number; if you want to seek to a row within a result set by number, use `mysql_data_seek()` instead.

This function requires that the result set structure contains the entire result of the query, so `mysql_row_seek()` may be used only in conjunction with `mysql_store_result()`, not with `mysql_use_result()`.

Return Values

The previous value of the row cursor. This value may be passed to a subsequent call to `mysql_row_seek()`.

Errors

None.

22.2.3.51. mysql_row_tell()

```
MYSQL_ROW_OFFSET mysql_row_tell(MYSQL_RES *result)
```

Description

Returns the current position of the row cursor for the last `mysql_fetch_row()`. This value can be used as an argument to `mysql_row_seek()`.

You should use `mysql_row_tell()` only after `mysql_store_result()`, not after `mysql_use_result()`.

Return Values

The current offset of the row cursor.

Errors

None.

22.2.3.52. mysql_select_db()

```
int mysql_select_db(MYSQL *mysql, const char *db)
```

Description

Causes the database specified by `db` to become the default (current) database on the connection specified by `mysql`. In subsequent queries, this database is the default for table references that do not include an explicit database specifier.

`mysql_select_db()` fails unless the connected user can be authenticated as having permission to use the database.

Return Values

Zero for success. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

22.2.3.53. `mysql_set_server_option()`

```
int mysql_set_server_option(MYSQL *mysql, enum enum_mysql_set_option option)
```

Description

Enables or disables an option for the connection. `option` can have one of the following values:

MYSQL_OPTION_MULTI_STATEMENTS_ON	Enable multi statement support.
MYSQL_OPTION_MULTI_STATEMENTS_OFF	Disable multi statement support.

This function was added in MySQL 4.1.1.

Return Values

Zero for success. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

  The MySQL server has gone away.

- CR_SERVER_LOST

  The connection to the server was lost during the query.

- ER_UNKNOWN_COM_ERROR

  The server didn't support mysql_set_server_option() (which is the case that the server is older than 4.1.1) or the server didn't support the option one tried to set.

22.2.3.54. mysql_shutdown()

int mysql_shutdown(MYSQL *mysql, enum enum_shutdown_level shutdown_level)

Description

Asks the database server to shut down. The connected user must have SHUTDOWN privileges. The shutdown_level argument was added in MySQL 4.1.3 (and 5.0.1). The MySQL server currently supports only one type (level of gracefulness) of shutdown; shutdown_level must be equal to SHUTDOWN_DEFAULT. Later we wil add more levels and then the shutdown_level argument will enable us to choose the desired level. MySQL servers and MySQL clients before and after 4.1.3 are compatible; MySQL servers newer than 4.1.3 accept the mysql_shutdown(MYSQL *mysql) call, and MySQL servers older than 4.1.3 accept the new mysql_shutdown() call. But dynamically linked executables which have been compiled with older versions of libmysqlclient headers, and call mysql_shutdown(), need to be used with the old libmysqlclient dynamic library.

The shutdown process is described in Section 5.3, “The MySQL Server Shutdown Process”.

Return Values

Zero for success. Non-zero if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

  Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

  The MySQL server has gone away.

- CR_SERVER_LOST

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

22.2.3.55. mysql_sqlstate()

```
const char *mysql_sqlstate(MYSQL *mysql)
```

Description

Returns a null-terminated string containing the SQLSTATE error code for the last error. The error code consists of five characters. `'00000'` means ``no error.'' The values are specified by ANSI SQL and ODBC. For a list of possible values, see Chapter 24, *Error Handling in MySQL*.

Note that not all MySQL errors are yet mapped to SQLSTATE's. The value `'HY000'` (general error) is used for unmapped errors.

This function was added to MySQL 4.1.1.

Return Values

A null-terminated character string containing the SQLSTATE error code.

See Also

See Section 22.2.3.12, "`mysql_errno()`". See Section 22.2.3.13, "`mysql_error()`". See Section 22.2.7.26, "`mysql_stmt_sqlstate()`".

22.2.3.56. `mysql_ssl_set()`

```
int mysql_ssl_set(MYSQL *mysql, const char *key, const char *cert,
const char *ca, const char *capath, const char *cipher)
```

Description

`mysql_ssl_set()` is used for establishing secure connections using SSL. It must be called before `mysql_real_connect()`.

`mysql_ssl_set()` does nothing unless OpenSSL support is enabled in the client library.

`mysql` is the connection handler returned from `mysql_init()`. The other parameters are specified as follows:

- `key` is the pathname to the key file.
- `cert` is the pathname to the certificate file.
- `ca` is the pathname to the certificate authority file.
- `capath` is the pathname to a directory that contains trusted SSL CA certificates in pem format.
- `cipher` is a list of allowable ciphers to use for SSL encryption.

Any unused SSL parameters may be given as `NULL`.

Return Values

This function always returns `0`. If SSL setup is incorrect, `mysql_real_connect()` returns an error when you attempt to connect.

22.2.3.57. `mysql_stat()`

```
char *mysql_stat(MYSQL *mysql)
```

Description

Returns a character string containing information similar to that provided by the **mysqladmin status** command. This includes uptime in seconds and the number of running threads, questions, reloads, and open tables.

Return Values

A character string describing the server status. `NULL` if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

22.2.3.58. `mysql_store_result()`

```
MYSQL_RES *mysql_store_result(MYSQL *mysql)
```

Description

You must call `mysql_store_result()` or `mysql_use_result()` for every query that successfully retrieves data (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`, `CHECK TABLE`, and so forth).

You don't have to call `mysql_store_result()` or `mysql_use_result()` for other queries, but it does not do any harm or cause any notable performance degredation if you call `mysql_store_result()` in all cases. You can detect if the query didn't have a result set by checking if `mysql_store_result()` returns 0 (more about this later on).

If you want to know whether the query should return a result set, you can use `mysql_field_count()` to check for this. See Section 22.2.3.20, “`mysql_field_count()`”.

`mysql_store_result()` reads the entire result of a query to the client, allocates a `MYSQL_RES` structure, and places the result into this structure.

`mysql_store_result()` returns a null pointer if the query didn't return a result set (if the query was, for example, an `INSERT` statement).

`mysql_store_result()` also returns a null pointer if reading of the result set failed. You can check whether an error occurred by checking if `mysql_error()` returns a non-empty string, if `mysql_errno()` returns non-zero, or if `mysql_field_count()` returns zero.

An empty result set is returned if there are no rows returned. (An empty result set differs from a null pointer as a return value.)

Once you have called `mysql_store_result()` and got a result back that isn't a null pointer, you may call `mysql_num_rows()` to find out how many rows are in the result set.

You can call `mysql_fetch_row()` to fetch rows from the result set, or `mysql_row_seek()` and `mysql_row_tell()` to obtain or set the current row position within the result set.

You must call `mysql_free_result()` once you are done with the result set.

See Section 22.2.13.1, "Why `mysql_store_result()` Sometimes Returns `NULL` After `mysql_query()` Returns Success".

Return Values

A `MYSQL_RES` result structure with the results. `NULL` if an error occurred.

Errors

`mysql_store_result()` resets `mysql_error()` and `mysql_errno()` if it succeeds.

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

22.2.3.59. `mysql_thread_id()`

```
unsigned long mysql_thread_id(MYSQL *mysql)
```

Description

Returns the thread ID of the current connection. This value can be used as an argument to `mysql_kill()` to kill the thread.

If the connection is lost and you reconnect with `mysql_ping()`, the thread ID changes. This means you should not get the thread ID and store it for later. You should get it when you need it.

Return Values

The thread ID of the current connection.

Errors

None.

22.2.3.60. mysql_use_result()

```
MYSQL_RES *mysql_use_result(MYSQL *mysql)
```

Description

You must call `mysql_store_result()` or `mysql_use_result()` for every query that successfully retrieves data (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`).

`mysql_use_result()` initiates a result set retrieval but does not actually read the result set into the client like `mysql_store_result()` does. Instead, each row must be retrieved individually by making calls to `mysql_fetch_row()`. This reads the result of a query directly from the server without storing it in a temporary table or local buffer, which is somewhat faster and uses much less memory than `mysql_store_result()`. The client allocates memory only for the current row and a communication buffer that may grow up to `max_allowed_packet` bytes.

On the other hand, you shouldn't use `mysql_use_result()` if you are doing a lot of processing for each row on the client side, or if the output is sent to a screen on which the user may type a `^S` (stop scroll). This ties up the server and prevent other threads from updating any tables from which the data is being fetched.

When using `mysql_use_result()`, you must execute `mysql_fetch_row()` until a `NULL` value is returned, otherwise, the unfetched rows are returned as part of the result set for your next query. The C API gives the error `Commands out of sync; you can't run this command now` if you forget to do this!

You may not use `mysql_data_seek()`, `mysql_row_seek()`, `mysql_row_tell()`, `mysql_num_rows()`, or `mysql_affected_rows()` with a result returned from `mysql_use_result()`, nor may you issue other queries until the `mysql_use_result()` has finished. (However, after you have fetched all the rows, `mysql_num_rows()` accurately returns the number of rows fetched.)

You must call `mysql_free_result()` once you are done with the result set.

Return Values

A `MYSQL_RES` result structure. `NULL` if an error occurred.

Errors

`mysql_use_result()` resets `mysql_error()` and `mysql_errno()` if it succeeds.

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

22.2.3.61. `mysql_warning_count()`

```
unsigned int mysql_warning_count(MYSQL *mysql)
```

Description

Returns the number of warnings generated during execution of the previous SQL statement.

This function was added in MySQL 4.1.0.

Return Values

The warning count.

Errors

None.

22.2.3.62. `mysql_commit()`

```
my_bool mysql_commit(MYSQL *mysql)
```

Description

Commits the current transaction.

This function was added in MySQL 4.1.0.

Return Values

Zero if successful. Non-zero if an error occurred.

Errors

None.

22.2.3.63. `mysql_rollback()`

```
my_bool mysql_rollback(MYSQL *mysql)
```

Description

Rolls back the current transaction.

This function was added in MySQL 4.1.0.

Return Values

Zero if successful. Non-zero if an error occurred.

Errors

None.

22.2.3.64. mysql_autocommit()

```
my_bool mysql_autocommit(MYSQL *mysql, my_bool mode)
```

Description

Sets autocommit mode on if `mode` is 1, off if `mode` is 0.

This function was added in MySQL 4.1.0.

Return Values

Zero if successful. Non-zero if an error occurred.

Errors

None.

22.2.3.65. mysql_more_results()

```
my_bool mysql_more_results(MYSQL *mysql)
```

Description

Returns true if more results exist from the currently executed query, and the application must call `mysql_next_result()` to fetch the results.

This function was added in MySQL 4.1.0.

Return Values

`TRUE` (1) if more results exist. `FALSE` (0) if no more results exist.

In most cases, you can call `mysql_next_result()` instead to test whether more results exist and initiate retrieval if so.

See Section 22.2.9, "C API Handling of Multiple Query Execution". See Section 22.2.3.66, "`mysql_next_result()`".

Errors

None.

22.2.3.66. mysql_next_result()

```
int mysql_next_result(MYSQL *mysql)
```

Description

If more query results exist, `mysql_next_result()` reads the next query results and returns the status back to application.

You must call `mysql_free_result()` for the preceding query if it returned a result set.

After calling `mysql_next_result()` the state of the connection is as if you had called `mysql_real_query()` or `mysql_query()` for the next query. This means that you can call

`mysql_store_result()`, `mysql_warning_count()`, `mysql_affected_rows()`, and so forth.

If `mysql_next_result()` returns an error, no other statements are executed and there are no more results to fetch.

See Section 22.2.9, “C API Handling of Multiple Query Execution”.

This function was added in MySQL 4.1.0.

Return Values

Return Value	Description
0	Successful and there are more results
#1	Successful and there are no more results
>0	An error occurred

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order. For example if you didn't call `mysql_use_result()` for a previous result set.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

22.2.4. C API Prepared Statements

As of MySQL 4.1, the client/server protocol provides for the use of prepared statements. This capability uses the `MYSQL_STMT` statement handler data structure returned by the `mysql_stmt_init()` initialization function. Prepared execution is an efficient way to execute a statement more than once. The statement is first parsed to prepare it for execution. Then it is executed one or more times at a later time, using the statement handle returned by the initialization function.

Prepared execution is faster than direct execution for statements executed more than once, primarily because the query is parsed only once. In the case of direct execution, the query is parsed every time it is executed. Prepared execution also can provide a reduction of network traffic because for each execution of the prepared statement, it is necessary only to send the data for the parameters.

Another advantage of prepared statements is that it uses a binary protocol that makes data transfer between client and server more efficient.

The following statements can be used as prepared statements: `CREATE TABLE`, `DELETE`, `DO`, `INSERT`, `REPLACE`, `SELECT`, `SET`, `UPDATE`, and most `SHOW` statements. Other statements are not yet supported.

22.2.5. C API Prepared Statement Data types

Note: Some incompatible changes were made in MySQL 4.1.2. See Section 22.2.7, “C API Prepared Statement Function Descriptions” for details.

Prepared statements mainly use the `MYSQL_STMT` and `MYSQL_BIND` data structures. A third structure, `MYSQL_TIME`, is used to transfer temporal data.

- `MYSQL_STMT`

  This structure represents a prepared statement. A statement is created by calling `mysql_stmt_init()`, which returns a statement handle, that is, a pointer to a `MYSQL_STMT`. The handle is used for all subsequent statement-related functions until you close it with `mysql_stmt_close()`.

  The `MYSQL_STMT` structure has no members that are for application use.

  Multiple statement handles can be associated with a single connection. The limit on the number of handles depends on the available system resources.

- `MYSQL_BIND`

  This structure is used both for statement input (data values sent to the server) and output (result values returned from the server). For input, it is used with `mysql_stmt_bind_param()` to bind parameter data values to buffers for use by `mysql_stmt_execute()`. For output, it is used with `mysql_stmt_bind_result()` to bind result set buffers for use in fetching rows with `mysql_stmt_fetch()`.

  The `MYSQL_BIND` structure contains the following members for use by application programs. Each is used both for input and for output, although sometimes for different purposes depending on the direction of data transfer.

  - `enum enum_field_types buffer_type`

    The type of the buffer. The allowable `buffer_type` values are listed later in this section. For input, `buffer_type` indicates what type of value you are binding to a statement parameter. For output, it indicates what type of value you expect to receive in a result buffer.

  - `void *buffer`

    For input, this is a pointer to the buffer in which a statement parameter's data value is stored. For output, it is a pointer to the buffer in which to return a result set column value. For numeric column types, `buffer` should point to a variable of the proper C type. (If you are associating the variable with a column that has the `UNSIGNED` attribute, the variable should be an `unsigned` C type. Indicate whether the variable is signed or unsigned by using the `is_unsigned` member, described later in this list.) For date and time column types, `buffer` should point to a `MYSQL_TIME` structure. For character and binary string column types, `buffer` should point to a character buffer.

  - `unsigned long buffer_length`

    The actual size of `*buffer` in bytes. This indicates the maximum amount of data that can be stored in the buffer. For character and binary C data, the `buffer_length` value specifies the length of `*buffer` when used with `mysql_stmt_bind_param()`, or the maximum number of data bytes that can be fetched into the buffer when used with `mysql_stmt_bind_result()`.

  - `unsigned long *length`

A pointer to an `unsigned long` variable that indicates the actual number of bytes of data stored in `*buffer`. `length` is used for character or binary C data. For input parameter data binding, `length` points to an `unsigned long` variable that indicates the length of the parameter value stored in `*buffer`; this is used by `mysql_stmt_execute()`. For output value binding, `mysql_stmt_fetch()` places the length of the column value that is returned into the variable that `length` points to.

`length` is ignored for numeric and temporal data types because the length of the data value is determined by the `buffer_type` value.

- `my_bool *is_null`

  This member points to a `my_bool` variable that is true if a value is `NULL`, false if it is not `NULL`. For input, set `*is_null` to true to indicate that you are passing a `NULL` value as a statement parameter. For output, this value is set to true after you fetch a row if the result set column value returned from the statement is `NULL`.

- `my_bool is_unsigned`

  This member is used for integer types. (These correspond to the `MYSQL_TYPE_TINY`, `MYSQL_TYPE_SHORT`, `MYSQL_TYPE_LONG`, and `MYSQL_TYPE_LONGLONG` type codes.) `is_unsigned` should be set to true for unsigned types and false for signed types.

- `my_bool error`

  For output, this member is used output to report data truncation errors. Truncation reporting must be enabled by calling `mysql_options()` with the `MYSQL_REPORT_DATA_TRUNCATION` option. When enabled, `mysql_stmt_fetch()` returns `MYSQL_DATA_TRUNCATED` and `error` is true in the `MYSQL_BIND` structures for parameters in which truncation occurred. Truncation indicates loss of sign or significant digits, or that a string was too long to fit in a column. The `error` member was added in MySQL 5.0.3.

To use a `MYSQL_BIND` structure, you should zero its contents to initialize it, and then set the members just described appropriately. For example, to declare and initialize an array of three `MYSQL_BIND` structures, use this code:

```
MYSQL_BIND    bind[3];
memset(bind, 0, sizeof(bind));
```

- `MYSQL_TIME`

  This structure is used to send and receive `DATE`, `TIME`, `DATETIME`, and `TIMESTAMP` data directly to and from the server. This is done by setting the `buffer_type` member of a `MYSQL_BIND` structure to one of the temporal types, and setting the `buffer` member to point to a `MYSQL_TIME` structure.

  The `MYSQL_TIME` structure contains the following members:

  - `unsigned int year`

    The year.

  - `unsigned int month`

The month of the year.

- `unsigned int day`

  The day of the month.

- `unsigned int hour`

  The hour of the day.

- `unsigned int minute`

  The minute of the hour.

- `unsigned int second`

  The second of the minute.

- `my_bool neg`

  A boolean flag to indicate whether the time is negative.

- `unsigned long second_part`

  The fractional part of the second. This member currently is unused.

Only those parts of a `MYSQL_TIME` structure that apply to a given type of temporal value are used: The `year`, `month`, and `day` elements are used for `DATE`, `DATETIME`, and `TIMESTAMP` values. The `hour`, `minute`, and `second` elements are used for `TIME`, `DATETIME`, and `TIMESTAMP` values. See Section 22.2.10, “C API Handling of Date and Time Values”.

The following table shows the allowable values that may be specified in the `buffer_type` member of `MYSQL_BIND` structures. The table also shows those SQL types that correspond most closely to each `buffer_type` value, and, for numeric and temporal types, the corresponding C type.

`buffer_type` **Value**	**SQL Type**	**C Type**
`MYSQL_TYPE_TINY`	`TINYINT`	`char`
`MYSQL_TYPE_SHORT`	`SMALLINT`	`short int`
`MYSQL_TYPE_LONG`	`INT`	`int`
`MYSQL_TYPE_LONGLONG`	`BIGINT`	`long long int`
`MYSQL_TYPE_FLOAT`	`FLOAT`	`float`
`MYSQL_TYPE_DOUBLE`	`DOUBLE`	`double`
`MYSQL_TYPE_TIME`	`TIME`	`MYSQL_TIME`
`MYSQL_TYPE_DATE`	`DATE`	`MYSQL_TIME`
`MYSQL_TYPE_DATETIME`	`DATETIME`	`MYSQL_TIME`
`MYSQL_TYPE_TIMESTAMP`	`TIMESTAMP`	`MYSQL_TIME`
`MYSQL_TYPE_STRING`	`CHAR`	
`MYSQL_TYPE_VAR_STRING`	`VARCHAR`	

MYSQL_TYPE_TINY_BLOB	TINYBLOB/TINYTEXT	
MYSQL_TYPE_BLOB	BLOB/TEXT	
MYSQL_TYPE_MEDIUM_BLOB	MEDIUMBLOB/MEDIUMTEXT	
MYSQL_TYPE_LONG_BLOB	LONGBLOB/LONGTEXT	

Implicit type conversion may be performed in both directions.

22.2.6. C API Prepared Statement Function Overview

Note: Some incompatible changes were made in MySQL 4.1.2. See Section 22.2.7, "C API Prepared Statement Function Descriptions" for details.

The functions available for prepared statement processing are summarized here and described in greater detail in a later section. See Section 22.2.7, "C API Prepared Statement Function Descriptions".

Function	Description
mysql_stmt_affected_rows()	Returns the number of rows changes, deleted, or inserted by prepared UPDATE, DELETE, or INSERT statement.
mysql_stmt_attr_get()	Get value of an attribute for a prepared statement.
mysql_stmt_attr_set()	Sets an attribute for a prepared statement.
mysql_stmt_bind_param()	Associates application data buffers with the parameter markers in a prepared SQL statement.
mysql_stmt_bind_result()	Associates application data buffers with columns in the result set.
mysql_stmt_close()	Frees memory used by prepared statement.
mysql_stmt_data_seek()	Seeks to an arbitrary row number in a statement result set.
mysql_stmt_errno()	Returns the error number for the last statement execution.
mysql_stmt_error()	Returns the error message for the last statement execution.
mysql_stmt_execute()	Executes the prepared statement.
mysql_stmt_fetch()	Fetches the next row of data from the result set and returns data for all bound columns.
mysql_stmt_fetch_column()	Fetch data for one column of the current row of the result set.
mysql_stmt_field_count()	Returns the number of result columns for the most recent statement.
mysql_stmt_free_result()	Free the resources allocated to the statement handle.
mysql_stmt_init()	Allocates memory for MYSQL_STMT structure and initializes it.
mysql_stmt_insert_id()	Returns the ID generated for an AUTO_INCREMENT column by prepared statement.
mysql_stmt_num_rows()	Returns total rows from the statement buffered result set.
mysql_stmt_param_count()	Returns the number of parameters in a prepared SQL statement.
mysql_stmt_param_metadata()	Return parameter metadata in the form of a result set.
mysql_stmt_prepare()	Prepares an SQL string for execution.
mysql_stmt_reset()	Reset the statement buffers in the server.
mysql_stmt_result_metadata()	Returns prepared statement metadata in the form of a result set.
mysql_stmt_row_seek()	Seeks to a row offset in a statement result set, using value returned from mysql_stmt_row_tell().

mysql_stmt_row_tell()	Returns the statement row cursor position.
mysql_stmt_send_long_data()	Sends long data in chunks to server.
mysql_stmt_sqlstate()	Returns the SQLSTATE error code for the last statement execution.
mysql_stmt_store_result()	Retrieves the complete result set to the client.

Call `mysql_stmt_init()` to create a statement handle, then `mysql_stmt_prepare` to prepare it, `mysql_stmt_bind_param()` to supply the parameter data, and `mysql_stmt_execute()` to execute the statement. You can repeat the `mysql_stmt_execute()` by changing parameter values in the respective buffers supplied through `mysql_stmt_bind_param()`.

If the statement is a `SELECT` or any other statement that produces a result set, `mysql_stmt_prepare()` also returns the result set metadata information in the form of a `MYSQL_RES` result set through `mysql_stmt_result_metadata()`.

You can supply the result buffers using `mysql_stmt_bind_result()`, so that the `mysql_stmt_fetch()` automatically returns data to these buffers. This is row-by-row fetching.

You can also send the text or binary data in chunks to server using `mysql_stmt_send_long_data()`. See Section 22.2.7.25, "`mysql_stmt_send_long_data()`".

When statement execution has been completed, the statement handle must be closed using `mysql_stmt_close()` so that all resources associated with it can be freed.

If you obtained a `SELECT` statement's result set metadata by calling `mysql_stmt_result_metadata()`, you should also free the metadata using `mysql_free_result()`.

Execution Steps

To prepare and execute a statement, an application follows these steps:

1. Create a prepared statement handle with `msyql_stmt_init()`. To prepare the statement on the server, call `mysql_stmt_prepare()` and pass it a string containing the SQL statement.

2. If the statement produces a result set, call `mysql_stmt_result_metadata()` to obtain the result set metadata. This metadata is itself in the form of result set, albeit a separate one from the one that contains the rows returned by the query. The metadata result set indicates how many columns are in the result and contains information about each column.

3. Set the values of any parameters using `mysql_stmt_bind_param()`. All parameters must be set. Otherwise, statement execution returns an error or produces unexpected results.

4. Call `mysql_stmt_execute()` to execute the statement.

5. If the statement produces a result set, bind the data buffers to use for retrieving the row values by calling `mysql_stmt_bind_result()`.

6. Fetch the data into the buffers row by row by calling `mysql_stmt_fetch()` repeatedly until no more rows are found.

7. Repeat steps 3 through 6 as necessary, by changing the parameter values and re-executing the statement.

When `mysql_stmt_prepare()` is called, the MySQL client/server protocol performs these actions:

- The server parses the statement and sends the okay status back to the client by assigning a statement ID. It also sends total number of parameters, a column count, and its metadata if it is a result set oriented statement. All syntax and semantics of the statement are checked by the server during this call.
- The client uses this statement ID for the further operations, so that the server can identify the statement from among its pool of statements.

When `mysql_stmt_execute()` is called, the MySQL client/server protocol performs these actions:

- The client uses the statement handle and sends the parameter data to the server.
- The server identifies the statement using the ID provided by the client, replaces the parameter markers with the newly supplied data, and executes the statement. If the statement produces a result set, the server sends the data back to the client. Otherwise, it sends an okay status and total number of rows changed, deleted, or inserted.

When `mysql_stmt_fetch()` is called, the MySQL client/server protocol performs these actions:

- The client reads the data from the packet row by row and places it into the application data buffers by doing the necessary conversions. If the application buffer type is same as that of the field type returned from the server, the conversions are straightforward.

If an error occurs, you can get the statement error code, error message, and SQLSTATE value using `mysql_stmt_errno()`, `mysql_stmt_error()`, and `mysql_stmt_sqlstate()`, respectively.

22.2.7. C API Prepared Statement Function Descriptions

To prepare and execute queries, use the functions in the following sections.

In MySQL 4.1.2, the names of several prepared statement functions were changed:

Old Name	New Name
`mysql_bind_param()`	`mysql_stmt_bind_param()`
`mysql_bind_result()`	`mysql_stmt_bind_result()`
`mysql_prepare()`	`mysql_stmt_prepare()`
`mysql_execute()`	`mysql_stmt_execute()`
`mysql_fetch()`	`mysql_stmt_fetch()`
`mysql_fetch_column()`	`mysql_stmt_fetch_column()`
`mysql_param_count()`	`mysql_stmt_param_count()`
`mysql_param_result()`	`mysql_stmt_param_metadata()`
`mysql_get_metadata()`	`mysql_stmt_result_metadata()`
`mysql_send_long_data()`	`mysql_stmt_send_long_data()`

All functions that operate with a `MYSQL_STMT` structure begin with the prefix `mysql_stmt_`.

Also in 4.1.2, the signature of the `mysql_stmt_prepare()` function was changed to `int`

`mysql_stmt_prepare(MYSQL_STMT *stmt, const char *query, unsigned long length)`. To create a `MYSQL_STMT` handle, you should use the `mysql_stmt_init()` function.

22.2.7.1. mysql_stmt_affected_rows()

```
my_ulonglong mysql_stmt_affected_rows(MYSQL_STMT *stmt)
```

Description

Returns the total number of rows changed, deleted, or inserted by the last executed statement. May be called immediately after `mysql_stmt_execute()` for `UPDATE`, `DELETE`, or `INSERT` statements. For `SELECT` statements, `mysql_stmt_affected_rows()` works like `mysql_num_rows()`.

This function was added in MySQL 4.1.0.

Return Values

An integer greater than zero indicates the number of rows affected or retrieved. Zero indicates that no records were updated for an `UPDATE` statement, no rows matched the `WHERE` clause in the query, or that no query has yet been executed. #1 indicates that the query returned an error or that, for a `SELECT` query, `mysql_stmt_affected_rows()` was called prior to calling `mysql_stmt_store_result()`. Because `mysql_stmt_affected_rows()` returns an unsigned value, you can check for #1 by comparing the return value to `(my_ulonglong)-1` (or to `(my_ulonglong)~0`, which is equivalent).

See Section 22.2.3.1, "`mysql_affected_rows()`" for additional information on the return value.

Errors

None.

Example

For the usage of `mysql_stmt_affected_rows()`, refer to the Example from Section 22.2.7.10, "`mysql_stmt_execute()`".

22.2.7.2. mysql_stmt_attr_get()

```
int mysql_stmt_attr_get(MYSQL_STMT *stmt, enum enum_stmt_attr_type option, void *arg)
```

Description

Can be used to get the current value for a statement attribute.

The `option` argument is the option that you want to get; the `arg` should point to a variable that should contain the option value. If the option is an integer, then `arg` should point to the value of the integer.

See `mysql_stmt_attr_set()` for a list of options and option types. See Section 22.2.7.3, "`mysql_stmt_attr_set()`".

This function was added in MySQL 4.1.2.

Return Values

`0` if okay. Non-zero if `option` is unknown.

Errors

None.

22.2.7.3. mysql_stmt_attr_set()

```
int mysql_stmt_attr_set(MYSQL_STMT *stmt, enum enum_stmt_attr_type option, const void *arg)
```

Description

Can be used to set affect behavior for a statement. This function may be called multiple times to set several options.

The `option` argument is the option that you want to set; the `arg` argument is the value for the option. If the option is an integer, then `arg` should point to the value of the integer.

Possible options values:

Option	Argument Type	Function
`STMT_ATTR_UPDATE_MAX_LENGTH`	`my_bool *`	If set to 1: Update metadata `MYSQL_FIELD->max_length` in `mysql_stmt_store_result()`.

This function was added in MySQL 4.1.2.

Return Values

`0` if okay. Non-zero if `option` is unknown.

Errors

None.

22.2.7.4. mysql_stmt_bind_param()

```
my_bool mysql_stmt_bind_param(MYSQL_STMT *stmt, MYSQL_BIND *bind)
```

Description

`mysql_stmt_bind_param()` is used to bind data for the parameter markers in the SQL statement that was passed to `mysql_stmt_prepare()`. It uses `MYSQL_BIND` structures to supply the data. `bind` is the address of an array of `MYSQL_BIND` structures. The client library expects the array to contain an element for each '?' parameter marker that is present in the query.

Suppose that you prepare the following statement:

```
INSERT INTO mytbl VALUES(?,?,?)
```

When you bind the parameters, the array of `MYSQL_BIND` structures must contain three elements, and can be declared like this:

```
MYSQL_BIND bind[3];
```

The members of each `MYSQL_BIND` element that should be set are described in Section 22.2.5, “C API Prepared Statement Data types”.

This function was added in MySQL 4.1.2.

Return Values

Zero if the bind was successful. Non-zero if an error occurred.

Errors

- `CR_INVALID_BUFFER_USE`

  Indicates if the bind is to supply the long data in chunks and if the buffer type is non string or binary.

- `CR_UNSUPPORTED_PARAM_TYPE`

  The conversion is not supported. Possibly the `buffer_type` value is illegal or is not one of the supported types.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

For the usage of `mysql_stmt_bind_param()`, refer to the Example from Section 22.2.7.10, “`mysql_stmt_execute()`”.

22.2.7.5. `mysql_stmt_bind_result()`

```
my_bool mysql_stmt_bind_result(MYSQL_STMT *stmt, MYSQL_BIND *bind)
```

Description

`mysql_stmt_bind_result()` is used to associate (bind) columns in the result set to data buffers and length buffers. When `mysql_stmt_fetch()` is called to fetch data, the MySQL client/server protocol places the data for the bound columns into the specified buffers.

All columns must be bound to buffers prior to calling `mysql_stmt_fetch()`. `bind` is the address of an array of `MYSQL_BIND` structures. The client library expects the array to contain an element for each column of the result set. If you do not bind columns to `MYSQL_BIND` structures, `mysql_stmt_fetch()` simply ignores the data fetch. The buffers should be large enough to hold the data values, because the protocol doesn't return data values in chunks.

A column can be bound or rebound at any time, even after a result set has been partially retrieved. The new binding takes effect the next time `mysql_stmt_fetch()` is called. Suppose that an application binds the columns in a result set and calls `mysql_stmt_fetch()`. The client/server protocol returns data in the bound buffers. Then suppose the application binds the columns to a different set of buffers. The protocol does not place data into the newly bound buffers until the next call to `mysql_stmt_fetch()` occurs.

To bind a column, an application calls `mysql_stmt_bind_result()` and passes the type, address, and the address of the length buffer. The members of each `MYSQL_BIND` element that should be set are described in Section 22.2.5, “C API Prepared Statement Data types”.

This function was added in MySQL 4.1.2.

Return Values

Zero if the bind was successful. Non-zero if an error occurred.

Errors

- `CR_UNSUPPORTED_PARAM_TYPE`

  The conversion is not supported. Possibly the `buffer_type` value is illegal or is not one of the supported types.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

For the usage of `mysql_stmt_bind_result()`, refer to the Example from Section 22.2.7.13, "`mysql_stmt_fetch()`".

22.2.7.6. `mysql_stmt_close()`

```
my_bool mysql_stmt_close(MYSQL_STMT *)
```

Description

Closes the prepared statement. `mysql_stmt_close()` also deallocates the statement handle pointed to by `stmt`.

If the current statement has pending or unread results, this function cancels them so that the next query can be executed.

This function was added in MySQL 4.1.0.

Return Values

Zero if the statement was freed successfully. Non-zero if an error occurred.

Errors

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

For the usage of `mysql_stmt_close()`, refer to the Example from Section 22.2.7.10, "`mysql_stmt_execute()`".

22.2.7.7. `mysql_stmt_data_seek()`

```
void mysql_stmt_data_seek(MYSQL_STMT *stmt, my_ulonglong offset)
```

Description

Seeks to an arbitrary row in a statement result set. The `offset` value is a row number and should be in the range from `0` to `mysql_stmt_num_rows(stmt)-1`.

This function requires that the statement result set structure contains the entire result of the last executed query, so `mysql_stmt_data_seek()` may be used only in conjunction with `mysql_stmt_store_result()`.

This function was added in MySQL 4.1.1.

Return Values

None.

Errors

None.

22.2.7.8. `mysql_stmt_errno()`

```
unsigned int mysql_stmt_errno(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_errno()` returns the error code for the most recently invoked statement API function that can succeed or fail. A return value of zero means that no error occurred. Client error message numbers are listed in the MySQL `errmsg.h` header file. Server error message numbers are listed in `mysqld_error.h`. In the MySQL source distribution you can find a complete list of error messages and error numbers in the file `Docs/mysqld_error.txt`. The server error codes also are listed at Chapter 24, *Error Handling in MySQL*.

This function was added in MySQL 4.1.0.

Return Values

An error code value. Zero if no error occurred.

Errors

None.

22.2.7.9. `mysql_stmt_error()`

```
const char *mysql_stmt_error(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_error()` returns a null-terminated string containing the error message for the most recently invoked statement API function that can succeed or fail. An empty string (`""`) is returned if no error occurred. This means the following two tests are equivalent:

```
if (mysql_stmt_errno(stmt))
{
  // an error occurred
}

if (mysql_stmt_error(stmt)[0])
{
  // an error occurred
}
```

The language of the client error messages may be changed by recompiling the MySQL client library. Currently you can choose error messages in several different languages.

This function was added in MySQL 4.1.0.

Return Values

A character string that describes the error. An empty string if no error occurred.

Errors

None.

22.2.7.10. `mysql_stmt_execute()`

```
int mysql_stmt_execute(MYSQL_STMT *stmt)
```

Description

`mysql_stmt_execute()` executes the prepared query associated with the statement handle. The currently bound parameter marker values are sent to server during this call, and the server replaces the markers with this newly supplied data.

If the statement is an `UPDATE`, `DELETE`, or `INSERT`, the total number of changed, deleted, or inserted rows can be found by calling `mysql_stmt_affected_rows()`. If this is a result set query such as `SELECT`, you must call `mysql_stmt_fetch()` to fetch the data prior to calling any other functions that result in query processing. For more information on how to fetch the results, refer to Section 22.2.7.13, "`mysql_stmt_fetch()`".

This function was added in MySQL 4.1.2.

Return Values

Zero if execution was successful. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

The MySQL server has gone away.

- CR_SERVER_LOST

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

Example

The following example demonstrates how to create and populate a table using mysql_stmt_init(), mysql_stmt_prepare(), mysql_stmt_param_count(), mysql_stmt_bind_param(), mysql_stmt_execute(), and mysql_stmt_affected_rows(). The mysql variable is assumed to be a valid connection handle.

```
#define STRING_SIZE 50

#define DROP_SAMPLE_TABLE "DROP TABLE IF EXISTS test_table"
#define CREATE_SAMPLE_TABLE "CREATE TABLE test_table(col1 INT,\
                                                     col2 VARCHAR(40),\
                                                     col3 SMALLINT,\
                                                     col4 TIMESTAMP)"
#define INSERT_SAMPLE "INSERT INTO test_table(col1,col2,col3) VALUES(?,?,?)"

MYSQL_STMT    *stmt;
MYSQL_BIND    bind[3];
my_ulonglong  affected_rows;
int           param_count;
short         small_data;
int           int_data;
char          str_data[STRING_SIZE];
unsigned long str_length;
my_bool       is_null;

if (mysql_query(mysql, DROP_SAMPLE_TABLE))
{
  fprintf(stderr, " DROP TABLE failed\n");
  fprintf(stderr, " %s\n", mysql_error(mysql));
  exit(0);
}

if (mysql_query(mysql, CREATE_SAMPLE_TABLE))
{
  fprintf(stderr, " CREATE TABLE failed\n");
  fprintf(stderr, " %s\n", mysql_error(mysql));
  exit(0);
}

/* Prepare an INSERT query with 3 parameters */
/* (the TIMESTAMP column is not named; the server */
/*  sets it to the current date and time) */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(stmt, INSERT_SAMPLE, strlen(INSERT_SAMPLE)))
{
```

```
  fprintf(stderr, " mysql_stmt_prepare(), INSERT failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
fprintf(stdout, " prepare, INSERT successful\n");

/* Get the parameter count from the statement */
param_count= mysql_stmt_param_count(stmt);
fprintf(stdout, " total parameters in INSERT: %d\n", param_count);

if (param_count != 3) /* validate parameter count */
{
  fprintf(stderr, " invalid parameter count returned by MySQL\n");
  exit(0);
}

/* Bind the data for all 3 parameters */

memset(bind, 0, sizeof(bind));

/* INTEGER PARAM */
/* This is a number type, so there is no need to specify buffer_length */
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= 0;
bind[0].length= 0;

/* STRING PARAM */
bind[1].buffer_type= MYSQL_TYPE_STRING;
bind[1].buffer= (char *)str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= 0;
bind[1].length= &str_length;

/* SMALLINT PARAM */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null;
bind[2].length= 0;

/* Bind the buffers */
if (mysql_stmt_bind_param(stmt, bind))
{
  fprintf(stderr, " mysql_stmt_bind_param() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Specify the data values for the first row */
int_data= 10;             /* integer */
strncpy(str_data, "MySQL", STRING_SIZE); /* string  */
str_length= strlen(str_data);

/* INSERT SMALLINT data as NULL */
is_null= 1;

/* Execute the INSERT statement - 1*/
if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, " mysql_stmt_execute(), 1 failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
```

```
/* Get the total number of affected rows */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 1): %lu\n",
                (unsigned long) affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
  fprintf(stderr, " invalid affected rows by MySQL\n");
  exit(0);
}

/* Specify data values for second row, then re-execute the statement */
int_data= 1000;
strncpy(str_data, "The most popular Open Source database", STRING_SIZE);
str_length= strlen(str_data);
small_data= 1000;         /* smallint */
is_null= 0;               /* reset */

/* Execute the INSERT statement - 2*/
if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, " mysql_stmt_execute, 2 failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Get the total rows affected */
affected_rows= mysql_stmt_affected_rows(stmt);
fprintf(stdout, " total affected rows(insert 2): %lu\n",
                (unsigned long) affected_rows);

if (affected_rows != 1) /* validate affected rows */
{
  fprintf(stderr, " invalid affected rows by MySQL\n");
  exit(0);
}

/* Close the statement */
if (mysql_stmt_close(stmt))
{
  fprintf(stderr, " failed while closing the statement\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
```

Note: For complete examples on the use of prepared statement functions, refer to the file `tests/mysql_client_test.c`. This file can be obtained from a MySQL source distribution or from the BitKeeper source repository.

22.2.7.11. `mysql_stmt_free_result()`

```
my_bool mysql_stmt_free_result(MYSQL_STMT *stmt)
```

Description

Releases memory associated with the result set produced by execution of the prepared statement.

This function was added in MySQL 4.1.1.

Return Values

Zero if the result set was freed successfully. Non-zero if an error occurred.

Errors

22.2.7.12. mysql_stmt_insert_id()

```
my_ulonglong mysql_stmt_insert_id(MYSQL_STMT *stmt)
```

Description

Returns the value generated for an AUTO_INCREMENT column by the prepared INSERT or UPDATE statement. Use this function after you have executed prepared INSERT statement into a table which contains an AUTO_INCREMENT field.

See Section 22.2.3.33, "mysql_insert_id()" for more information.

This function was added in MySQL 4.1.2.

Return Values

Value for AUTO_INCREMENT column which was automatically generated or explicitly set during execution of prepared statement, or value generated by LAST_INSERT_ID(*expr*) function. Return value is undefined if statement does not set AUTO_INCREMENT value.

Errors

None.

22.2.7.13. mysql_stmt_fetch()

```
int mysql_stmt_fetch(MYSQL_STMT *stmt)
```

Description

mysql_stmt_fetch() returns the next row in the result set. It can be called only while the result set exists, that is, after a call to mysql_stmt_execute() that creates a result set or after mysql_stmt_store_result(), which is called after mysql_stmt_execute() to buffer the entire result set.

mysql_stmt_fetch() returns row data using the buffers bound by mysql_stmt_bind_result(). It returns the data in those buffers for all the columns in the current row set and the lengths are returned to the length pointer.

All columns must be bound by the application before calling mysql_stmt_fetch().

If a fetched data value is a NULL value, the *is_null value of the corresponding MYSQL_BIND structure contains TRUE (1). Otherwise, the data and its length are returned in the *buffer and *length elements based on the buffer type specified by the application. Each numeric and temporal type has a fixed length, as listed in the following table. The length of the string types depends on the length of the actual data value, as indicated by data_length.

Type	Length
MYSQL_TYPE_TINY	1
MYSQL_TYPE_SHORT	2
MYSQL_TYPE_LONG	4
MYSQL_TYPE_LONGLONG	8

MYSQL_TYPE_FLOAT	4
MYSQL_TYPE_DOUBLE	8
MYSQL_TYPE_TIME	sizeof(MYSQL_TIME)
MYSQL_TYPE_DATE	sizeof(MYSQL_TIME)
MYSQL_TYPE_DATETIME	sizeof(MYSQL_TIME)
MYSQL_TYPE_STRING	data length
MYSQL_TYPE_BLOB	data_length

This function was added in MySQL 4.1.2.

Return Values

Return Value	Description
0	Successful, the data has been fetched to application data buffers.
1	Error occurred. Error code and message can be obtained by calling mysql_stmt_errno() and mysql_stmt_error().
MYSQL_NO_DATA	No more rows/data exists
MYSQL_DATA_TRUNCATED	Data truncation occurred

MYSQL_DATA_TRUNCATED is not returned unless truncation reporting is enabled with mysql_options(). To determine which parameters were truncated when this value is returned, check the error members of the MYSQL_BIND parameter structures.

Errors

- CR_COMMANDS_OUT_OF_SYNC

  Commands were executed in an improper order.

- CR_OUT_OF_MEMORY

  Out of memory.

- CR_SERVER_GONE_ERROR

  The MySQL server has gone away.

- CR_SERVER_LOST

  The connection to the server was lost during the query.

- CR_UNKNOWN_ERROR

  An unknown error occurred.

- CR_UNSUPPORTED_PARAM_TYPE

  The buffer type is MYSQL_TYPE_DATE, MYSQL_TYPE_TIME, MYSQL_TYPE_DATETIME, or MYSQL_TYPE_TIMESTAMP, but the data type is not DATE, TIME, DATETIME, or TIMESTAMP.

- All other unsupported conversion errors are returned from mysql_stmt_bind_result().

Example

The following example demonstrates how to fetch data from a table using `mysql_stmt_result_metadata()`, `mysql_stmt_bind_result()`, and `mysql_stmt_fetch()`. (This example expects to retrieve the two rows inserted by the example shown in Section 22.2.7.10, "`mysql_stmt_execute()`".) The `mysql` variable is assumed to be a valid connection handle.

```
#define STRING_SIZE 50

#define SELECT_SAMPLE "SELECT col1, col2, col3, col4 FROM test_table"

MYSQL_STMT    *stmt;
MYSQL_BIND    bind[4];
MYSQL_RES     *prepare_meta_result;
MYSQL_TIME    ts;
unsigned long length[4];
int           param_count, column_count, row_count;
short         small_data;
int           int_data;
char          str_data[STRING_SIZE];
my_bool       is_null[4];

/* Prepare a SELECT query to fetch data from test_table */
stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(stmt, SELECT_SAMPLE, strlen(SELECT_SAMPLE)))
{
  fprintf(stderr, " mysql_stmt_prepare(), SELECT failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
fprintf(stdout, " prepare, SELECT successful\n");

/* Get the parameter count from the statement */
param_count= mysql_stmt_param_count(stmt);
fprintf(stdout, " total parameters in SELECT: %d\n", param_count);

if (param_count != 0) /* validate parameter count */
{
  fprintf(stderr, " invalid parameter count returned by MySQL\n");
  exit(0);
}

/* Fetch result set meta information */
prepare_meta_result = mysql_stmt_result_metadata(stmt);
if (!prepare_meta_result)
{
  fprintf(stderr,
         " mysql_stmt_result_metadata(), returned no meta information\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Get total columns in the query */
column_count= mysql_num_fields(prepare_meta_result);
fprintf(stdout, " total columns in SELECT statement: %d\n", column_count);

if (column_count != 4) /* validate column count */
```

```
{
  fprintf(stderr, " invalid column count returned by MySQL\n");
  exit(0);
}

/* Execute the SELECT query */
if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, " mysql_stmt_execute(), failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Bind the result buffers for all 4 columns before fetching them */

memset(bind, 0, sizeof(bind));

/* INTEGER COLUMN */
bind[0].buffer_type= MYSQL_TYPE_LONG;
bind[0].buffer= (char *)&int_data;
bind[0].is_null= &is_null[0];
bind[0].length= &length[0];

/* STRING COLUMN */
bind[1].buffer_type= MYSQL_TYPE_STRING;
bind[1].buffer= (char *)str_data;
bind[1].buffer_length= STRING_SIZE;
bind[1].is_null= &is_null[1];
bind[1].length= &length[1];

/* SMALLINT COLUMN */
bind[2].buffer_type= MYSQL_TYPE_SHORT;
bind[2].buffer= (char *)&small_data;
bind[2].is_null= &is_null[2];
bind[2].length= &length[2];

/* TIMESTAMP COLUMN */
bind[3].buffer_type= MYSQL_TYPE_TIMESTAMP;
bind[3].buffer= (char *)&ts;
bind[3].is_null= &is_null[3];
bind[3].length= &length[3];

/* Bind the result buffers */
if (mysql_stmt_bind_result(stmt, bind))
{
  fprintf(stderr, " mysql_stmt_bind_result() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Now buffer all results to client */
if (mysql_stmt_store_result(stmt))
{
  fprintf(stderr, " mysql_stmt_store_result() failed\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}

/* Fetch all rows */
row_count= 0;
fprintf(stdout, "Fetching results ...\n");
while (!mysql_stmt_fetch(stmt))
{
  row_count++;
```

```
  fprintf(stdout, "  row %d\n", row_count);

  /* column 1 */
  fprintf(stdout, "   column1 (integer)  : ");
  if (is_null[0])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %d(%ld)\n", int_data, length[0]);

  /* column 2 */
  fprintf(stdout, "   column2 (string)   : ");
  if (is_null[1])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %s(%ld)\n", str_data, length[1]);

  /* column 3 */
  fprintf(stdout, "   column3 (smallint) : ");
  if (is_null[2])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %d(%ld)\n", small_data, length[2]);

  /* column 4 */
  fprintf(stdout, "   column4 (timestamp): ");
  if (is_null[3])
    fprintf(stdout, " NULL\n");
  else
    fprintf(stdout, " %04d-%02d-%02d %02d:%02d:%02d (%ld)\n",
                     ts.year, ts.month, ts.day,
                     ts.hour, ts.minute, ts.second,
                     length[3]);
  fprintf(stdout, "\n");
}

/* Validate rows fetched */
fprintf(stdout, " total rows fetched: %d\n", row_count);
if (row_count != 2)
{
  fprintf(stderr, " MySQL failed to return all rows\n");
  exit(0);
}

/* Free the prepared result metadata */
mysql_free_result(prepare_meta_result);


/* Close the statement */
if (mysql_stmt_close(stmt))
{
  fprintf(stderr, " failed while closing the statement\n");
  fprintf(stderr, " %s\n", mysql_stmt_error(stmt));
  exit(0);
}
```

22.2.7.14. mysql_stmt_fetch_column()

```
int mysql_stmt_fetch_column(MYSQL_STMT *stmt, MYSQL_BIND *bind, un-
signed int column, unsigned long offset)
```

Description

Fetch one column from the current result set row. `bind` provides the buffer where data should be placed. It should be set up the same way as for `mysql_stmt_bind_result()`. `column` indicates which column to fetch. The first column is numbered 0. `offset` is the offset within the data value at which to begin retrieving data. This can be used for fetching the data value in pieces. The beginning of the value is offset 0.

This function was added in MySQL 4.1.2.

Return Values

Zero if the value was fetched successfully. Non-zero if an error occurred.

Errors

- `CR_INVALID_PARAMETER_NO`

  Invalid column number.

- `CR_NO_DATA`

  The end of the result set has already been reached.

22.2.7.15. `mysql_stmt_field_count()`

```
unsigned int mysql_stmt_field_count(MYSQL_STMT *stmt)
```

Description

Returns the number of columns for the most recent statement for the statement handler. This value is zero for statements such as `INSERT` or `DELETE` that do not produce result sets.

`mysql_stmt_field_count()` can be called after you have prepared a statement by invoking `mysql_stmt_prepare()`.

This function was added in MySQL 4.1.3.

Return Values

An unsigned integer representing the number of columns in a result set.

Errors

None.

22.2.7.16. `mysql_stmt_init()`

```
MYSQL_STMT *mysql_stmt_init(MYSQL *mysql)
```

Description

Create a `MYSQL_STMT` handle. The handle should be freed with `mysql_stmt_close(MYSQL_STMT *)`.

This function was added in MySQL 4.1.2.

Return values

A pointer to a `MYSQL_STMT` structure in case of success. `NULL` if out of memory.

Errors

- `CR_OUT_OF_MEMORY`

  Out of memory.

22.2.7.17. mysql_stmt_num_rows()

```
my_ulonglong mysql_stmt_num_rows(MYSQL_STMT *stmt)
```

Description

Returns the number of rows in the result set.

The use of `mysql_stmt_num_rows()` depends on whether or not you used `mysql_stmt_store_result()` to buffer the entire result set in the statement handle.

If you use `mysql_stmt_store_result()`, `mysql_stmt_num_rows()` may be called immediately.

This function was added in MySQL 4.1.1.

Return Values

The number of rows in the result set.

Errors

None.

22.2.7.18. mysql_stmt_param_count()

```
unsigned long mysql_stmt_param_count(MYSQL_STMT *stmt)
```

Description

Returns the number of parameter markers present in the prepared statement.

This function was added in MySQL 4.1.2.

Return Values

An unsigned long integer representing the number of parameters in a statement.

Errors

None.

Example

For the usage of `mysql_stmt_param_count()`, refer to the Example from Section 22.2.7.10, “`mysql_stmt_execute()`”.

22.2.7.19. mysql_stmt_param_metadata()

```
MYSQL_RES *mysql_stmt_param_metadata(MYSQL_STMT *stmt)
```

To be added.

This function was added in MySQL 4.1.2.

Description

Return Values

Errors

22.2.7.20. mysql_stmt_prepare()

```
int mysql_stmt_prepare(MYSQL_STMT *stmt, const char *query, unsigned long length)
```

Description

Given the statement handle returned by `mysql_stmt_init()`, prepares the SQL statement pointed to by the string `query` and returns a status value. The string length should be given by the `length` argument. The string must consist of a single SQL statement. You should not add a terminating semicolon ('`;`') or `\g` to the statement.

The application can include one or more parameter markers in the SQL statement by embedding question mark ('`?`') characters into the SQL string at the appropriate positions.

The markers are legal only in certain places in SQL statements. For example, they are allowed in the `VALUES()` list of an `INSERT` statement (to specify column values for a row), or in a comparison with a column in a `WHERE` clause to specify a comparison value. However, they are not allowed for identifiers (such as table or column names), or to specify both operands of a binary operator such as the `=` equal sign. The latter restriction is necessary because it would be impossible to determine the parameter type. In general, parameters are legal only in Data Manipulation Language (DML) statements, and not in Data Definition Language (DDL) statements.

The parameter markers must be bound to application variables using `mysql_stmt_bind_param()` before executing the statement.

This function was added in MySQL 4.1.2.

Return Values

Zero if the statement was prepared successfully. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- CR_SERVER_LOST

  The connection to the server was lost during the query

- CR_UNKNOWN_ERROR

  An unknown error occurred.

If the prepare operation was unsuccessful (that is, mysql_stmt_prepare() returns non-zero), the error message can be obtained by calling mysql_stmt_error().

Example

For the usage of mysql_stmt_prepare(), refer to the Example from Section 22.2.7.10, "mysql_stmt_execute()".

22.2.7.21. mysql_stmt_reset()

```
my_bool mysql_stmt_reset(MYSQL_STMT *stmt)
```

Description

Reset prepared statement on client and server to state after prepare. This is mainly used to reset data sent with mysql_stmt_send_long_data().

To re-prepare the statement with another query, use mysql_stmt_prepare().

This function was added in MySQL 4.1.1.

Return Values

Zero if the statement was reset successfully. Non-zero if an error occurred.

Errors

- CR_COMMANDS_OUT_OF_SYNC

  Commands were executed in an improper order.

- CR_SERVER_GONE_ERROR

  The MySQL server has gone away.

- CR_SERVER_LOST

  The connection to the server was lost during the query

- CR_UNKNOWN_ERROR

  An unknown error occurred.

22.2.7.22. mysql_stmt_result_metadata()

```
MYSQL_RES *mysql_stmt_result_metadata(MYSQL_STMT *stmt)
```

Description

If a statement passed to `mysql_stmt_prepare()` is one that produces a result set, `mysql_stmt_result_metadata()` returns the result set metadata in the form of a pointer to a `MYSQL_RES` structure that can be used to process the meta information such as total number of fields and individual field information. This result set pointer can be passed as an argument to any of the field-based API functions that process result set metadata, such as:

- `mysql_num_fields()`
- `mysql_fetch_field()`
- `mysql_fetch_field_direct()`
- `mysql_fetch_fields()`
- `mysql_field_count()`
- `mysql_field_seek()`
- `mysql_field_tell()`
- `mysql_free_result()`

The result set structure should be freed when you are done with it, which you can do by passing it to `mysql_free_result()`. This is similar to the way you free a result set obtained from a call to `mysql_store_result()`.

The result set returned by `mysql_stmt_result_metadata()` contains only metadata. It does not contain any row results. The rows are obtained by using the statement handle with `mysql_stmt_fetch()`.

This function was added in MySQL 4.1.2.

Return Values

A `MYSQL_RES` result structure. `NULL` if no meta information exists for the prepared query.

Errors

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

For the usage of `mysql_stmt_result_metadata()`, refer to the Example from Section 22.2.7.13, “`mysql_stmt_fetch()`”.

22.2.7.23. `mysql_stmt_row_seek()`

```
MYSQL_ROW_OFFSET         mysql_stmt_row_seek(MYSQL_STMT         *stmt,
MYSQL_ROW_OFFSET offset)
```

Description

Sets the row cursor to an arbitrary row in a statement result set. The `offset` value is a row offset that should be a value returned from `mysql_stmt_row_tell()` or from `mysql_stmt_row_seek()`. This value is not a row number; if you want to seek to a row within a result set by number, use `mysql_stmt_data_seek()` instead.

This function requires that the result set structure contains the entire result of the query, so `mysql_stmt_row_seek()` may be used only in conjunction with `mysql_stmt_store_result()`.

This function was added in MySQL 4.1.1.

Return Values

The previous value of the row cursor. This value may be passed to a subsequent call to `mysql_stmt_row_seek()`.

Errors

None.

22.2.7.24. mysql_stmt_row_tell()

```
MYSQL_ROW_OFFSET mysql_stmt_row_tell(MYSQL_STMT *stmt)
```

Description

Returns the current position of the row cursor for the last `mysql_stmt_fetch()`. This value can be used as an argument to `mysql_stmt_row_seek()`.

You should use `mysql_stmt_row_tell()` only after `mysql_stmt_store_result()`.

This function was added in MySQL 4.1.1.

Return Values

The current offset of the row cursor.

Errors

None.

22.2.7.25. mysql_stmt_send_long_data()

```
my_bool mysql_stmt_send_long_data(MYSQL_STMT *stmt, unsigned int para-
meter_number, const char *data, unsigned long length)
```

Description

Allows an application to send parameter data to the server in pieces (or ``chunks''). This function can be called multiple times to send the parts of a character or binary data value for a column, which must be one of the `TEXT` or `BLOB` data types.

`parameter_number` indicates which parameter to associate the data with. Parameters are numbered beginning with 0. `data` is a pointer to a buffer containing data to be sent, and `length` indicates the number of bytes in the buffer.

Note: The next `mysql_stmt_execute()` call ignores the bind buffer for all parameters that have

been used with `mysql_stmt_send_long_data()` since last `mysql_stmt_execute()` or `mysql_stmt_reset()`.

If you want to reset/forget the sent data, you can do it with `mysql_stmt_reset()`. See Section 22.2.7.21, "`mysql_stmt_reset()`".

This function was added in MySQL 4.1.2.

Return Values

Zero if the data is sent successfully to server. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

Example

The following example demonstrates how to send the data for a `TEXT` column in chunks. It inserts the data value `'MySQL - The most popular Open Source database'` into the `text_column` column. The `mysql` variable is assumed to be a valid connection handle.

```
#define INSERT_QUERY "INSERT INTO test_long_data(text_column) VALUES(?)"

MYSQL_BIND bind[1];
long       length;

smtt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(stmt, INSERT_QUERY, strlen(INSERT_QUERY)))
{
  fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}
 memset(bind, 0, sizeof(bind));
 bind[0].buffer_type= MYSQL_TYPE_STRING;
 bind[0].length= &length;
 bind[0].is_null= 0;

/* Bind the buffers */
if (mysql_stmt_bind_param(stmt, bind))
```

```
{
  fprintf(stderr, "\n param bind failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

 /* Supply data in chunks to server */
 if (!mysql_stmt_send_long_data(stmt,0,"MySQL",5))
{
  fprintf(stderr, "\n send_long_data failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

 /* Supply the next piece of data */
 if (mysql_stmt_send_long_data(stmt,0," - The most popular Open Source database",4
{
  fprintf(stderr, "\n send_long_data failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

 /* Now, execute the query */
 if (mysql_stmt_execute(stmt))
{
  fprintf(stderr, "\n mysql_stmt_execute failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}
```

22.2.7.26. mysql_stmt_sqlstate()

```
const char *mysql_stmt_sqlstate(MYSQL_STMT *stmt)
```

Description

For the statement specified by `stmt`, `mysql_stmt_sqlstate()` returns a null-terminated string containing the SQLSTATE error code for the most recently invoked prepared statement API function that can succeed or fail. The error code consists of five characters. `"00000"` means ``no error.'' The values are specified by ANSI SQL and ODBC. For a list of possible values, see Chapter 24, *Error Handling in MySQL*.

Note that not all MySQL errors are yet mapped to SQLSTATE's. The value `"HY000"` (general error) is used for unmapped errors.

This function was added to MySQL 4.1.1.

Return Values

A null-terminated character string containing the SQLSTATE error code.

22.2.7.27. mysql_stmt_store_result()

```
int mysql_stmt_store_result(MYSQL_STMT *stmt)
```

Description

You must call `mysql_stmt_store_result()` for every statement that successfully produces a result set (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`), and only if you want to buffer the complete result set by the client, so that the subsequent `mysql_stmt_fetch()` call returns buffered data.

It is unnecessary to call `mysql_stmt_store_result()` for other statements, but if you do, it does not harm or cause any notable performance problem. You can detect whether the statement produced a result set by checking if `mysql_stmt_result_metadata()` returns `NULL`. For more information, refer to Section 22.2.7.22, "`mysql_stmt_result_metadata()`".

Note: MySQL doesn't by default calculate `MYSQL_FIELD->max_length` for all columns in `mysql_stmt_store_result()` because calculating this would slow down `mysql_stmt_store_result()` considerably and most applications doesn't need `max_length`. If you want `max_length` to be updated, you can call `mysql_stmt_attr_set(MYSQL_STMT, STMT_ATTR_UPDATE_MAX_LENGTH, &flag)` to enable this. See Section 22.2.7.3, "`mysql_stmt_attr_set()`".

This function was added in MySQL 4.1.0.

Return Values

Zero if the results are buffered successfully. Non-zero if an error occurred.

Errors

- `CR_COMMANDS_OUT_OF_SYNC`

  Commands were executed in an improper order.

- `CR_OUT_OF_MEMORY`

  Out of memory.

- `CR_SERVER_GONE_ERROR`

  The MySQL server has gone away.

- `CR_SERVER_LOST`

  The connection to the server was lost during the query.

- `CR_UNKNOWN_ERROR`

  An unknown error occurred.

22.2.8. C API Prepared statement problems

Here follows a list of the currently known problems with prepared statements:

- `TIME`, `TIMESTAMP`, and `DATETIME` don't support sub seconds (for example from `DATE_FORMAT()`.

- When converting an integer to string, `ZEROFILL` is honored with prepared statements in some cases where the MySQL server doesn't print the leading zeros. (For example, with `MIN(number-with-zerofill)`).

- When converting a floating point number to a string in the client, the value may be slightly different in the last digits.

22.2.9. C API Handling of Multiple Query Execution

From version 4.1, MySQL supports the execution of multiple statements specified in a single query string. To use this capability with a given connection, you must specify the `CLIENT_MULTI_STATEMENTS` option in the flags parameter of `mysql_real_connect()` when opening the connection. You can also set this for an existing connection by calling `mysql_set_server_option(MYSQL_OPTION_MULTI_STATEMENTS_ON)`

By default, `mysql_query()` and `mysql_real_query()` return only the first query status and the subsequent queries status can be processed using `mysql_more_results()` and `mysql_next_result()`.

```
/* Connect to server with option CLIENT_MULTI_STATEMENTS */
mysql_real_connect(..., CLIENT_MULTI_STATEMENTS);

/* Now execute multiple queries */
mysql_query(mysql,"DROP TABLE IF EXISTS test_table;\
                   CREATE TABLE test_table(id INT);\
                   INSERT INTO test_table VALUES(10);\
                   UPDATE test_table SET id=20 WHERE id=10;\
                   SELECT * FROM test_table;\
                   DROP TABLE test_table");
do
{
  /* Process all results */
  ...
  printf("total affected rows: %lld", mysql_affected_rows(mysql));
  ...
  if (!(result= mysql_store_result(mysql)))
  {
     printf(stderr, "Got fatal error processing query\n");
     exit(1);
  }
  process_result_set(result); /* client function */
  mysql_free_result(result);
} while (!mysql_next_result(mysql));
```

The multiple-statement capability can be used with `mysql_query()` or `mysql_real_query()`. It cannot be used with the prepared statement interface. Prepared statement handles are defined to work only with strings that contain a single statement.

22.2.10. C API Handling of Date and Time Values

The new binary protocol available in MySQL 4.1 and above allows you to send and receive date and time values (`DATE`, `TIME`, `DATETIME`, and `TIMESTAMP`), using the `MYSQL_TIME` structure. The members of this structure are described in Section 22.2.5, “C API Prepared Statement Data types”.

To send temporal data values, you create a prepared statement with `mysql_stmt_prepare()`. Then, before calling `mysql_stmt_execute()` to execute the statement, use the following procedure to set up each temporal parameter:

1. In the `MYSQL_BIND` structure associated with the data value, set the `buffer_type` member to the type that indicates what kind of temporal value you're sending. For `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP` values, set `buffer_type` to `MYSQL_TYPE_DATE`, `MYSQL_TYPE_TIME`, `MYSQL_TYPE_DATETIME`, or `MYSQL_TYPE_TIMESTAMP`, respectively.

2. Set the `buffer` member of the `MYSQL_BIND` structure to the address of the `MYSQL_TIME` structure in which you pass the temporal value.

3. Fill in the members of the `MYSQL_TIME` structure that are appropriate for the type of temporal value you're passing.

Use `mysql_stmt_bind_param()` to bind the parameter data to the statement. Then you can call `mysql_stmt_execute()`.

To retrieve temporal values, the procedure is similar, except that you set the `buffer_type` member to the type of value you expect to receive, and the `buffer` member to the address of a `MYSQL_TIME` structure into which the returned value should be placed. Use `mysql_bind_results()` to bind the buffers to the statement after calling `mysql_stmt_execute()` and before fetching the results.

Here is a simple example that inserts `DATE`, `TIME`, and `TIMESTAMP` data. The `mysql` variable is assumed to be a valid connection handle.

```
MYSQL_TIME  ts;
MYSQL_BIND  bind[3];
MYSQL_STMT  *stmt;

strmov(query, "INSERT INTO test_table(date_field, time_field,
                                      timestamp_field) VALUES(?,?,?");

stmt = mysql_stmt_init(mysql);
if (!stmt)
{
  fprintf(stderr, " mysql_stmt_init(), out of memory\n");
  exit(0);
}
if (mysql_stmt_prepare(mysql, query, strlen(query)))
{
  fprintf(stderr, "\n mysql_stmt_prepare(), INSERT failed");
  fprintf(stderr, "\n %s", mysql_stmt_error(stmt));
  exit(0);
}

/* set up input buffers for all 3 parameters */
bind[0].buffer_type= MYSQL_TYPE_DATE;
bind[0].buffer= (char *)&ts;
bind[0].is_null= 0;
bind[0].length= 0;
...
bind[1]= bind[2]= bind[0];
...

mysql_stmt_bind_param(stmt, bind);

/* supply the data to be sent in the ts structure */
ts.year= 2002;
ts.month= 02;
ts.day= 03;

ts.hour= 10;
ts.minute= 45;
ts.second= 20;

mysql_stmt_execute(stmt);
..
```

22.2.11. C API Threaded Function Descriptions

You need to use the following functions when you want to create a threaded client. See Section 22.2.15, “How to Make a Threaded Client”.

22.2.11.1. my_init()

```
void my_init(void)
```

Description

This function needs to be called once in the program before calling any MySQL function. This initializes some global variables that MySQL needs. If you are using a thread-safe client library, this also calls `mysql_thread_init()` for this thread.

This is automatically called by `mysql_init()`, `mysql_library_init()`, `mysql_server_init()` and `mysql_connect()`.

Return Values

None.

22.2.11.2. mysql_thread_init()

```
my_bool mysql_thread_init(void)
```

Description

This function needs to be called for each created thread to initialize thread-specific variables.

This is automatically called by `my_init()` and `mysql_connect()`.

Return Values

Zero if successful. Non-zero if an error occurred.

22.2.11.3. mysql_thread_end()

```
void mysql_thread_end(void)
```

Description

This function needs to be called before calling `pthread_exit()` to free memory allocated by `mysql_thread_init()`.

Note that this function **is not invoked automatically** by the client library. It must be called explicitly to avoid a memory leak.

Return Values

None.

22.2.11.4. mysql_thread_safe()

```
unsigned int mysql_thread_safe(void)
```

Description

This function indicates whether the client is compiled as thread-safe.

Return Values

1 is the client is thread-safe, 0 otherwise.

22.2.12. C API Embedded Server Function Descriptions

If you want to allow your application to be linked against the embedded MySQL server library, you must use the `mysql_server_init()` and `mysql_server_end()` functions. See Section 22.2.16, “libmysqld, the Embedded MySQL Server Library”.

However, to provide improved memory management, even programs that are linked with `-lmysqlclient` rather than `-lmysqld` should include calls to begin and end use of the library. As of MySQL 4.1.10 and 5.0.3, the `mysql_library_init()` and `mysql_library_end()` functions can be used to do this. These actually are `#define` symbols that make them equivalent to `mysql_server_init()` and `mysql_server_end()`, but the names more clearly indicate that they should be called when beginning and ending use of a MySQL C API library no matter whether the application uses `libmysqlclient` or `libmysqld`. For more information, see Section 22.2.2, “C API Function Overview”.

22.2.12.1. `mysql_server_init()`

```
int mysql_server_init(int argc, char **argv, char **groups)
```

Description

This function **must** be called once in the program using the embedded server before calling any other MySQL function. It starts the server and initializes any subsystems (`mysys`, `InnoDB`, etc.) that the server uses. If this function is not called, the next call to `mysql_init()` executes `mysql_server_init()`. If you are using the DBUG package that comes with MySQL, you should call this after you have called `my_init()`.

The `argc` and `argv` arguments are analogous to the arguments to `main()`. The first element of `argv` is ignored (it typically contains the program name). For convenience, `argc` may be `0` (zero) if there are no command-line arguments for the server. `mysql_server_init()` makes a copy of the arguments so it's safe to destroy `argv` or `groups` after the call.

If you want to connect to an external server without starting the embedded server, you have to specify a negative value for `argc`.

The `NULL`-terminated list of strings in `groups` selects which groups in the option files are active. See Section 4.3.2, “Using Option Files”. For convenience, `groups` may be `NULL`, in which case the `[server]` and `[embedded]` groups are active.

Example

```
#include <mysql.h>
#include <stdlib.h>

static char *server_args[] = {
  "this_program",       /* this string is not used */
  "--datadir=.",
  "--key_buffer_size=32M"
};
static char *server_groups[] = {
  "embedded",
  "server",
  "this_program_SERVER",
  (char *)NULL
};
```

```
int main(void) {
  if (mysql_server_init(sizeof(server_args) / sizeof(char *),
                        server_args, server_groups))
    exit(1);

  /* Use any MySQL API functions here */

  mysql_server_end();

  return EXIT_SUCCESS;
}
```

Return Values

0 if okay, 1 if an error occurred.

22.2.12.2. `mysql_server_end()`

```
void mysql_server_end(void)
```

Description

This function **must** be called once in the program after all other MySQL functions. It shuts down the embedded server.

Return Values

None.

22.2.13. Common questions and problems when using the C API

22.2.13.1. Why `mysql_store_result()` Sometimes Returns `NULL` After `mysql_query()` Returns Success

It is possible for `mysql_store_result()` to return `NULL` following a successful call to `mysql_query()`. When this happens, it means one of the following conditions occurred:

- There was a `malloc()` failure (for example, if the result set was too large).
- The data couldn't be read (an error occurred on the connection).
- The query returned no data (for example, it was an `INSERT`, `UPDATE`, or `DELETE`).

You can always check whether the statement should have produced a non-empty result by calling `mysql_field_count()`. If `mysql_field_count()` returns zero, the result is empty and the last query was a statement that does not return values (for example, an `INSERT` or a `DELETE`). If `mysql_field_count()` returns a non-zero value, the statement should have produced a non-empty result. See the description of the `mysql_field_count()` function for an example.

You can test for an error by calling `mysql_error()` or `mysql_errno()`.

22.2.13.2. What Results You Can Get from a Query

In addition to the result set returned by a query, you can also get the following information:

- `mysql_affected_rows()` returns the number of rows affected by the last query when doing an `INSERT`, `UPDATE`, or `DELETE`.

  In MySQL 3.23, there is an exception when `DELETE` is used without a `WHERE` clause. In this case, the table is re-created as an empty table and `mysql_affected_rows()` returns zero for the number of records affected. In MySQL 4.0, `DELETE` always returns the correct number of rows deleted. For a fast recreate, use `TRUNCATE TABLE`.

- `mysql_num_rows()` returns the number of rows in a result set. With `mysql_store_result()`, `mysql_num_rows()` may be called as soon as `mysql_store_result()` returns. With `mysql_use_result()`, `mysql_num_rows()` may be called only after you have fetched all the rows with `mysql_fetch_row()`.

- `mysql_insert_id()` returns the ID generated by the last query that inserted a row into a table with an `AUTO_INCREMENT` index. See Section 22.2.3.33, "`mysql_insert_id()`".

- Some queries (`LOAD DATA INFILE ...`, `INSERT INTO ... SELECT ...`, `UPDATE`) return additional information. The result is returned by `mysql_info()`. See the description for `mysql_info()` for the format of the string that it returns. `mysql_info()` returns a `NULL` pointer if there is no additional information.

22.2.13.3. How to Get the Unique ID for the Last Inserted Row

If you insert a record into a table that contains an `AUTO_INCREMENT` column, you can obtain the value stored into that column by calling the `mysql_insert_id()` function.

You can check from your C applications whether a value was stored into an `AUTO_INCREMENT` column by executing the following code (which assumes that you've checked that the statement succeeded). It determines whether the query was an `INSERT` with an `AUTO_INCREMENT` index:

```
if ((result = mysql_store_result(&mysql)) == 0 &&
    mysql_field_count(&mysql) == 0 &&
    mysql_insert_id(&mysql) != 0)
{
    used_id = mysql_insert_id(&mysql);
}
```

For more information, see Section 22.2.3.33, "`mysql_insert_id()`".

When a new `AUTO_INCREMENT` value has been generated, you can also obtain it by executing a `SELECT LAST_INSERT_ID()` statement with `mysql_query()` and retrieving the value from the result set returned by the statement.

For `LAST_INSERT_ID()`, the most recently generated ID is maintained in the server on a per-connection basis. It is not changed by another client. It is not even changed if you update another `AUTO_INCREMENT` column with a non-magic value (that is, a value that is not `NULL` and not `0`).

If you want to use the ID that was generated for one table and insert it into a second table, you can use SQL statements like this:

```
INSERT INTO foo (auto,text)
    VALUES(NULL,'text');              # generate ID by inserting NULL
INSERT INTO foo2 (id,text)
    VALUES(LAST_INSERT_ID(),'text');  # use ID in second table
```

Note that `mysql_insert_id()` returns the value stored into an `AUTO_INCREMENT` column, whether that value is automatically generated by storing `NULL` or `0` or was specified as an explicit value. `LAST_INSERT_ID()` returns only automatically generated `AUTO_INCREMENT` values. If you store an explicit value other than `NULL` or `0`, it does not affect the value returned by `LAST_INSERT_ID()`.

22.2.13.4. Problems Linking with the C API

When linking with the C API, the following errors may occur on some systems:

```
gcc -g -o client test.o -L/usr/local/lib/mysql -lmysqlclient -lsocket -lnsl

Undefined        first referenced
 symbol          in file
floor            /usr/local/lib/mysql/libmysqlclient.a(password.o)
ld: fatal: Symbol referencing errors. No output written to client
```

If this happens on your system, you must include the math library by adding `-lm` to the end of the compile/link line.

22.2.14. Building Client Programs

If you compile MySQL clients that you've written yourself or that you obtain from a third-party, they must be linked using the `-lmysqlclient -lz` option on the link command. You may also need to specify a `-L` option to tell the linker where to find the library. For example, if the library is installed in `/usr/local/mysql/lib`, use `-L/usr/local/mysql/lib -lmysqlclient -lz` on the link command.

For clients that use MySQL header files, you may need to specify a `-I` option when you compile them (for example, `-I/usr/local/mysql/include`), so the compiler can find the header files.

To make it simpler to compile MySQL programs on Unix, we have provided the **mysql_config** script for you. See Section 22.1.2, "**mysql_config**, Get compile options for compiling clients".

You can use it to compile a MySQL client as follows:

```
CFG=/usr/local/mysql/bin/mysql_config
sh -c "gcc -o progname `$CFG --cflags` progname.c `$CFG --libs`"
```

The `sh -c` is needed to get the shell not to treat the output from **mysql_config** as one word.

22.2.15. How to Make a Threaded Client

The client library is almost thread-safe. The biggest problem is that the subroutines in `net.c` that read from sockets are not interrupt safe. This was done with the thought that you might want to have your own alarm that can break a long read to a server. If you install interrupt handlers for the `SIGPIPE` interrupt, the socket handling should be thread-safe.

New in 4.0.16: To not abort the program when a connection terminates, MySQL blocks `SIGPIPE` on the first call to `mysql_server_init`(), `mysql_init()` or `mysql_connect()`. If you want to have your own `SIGPIPE` handler, you should first call `mysql_server_init()` and then install your handler. In older versions of MySQL `SIGPIPE` was blocked, but only in the thread safe client library, for every call to `mysql_init()`.

In the older binaries we distribute on our Web site (http://www.mysql.com/), the client libraries are not normally compiled with the thread-safe option (the Windows binaries are by default compiled to be thread-safe). Newer binary distributions should have both a normal and a thread-safe client library.

To get a threaded client where you can interrupt the client from other threads and set timeouts when talking with the MySQL server, you should use the `-lmysys`, `-lmystrings`, and `-ldbug` libraries and the `net_serv.o` code that the server uses.

If you don't need interrupts or timeouts, you can just compile a thread-safe client library (`mysqlclient_r`) and use this. See Section 22.2, “MySQL C API”. In this case, you don't have to worry about the `net_serv.o` object file or the other MySQL libraries.

When using a threaded client and you want to use timeouts and interrupts, you can make great use of the routines in the `thr_alarm.c` file. If you are using routines from the `mysys` library, the only thing you must remember is to call `my_init()` first! See Section 22.2.11, “C API Threaded Function Descriptions”.

All functions except `mysql_real_connect()` are by default thread-safe. The following notes describe how to compile a thread-safe client library and use it in a thread-safe manner. (The notes below for `mysql_real_connect()` actually apply to `mysql_connect()` as well, but because `mysql_connect()` is deprecated, you should be using `mysql_real_connect()` anyway.)

To make `mysql_real_connect()` thread-safe, you must recompile the client library with this command:

```
shell> ./configure --enable-thread-safe-client
```

This creates a thread-safe client library `libmysqlclient_r`. (Assuming that your OS has a thread-safe `gethostbyname_r()` function.) This library is thread-safe per connection. You can let two threads share the same connection with the following caveats:

- Two threads can't send a query to the MySQL server at the same time on the same connection. In particular, you have to ensure that between a `mysql_query()` and `mysql_store_result()` no other thread is using the same connection.
- Many threads can access different result sets that are retrieved with `mysql_store_result()`.
- If you use `mysql_use_result`, you have to ensure that no other thread is using the same connection until the result set is closed. However, it really is best for threaded clients that share the same connection to use `mysql_store_result()`.
- If you want to use multiple threads on the same connection, you must have a mutex lock around your `mysql_query()` and `mysql_store_result()` call combination. Once `mysql_store_result()` is ready, the lock can be released and other threads may query the same connection.
- If you program with POSIX threads, you can use `pthread_mutex_lock()` and `pthread_mutex_unlock()` to establish and release a mutex lock.

You need to know the following if you have a thread that is calling MySQL functions which did not create the connection to the MySQL database:

When you call `mysql_init()` or `mysql_connect()`, MySQL creates a thread-specific variable for the thread that is used by the debug library (among other things).

If you call a MySQL function, before the thread has called `mysql_init()` or `mysql_connect()`, the thread does have the necessary thread-specific variables in place and you are likely to end up with a core dump sooner or later.

The get things to work smoothly you have to do the following:

1. Call `my_init()` at the start of your program if it calls any other MySQL function before calling `mysql_real_connect()`.
2. Call `mysql_thread_init()` in the thread handler before calling any MySQL function.
3. In the thread, call `mysql_thread_end()` before calling `pthread_exit()`. This frees the memory used by MySQL thread-specific variables.

You may get some errors because of undefined symbols when linking your client with `libmysqlclient_r`. In most cases this is because you haven't included the thread libraries on the link/compile line.

22.2.16. libmysqld, the Embedded MySQL Server Library

22.2.16.1. Overview of the Embedded MySQL Server Library

The embedded MySQL server library makes it possible to run a full-featured MySQL server inside a client application. The main benefits are increased speed and more simple management for embedded applications.

The embedded server library is based on the client/server version of MySQL, which is written in C/C++. Consequently, the embedded server also is written in C/C++. There is no embedded server available in other languages.

The API is identical for the embedded MySQL version and the client/server version. To change an old threaded application to use the embedded library, you normally only have to add calls to the following functions:

Function	When to Call
`mysql_server_init()`	Should be called before any other MySQL function is called, preferably early in the `main()` function.
`mysql_server_end()`	Should be called before your program exits.
`mysql_thread_init()`	Should be called in each thread you create that accesses MySQL.
`mysql_thread_end()`	Should be called before calling `pthread_exit()`

Then you must link your code with `libmysqld.a` instead of `libmysqlclient.a`.

The `mysql_server_xxx()` functions are also included in `libmysqlclient.a` to allow you to change between the embedded and the client/server version by just linking your application with the right library. See Section 22.2.12.1, “`mysql_server_init()`”.

22.2.16.2. Compiling Programs with `libmysqld`

To get a `libmysqld` library you should configure MySQL with the `--with-embedded-server` option.

When you link your program with `libmysqld`, you must also include the system-specific `pthread` libraries and some libraries that the MySQL server uses. You can get the full list of libraries by executing **mysql_config --libmysqld-libs**.

The correct flags for compiling and linking a threaded program must be used, even if you do not directly

call any thread functions in your code.

22.2.16.3. Restrictions when using the Embedded MySQL Server

The embedded server has the following limitations:

- No support for `ISAM` tables. (This is mainly done to make the library smaller)
- No user-defined functions (UDFs).
- No stack trace on core dump.
- No internal RAID support. (This is not normally needed as most current operating systems support big files).
- You cannot set this up as a master or a slave (no replication).
- You can't connect to an embedded server from an outside process with sockets or TCP/IP.

Some of these limitations can be changed by editing the `mysql_embed.h` include file and recompiling MySQL.

22.2.16.4. Using Option Files with the Embedded Server

The following is the recommended way to use option files to make it easy to switch between a client/server application and one where MySQL is embedded. See Section 4.3.2, “Using Option Files”.

- Put common options in the `[server]` section. These are read by both MySQL versions.
- Put client/server-specific options in the `[mysqld]` section.
- Put embedded MySQL-specific options in the `[embedded]` section.
- Put application-specific options in a `[ApplicationName_SERVER]` section.

22.2.16.5. Things left to do in Embedded Server (TODO)

- We are going to provide options to leave out some parts of MySQL to make the library smaller.
- There is still a lot of speed optimization to do.
- Errors are written to `stderr`. We will add an option to specify a filename for these.
- We have to change `InnoDB` to not be so verbose when using in the embedded version.

22.2.16.6. A Simple Embedded Server Example

This example program and makefile should work without any changes on a Linux or FreeBSD system. For other operating systems, minor changes are needed. This example is designed to give enough details to understand the problem, without the clutter that is a necessary part of a real application.

To try out the example, create an `test_libmysqld` directory at the same level as the mysql-4.0

source directory. Save the `test_libmysqld.c` source and the `GNUmakefile` in the directory, and run GNU `make` from inside the `test_libmysqld` directory.

```
test_libmysqld.c

/*
 * A simple example client, using the embedded MySQL server library
 */

#include <mysql.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>

MYSQL *db_connect(const char *dbname);
void db_disconnect(MYSQL *db);
void db_do_query(MYSQL *db, const char *query);

const char *server_groups[] = {
  "test_libmysqld_SERVER", "embedded", "server", NULL
};

int
main(int argc, char **argv)
{
  MYSQL *one, *two;

  /* mysql_server_init() must be called before any other mysql
   * functions.
   *
   * You can use mysql_server_init(0, NULL, NULL), and it
   * initializes the server using groups = {
   *   "server", "embedded", NULL
   *  }.
   *
   * In your $HOME/.my.cnf file, you probably want to put:

[test_libmysqld_SERVER]
language = /path/to/source/of/mysql/sql/share/english

   * You could, of course, modify argc and argv before passing
   * them to this function.  Or you could create new ones in any
   * way you like.  But all of the arguments in argv (except for
   * argv[0], which is the program name) should be valid options
   * for the MySQL server.
   *
   * If you link this client against the normal mysqlclient
   * library, this function is just a stub that does nothing.
   */
  mysql_server_init(argc, argv, (char **)server_groups);

  one = db_connect("test");
  two = db_connect(NULL);

  db_do_query(one, "SHOW TABLE STATUS");
  db_do_query(two, "SHOW DATABASES");

  mysql_close(two);
  mysql_close(one);

  /* This must be called after all other mysql functions */
  mysql_server_end();

  exit(EXIT_SUCCESS);
}
```

```
static void
die(MYSQL *db, char *fmt, ...)
{
  va_list ap;
  va_start(ap, fmt);
  vfprintf(stderr, fmt, ap);
  va_end(ap);
  (void)putc('\n', stderr);
  if (db)
    db_disconnect(db);
  exit(EXIT_FAILURE);
}

MYSQL *
db_connect(const char *dbname)
{
  MYSQL *db = mysql_init(NULL);
  if (!db)
    die(db, "mysql_init failed: no memory");
  /*
   * Notice that the client and server use separate group names.
   * This is critical, because the server does not accept the
   * client's options, and vice versa.
   */
  mysql_options(db, MYSQL_READ_DEFAULT_GROUP, "test_libmysqld_CLIENT");
  if (!mysql_real_connect(db, NULL, NULL, NULL, dbname, 0, NULL, 0))
    die(db, "mysql_real_connect failed: %s", mysql_error(db));

  return db;
}

void
db_disconnect(MYSQL *db)
{
  mysql_close(db);
}

void
db_do_query(MYSQL *db, const char *query)
{
  if (mysql_query(db, query) != 0)
    goto err;

  if (mysql_field_count(db) > 0)
  {
    MYSQL_RES   *res;
    MYSQL_ROW    row, end_row;
    int num_fields;

    if (!(res = mysql_store_result(db)))
      goto err;
    num_fields = mysql_num_fields(res);
    while ((row = mysql_fetch_row(res)))
    {
      (void)fputs(">> ", stdout);
      for (end_row = row + num_fields; row < end_row; ++row)
        (void)printf("%s\t", row ? (char*)*row : "NULL");
      (void)fputc('\n', stdout);
    }
    (void)fputc('\n', stdout);
    mysql_free_result(res);
  }
  else
```

```
    (void)printf("Affected rows: %lld\n", mysql_affected_rows(db));

  return;

err:
  die(db, "db_do_query failed: %s [%s]", mysql_error(db), query);
}
```

```
GNUmakefile

# This assumes the MySQL software is installed in /usr/local/mysql
inc      := /usr/local/mysql/include/mysql
lib      := /usr/local/mysql/lib

# If you have not installed the MySQL software yet, try this instead
#inc      := $(HOME)/mysql-4.0/include
#lib      := $(HOME)/mysql-4.0/libmysqld

CC       := gcc
CPPFLAGS := -I$(inc) -D_THREAD_SAFE -D_REENTRANT
CFLAGS   := -g -W -Wall
LDFLAGS  := -static
# You can change -lmysqld to -lmysqlclient to use the
# client/server library
LDLIBS    = -L$(lib) -lmysqld -lz -lm -lcrypt

ifneq (,$(shell grep FreeBSD /COPYRIGHT 2>/dev/null))
# FreeBSD
LDFLAGS += -pthread
else
# Assume Linux
LDLIBS += -lpthread
endif

# This works for simple one-file test programs
sources := $(wildcard *.c)
objects := $(patsubst %c,%o,$(sources))
targets := $(basename $(sources))

all: $(targets)

clean:
        rm -f $(targets) $(objects) *.core
```

22.2.16.7. Licensing the Embedded Server

We encourage everyone to promote free software by releasing code under the GPL or a compatible license. For those who are not able to do this, another option is to purchase a commercial license for the MySQL code from MySQL AB. For details, please see http://www.mysql.com/company/legal/licensing/.

22.3. MySQL PHP API

PHP is a server-side, HTML-embedded scripting language that may be used to create dynamic Web pages. It contains support for accessing several databases, including MySQL. PHP may be run as a separate program or compiled as a module for use with the Apache Web server.

The distribution and documentation are available at the PHP Web site (http://www.php.net/).

22.3.1. Common Problems with MySQL and PHP

- Error: "Maximum Execution Time Exceeded" This is a PHP limit; go into the `php.ini` file and set the maximum execution time up from 30 seconds to something higher, as needed. It is also not a bad idea to double the RAM allowed per script to 16MB instead of 8MB.
- Error: "Fatal error: Call to unsupported or undefined function mysql_connect() in .." This means that your PHP version isn't compiled with MySQL support. You can either compile a dynamic MySQL module and load it into PHP or recompile PHP with built-in MySQL support. This is described in detail in the PHP manual.
- Error: "undefined reference to `uncompress'" This means that the client library is compiled with support for a compressed client/server protocol. The fix is to add `-lz` last when linking with `-lmysqlclient`.

22.4. MySQL Perl API

The Perl `DBI` module provides a generic interface for database access. You can write a DBI script that works with many different database engines without change. To use DBI, you must install the `DBI` module, as well as a DataBase Driver (DBD) module for each type of server you want to access. For MySQL, this driver is the `DBD::mysql` module.

Perl DBI is the recommended Perl interface. It replaces an older interface called `mysqlperl`, which should be considered obsolete.

Installation instructions for Perl DBI support are given in Section 2.13, “Perl Installation Notes”.

DBI information is available at the command line, online, or in printed form:

- Once you have the `DBI` and `DBD::mysql` modules installed, you can get information about them at the command line with the `perldoc` command:

  ```
  shell> perldoc DBI
  shell> perldoc DBI::FAQ
  shell> perldoc DBD::mysql
  ```

  You can also use `pod2man`, `pod2html`, and so forth to translate this information into other formats.
- For online information about Perl DBI, visit the DBI Web site, http://dbi.perl.org/. That site hosts a general DBI mailing list. MySQL AB hosts a list specifically about `DBD::mysql`; see Section 1.4.1.1, “The MySQL Mailing Lists”.
- For printed information, the official DBI book is *Programming the Perl DBI* (Alligator Descartes and Tim Bunce, O'Reilly & Associates, 2000). Information about the book is available at the DBI Web site, http://dbi.perl.org/.

  For information that focuses specifically on using DBI with MySQL, see *MySQL and Perl for the Web* (Paul DuBois, New Riders, 2001). This book's Web site is http://www.kitebird.com/mysql-perl/.

22.5. MySQL C++ API

`MySQL++` is the MySQL API for C++. More information can be found at http://www.mysql.com/products/mysql++/.

22.5.1. Borland C++

You can compile the MySQL Windows source with Borland C++ 5.02. (The Windows source includes only projects for Microsoft VC++, for Borland C++ you have to do the project files yourself.)

One known problem with Borland C++ is that it uses a different structure alignment than VC++. This means that you run into problems if you try to use the default `libmysql.dll` libraries (that were compiled with VC++) with Borland C++. To avoid this problem, only call `mysql_init()` with `NULL` as an argument, not a pre-allocated `MYSQL` structure.

22.6. MySQL Python API

`MySQLdb` provides MySQL support for Python, compliant with the Python DB API version 2.0. It can be found at http://sourceforge.net/projects/mysql-python/.

22.7. MySQL Tcl API

`MySQLtcl` is a simple API for accessing a MySQL database server from the Tcl programming language. It can be found at http://www.xdobry.de/mysqltcl/.

22.8. MySQL Eiffel Wrapper

Eiffel MySQL is an interface to the MySQL database server using the Eiffel programming language, written by Michael Ravits. It can be found at http://efsa.sourceforge.net/archive/ravits/mysql.htm.

Chapter 23. MySQL Connectors

This chapter describes MySQL Connectors, drivers that provide connectivity to the MySQL server for client programs.

23.1. MySQL ODBC Support

MySQL provides support for ODBC by means of MySQL Connector/ODBC, the family of MyODBC drivers. This is the reference for the Connector/ODBC product family of MyODBC drivers that provide ODBC 3.5x compliant access to the MySQL Database System. It teaches you how to install MyODBC and how to use it. There is also information about common programs that are known to work with MyODBC and answers to some of the most frequently asked questions about MyODBC.

This reference applies to MyODBC 3.51. You can find a manual for an older version of MyODBC in the binary or source distribution for that version.

This is a reference to the MySQL ODBC drivers, not a general ODBC reference. For more information about ODBC, refer to http://www.microsoft.com/data/.

The application development part of this reference assumes a good working knowledge of C, general DBMS knowledge, and finally, but not least, familiarity with MySQL. For more information about MySQL functionality and its syntax, refer to http://dev.mysql.com/doc/.

If you have questions that are not answered in this document, please send a mail message to `<myodbc@lists.mysql.com>`.

23.1.1. Introduction to MyODBC

23.1.1.1. What is ODBC?

ODBC (Open Database Connectivity) provides a way for client programs to access a wide range of databases or data sources. ODBC is a standardized API that allows connections to SQL database servers. It was developed according to the specifications of the SQL Access Group and defines a set of function calls, error codes, and data types that can be used to develop database-independent applications. ODBC usually is used when database independence or simultaneous access to different data sources is required.

For more information about ODBC, refer to http://www.microsoft.com/data/.

23.1.1.2. What is Connector/ODBC?

Connector/ODBC is the term designating the MySQL AB product family of MySQL ODBC drivers. These are known as the MyODBC drivers.

23.1.1.3. What is MyODBC 2.50?

MyODBC 2.50 is a 32-bit ODBC driver from MySQL AB that is based on ODBC 2.50 specification level 0 (with level 1 and 2 features). This is one of the most popular ODBC drivers in the Open Source market, used by many users to access the MySQL functionality.

23.1.1.4. What is MyODBC 3.51?

MyODBC 3.51 is a 32-bit ODBC driver, also known as the MySQL ODBC 3.51 driver. This version is enhanced compared to the existing MyODBC 2.50 driver. It has support for ODBC 3.5x specification level 1 (complete core API + level 2 features) in order to continue to provide all functionality of ODBC

for accessing MySQL.

23.1.1.5. Where to Get MyODBC

MySQL AB distributes all its products under the General Public License (GPL). You can get a copy of the latest version of MyODBC binaries and sources from the MySQL AB Web site http://dev.mysql.com/downloads/.

For more information about MyODBC, visit http://www.mysql.com/products/myodbc/.

For more information about licensing, visit http://www.mysql.com/company/legal/licensing/.

23.1.1.6. Supported Platforms

MyODBC can be used on all major platforms supported by MySQL, such as:

- Windows 95, 98, Me, NT, 2000, XP, and 2003
- All Unix Operating Systems
  - AIX
  - Amiga
  - BSDI
  - DEC
  - FreeBSD
  - HP-UX 10, 11
  - Linux
  - Mac OS X Server
  - Mac OS X
  - NetBSD
  - OpenBSD
  - OS/2
  - SGI Irix
  - Solaris
  - SunOS
  - SCO OpenServer
  - SCO UnixWare
  - Tru64 Unix

If a binary distribution is not available for downloading for a particular platform, you can build the

driver yourself by downloading the driver sources. You can contribute the binaries to MySQL by sending a mail message to `<myodbc@lists.mysql.com>`, so that it becomes available for other users.

23.1.1.7. MyODBC Mailing List

MySQL AB provides assistance to the user community by means of its mailing lists. For MyODBC-related issues, you can get help from experienced users by using the `<myodbc@lists.mysql.com>` mailing list.

For information about subscribing to MySQL mailing lists or to browse list archives, visit http://lists.mysql.com/.

Of particular interest is the ODBC forum in the MySQL Connectors section of the forums.

23.1.1.8. MyODBC Forum

Community support from experienced users is available through the MySQL Forums, located at http://forums.mysql.com.

23.1.1.9. How to Report MyODBC Problems or Bugs

If you encounter difficulties or problems with MyODBC, you should start by making a log file from the `ODBC Manager` (the log you get when requesting logs from `ODBC ADMIN`) and MyODBC. The procedure for doing this is described in Section 23.1.9.7, “Getting an ODBC Trace File”.

Check the MyODBC trace file to find out what could be wrong. You should be able to determine what statements were issued by searching for the string `>mysql_real_query` in the `myodbc.log` file.

You should also try issuing the statements from the **mysql** client program or from `admndemo`. This helps you determine whether the error is in MyODBC or MySQL.

If you find out something is wrong, please only send the relevant rows (maximum 40 rows) to the `myodbc` mailing list. See Section 1.4.1.1, “The MySQL Mailing Lists”. Please never send the whole MyODBC or ODBC log file!

If you are unable to find out what's wrong, the last option is to create an archive in **tar** or Zip format that contains a MyODBC trace file, the ODBC log file, and a `README` file that explains the problem. You can send this to ftp://ftp.mysql.com/pub/mysql/upload/. Only we at MySQL AB has access to the files you upload, and we are very discreet with the data.

If you can create a program that also demonstrates the problem, please include it in the archive as well.

If the program works with some other SQL server, you should include an ODBC log file where you do exactly the same thing in the other SQL server.

Remember that the more information you can supply to us, the more likely it is that we can fix the problem.

23.1.1.10. How to Submit a MyODBC Patch

You can send a patch or suggest a better solution for any existing code or problems by sending a mail message to `<myodbc@lists.mysql.com>`.

23.1.2. General Information About ODBC and MyODBC

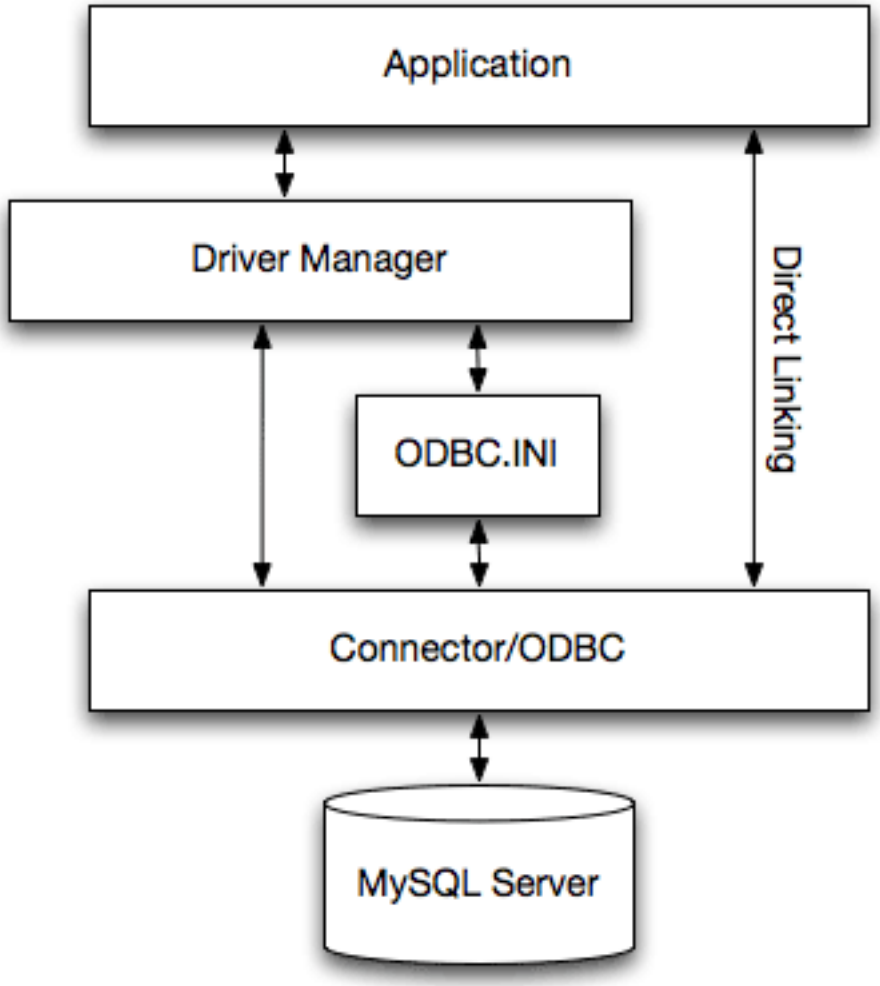
23.1.2.1. Introduction to ODBC

Open Database Connectivity (ODBC) is a widely accepted application-programming interface (API) for database access. It is based on the Call-Level Interface (CLI) specifications from X/Open and ISO/IEC for database APIs and uses Structured Query Language (SQL) as its database access language.

A survey of ODBC functions supported by MyODBC is given at Section 23.1.16, “MyODBC API Reference”. For general information about ODBC, see http://www.microsoft.com/data/.

23.1.2.2. MyODBC Architecture

The MyODBC architecture is based on five components, as shown in the following diagram:



- **Application:**

  An application is a program that calls the ODBC API to access the data from the MySQL server. The Application communicates with the Driver Manager using the standard ODBC calls. The Application does not care where the data is stored, how it is stored, or even how the system is configured to access the data. It needs to know only the Data Source Name (DSN).

  A number of tasks are common to all applications, no matter how they use ODBC. These tasks are:

- Selecting the MySQL server and connecting to it
- Submitting SQL statements for execution
- Retrieving results (if any)
- Processing errors
- Committing or rolling back the transaction enclosing the SQL statement
- Disconnecting from the MySQL server

Because most data access work is done with SQL, the primary tasks for applications that use ODBC are submitting SQL statements and retrieving any results generated by those statements.

- **Driver manager:**

  The Driver Manager is a library that manages communication between application and driver or drivers. It performs the following tasks:

  - Resolves Data Source Names (DSN)
  - Driver loading and unloading
  - Processes ODBC function calls or passes them to the driver

- **MyODBC Driver:**

  The MyODBC driver is a library that implements the functions in the ODBC API. It processes ODBC function calls, submits SQL requests to MySQL server, and returns results back to the application. If necessary, the driver modifies an application's request so that the request conforms to syntax supported by the MySQL.

- **ODBC.INI:**

  `ODBC.INI` is the ODBC configuration file that stores the driver and database information required to connect to the server. It is used by the Driver Manager to determine which driver to be loaded using the Data Source Name. The driver uses this to read connection parameters based on the DSN specified. For more information, Section 23.1.9, “MyODBC Configuration”.

- **MySQL Server:**

  The MySQL server is the source of data. MySQL is:

  - A database management system (DBMS)
  - A relational database management system (RDBMS)
  - Open Source Software

23.1.2.3. ODBC Driver Managers

An ODBC Driver Manager is a library that manages communication between the ODBC aware application and driver(s). Its main functionality includes:

- Resolving Data Source Names (DSN)
- Driver loading and unloading
- Processing ODBC function calls or passing them to the driver

The following driver managers are commonly used:

- Microsoft Windows ODBC Driver Manager (`odbc32.dll`), http://www.microsoft.com/data/
- unixODBC Driver Manager for Unix (`libodbc.so`), http://www.unixodbc.org.
- iODBC ODBC Driver Manager for Unix (`libiodbc.so`), http://www.iodbc.org

MyODBC 3.51 also is shipped with UnixODBC beginning with version 2.1.2.

23.1.2.4. Types of MySQL ODBC Drivers

MySQL AB supports two Open Source ODBC drivers for accessing MySQL functionality through the ODBC API: MyODBC (MyODBC 2.50) and MySQL ODBC 3.51 Driver (MyODBC 3.51).

Note: From this section onward, we refer both the drivers generically as MyODBC. Whenever there is a difference, we use the original names.

23.1.3. How to Install MyODBC

MyODBC works on Windows 9x, Me, NT, 2000, XP, and 2003, and on most Unix platforms.

MyODBC is Open Source. You can find the newest version at http://dev.mysql.com/downloads/connector/odbc/. Please note that the 2.50.x versions are LGPL licensed, whereas the 3.51.x versions are GPL licensed.

If you have problem with MyODBC and your program also works with OLEDB, you should try the OLEDB driver.

Normally you need to install MyODBC only on Windows machines. You need MyODBC for Unix only if you have a program like ColdFusion that is running on a Unix machine and uses ODBC to connect for database access.

If you want to install MyODBC on a Unix box, you also need an ODBC manager. MyODBC is known to work with most Unix ODBC managers.

- To make a connection to a Unix box from a Windows box with an ODBC application (one that doesn't support MySQL natively), you must first install MyODBC on the Windows machine.
- The user and Windows machine must have access privileges for the MySQL server on the Unix machine. This is set up with the `GRANT` command. See Section 13.5.1.3, “`GRANT` and `REVOKE` Syntax”.
- You must create an ODBC DSN entry as follows:
  1. Open the Control Panel on the Windows machine.
  2. Double-click the `ODBC Data Sources 32-bit` icon.

3. Click the tab `User DSN`.
4. Click the `Add` button.
5. Select MySQL in the screen `Create New Data Source` and click the `Finish` button.
6. The MySQL Driver default configuration screen is shown. See Section 23.1.9.2, “Configuring a MyODBC DSN on Windows”.

- Start your application and select the ODBC driver with the DSN that you specified in the ODBC administrator.

Notice that other configuration options are shown on the MySQL screen that you can try if you run into problems (options such as trace, don't prompt on connect, and so forth).

23.1.4. Installing MyODBC from a Binary Distribution on Windows

To install MyODBC on Windows, you should download the appropriate distribution file from http://dev.mysql.com/downloads/connector/odbc/, unpack it, and execute the **MyODBC-*VERSION*.exe** file.

On Windows, you may get the following error when trying to install the older MyODBC 2.50 driver:

```
An error occurred while copying C:\WINDOWS\SYSTEM\MFC30.DLL. Restart
Windows and try installing again (before running any applications
which use ODBC)
```

The problem is that some other program is using ODBC. Because of how Windows is designed, you may not be able in this case to install new ODBC drivers with Microsoft's ODBC setup program. In most cases, you can continue by pressing `Ignore` to copy the rest of the MyODBC files and the final installation should still work. If it doesn't, the solution is to re-boot your computer in ``safe mode.'' Choose safe mode by pressing F8 just before your machine starts Windows during re-booting, install MyODBC, and re-boot to normal mode.

23.1.5. Installing MyODBC from a Binary Distribution on Unix

23.1.5.1. Installing MyODBC from an RPM Distribution

To install or upgrade MyODBC from an RPM distribution on Linux, simply download the RPM distribution of the latest version of MyODBC and follow the instructions below. Use **su root** to become `root`, then install the RPM file.

If you are installing for the first time:

```
shell> su root
shell> rpm -ivh MyODBC-3.51.01.i386-1.rpm
```

If the driver exists, upgrade it like this:

```
shell> su root
```

```
shell> rpm -Uvh MyODBC-3.51.01.i386-1.rpm
```

If there is any dependency error for MySQL client library, `libmysqlclient`, simply ignore it by supplying the `--nodeps` option, and then make sure the MySQL client shared library is in the path or set through `LD_LIBRARY_PATH`.

This installs the driver libraries and related documents to `/usr/local/lib` and `/usr/share/doc/MyODBC` respectively. Proceed onto Section 23.1.9.3, “Configuring a MyODBC DSN on Unix”.

To **uninstall** the driver, become `root` and execute an **rpm** command:

```
shell> su root
shell> rpm -e MyODBC
```

23.1.5.2. Installing MyODBC from a Binary Tarball Distribution

To install the driver from a tarball distribution (`.tar.gz` file), download the latest version of the driver for your operating system and follow these steps:

```
shell> su root
shell> gunzip MyODBC-3.51.01-i686-pc-linux.tar.gz
shell> tar xvf MyODBC-3.51.01-i686-pc-linux.tar
shell> cd MyODBC-3.51.01-i686-pc-linux
```

Read the installation instructions in the `INSTALL-BINARY` file and execute these commands.

```
shell> cp libmyodbc* /usr/local/lib
shell> cp odbc.ini /usr/local/etc
shell> export ODBCINI=/usr/local/etc/odbc.ini
```

Then proceed on to Section 23.1.9.3, “Configuring a MyODBC DSN on Unix” to configure the DSN for MyODBC. For more information, refer to the `INSTALL-BINARY` file that comes with your distribution.

23.1.6. Installing MyODBC from a Source Distribution on Windows

23.1.6.1. Requirements

- MDAC, Microsoft Data Access SDK from http://www.microsoft.com/data/.
- MySQL client libraries and include files from MySQL 4.0.0 or higher. (Preferably MySQL 4.0.16 or higher). This is required because MyODBC uses new calls and structures that exist only starting from this version of the library. To get the client libraries and include files, visit http://dev.mysql.com/downloads/.

23.1.6.2. Building MyODBC 3.51

MyODBC 3.51 source distributions include `Makefiles` that uses **nmake**. In the distribution, you can find `Makefile` for building the release version and `Makefile_debug` for building debugging ver-

sions of the driver libraries and DLLs.

To build the driver, use this procedure:

1. Download and extract the sources to a folder, then change location into that folder. The following command assumes the folder is named `myodbc3-src`:

   ```
   C:\> cd myodbc3-src
   ```

2. Edit `Makefile` to specify the correct path for the MySQL client libraries and header files. Then use the following commands to build and install the release version:

   ```
   C:\> nmake -f Makefile
   C:\> nmake -f Makefile install
   ```

   nmake -f Makefile builds the release version of the driver and places the binaries in subdirectory called `Release`.

   nmake -f Makefile install installs (copies) the driver DLLs and libraries(`myodbc3.dll`, `myodbc3.lib`) to your system directory.

3. To build the debug version, use `Makefile_Debug` rather than `Makefile`, as shown below:

   ```
   C:\> nmake -f Makefile_debug
   C:\> nmake -f Makefile_debug install
   ```

4. You can clean and rebuild the driver by using:

   ```
   C:\> nmake -f Makefile clean
   C:\> nmake -f Makefile install
   ```

Note:

- Make sure to specify the correct MySQL client libraries and header files path in the Makefiles (set the `MYSQL_LIB_PATH` and `MYSQL_INCLUDE_PATH` variables). The default header file path is assumed to be `C:\mysql\include`. The default library path is assumed to be `C:\mysql\lib\opt` for release DLLs and `C:\mysql\lib\debug` for debug versions.
- For the complete usage of **nmake**, visit http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vcce4/html/evgrfRunningNMAKE.asp.
- If you are using the BitKeeper tree for compiling, All Windows-specific `Makefiles` are named as `Win_Makefile*`.

23.1.6.3. Testing

After the driver libraries are copied/installed to the system directory, you can test whether the libraries are properly built by using the samples provided in the `samples` subdirectory:

```
C:\> cd samples
C:\> nmake -f Makefile all
```

23.1.6.4. Building MyODBC 2.50

The MyODBC 2.50 source distribution includes VC workspace files. You can build the driver using these files (`.dsp` and `.dsw`) directly by loading them from Microsoft Visual Studio 6.0 or higher.

23.1.7. Installing MyODBC from a Source Distribution on Unix

23.1.7.1. Requirements

- MySQL client libraries and include files from MySQL 4.0.0 or higher. (Preferably MySQL 4.0.16 or higher). This is required because MyODBC uses new calls and structures that exist only starting from this version of the library. To get the client libraries and include files, visit http://dev.mysql.com/downloads/.
- The MySQL library must be configured with the `--enable-thread-safe-client` option. libmysqlclient installed as a shared library.
- One of the following Unix ODBC driver managers must be installed:
  - `iodbc` 3.0 or later (http://www.iodbc.org)
  - `unixodbc` Alpha 3 or later (http://www.unixodbc.org)
- If using a **character set** that isn't compiled into the MySQL client library (the defaults are: latin1 big5 czech euc_kr gb2312 gbk sjis tis620 ujis) then you need to install the mysql character definitions from the `charsets` directory into *`SHAREDIR`* (by default, `/usr/local/mysql/share/mysql/charsets`). These should be in place if you have installed the MySQL server on the same machine.

Once you have all the required files, unpack the source files to a separate directory and follow the instructions as given below:

23.1.7.2. Typical configure Options

The **configure** script gives you a great deal of control over how you configure your MyODBC build. Typically you do this using options on the **configure** command line. You can also affect **configure** using certain environment variables. For a list of options and environment variables supported by **configure**, run this command:

```
shell> ./configure --help
```

Some of the more commonly used **configure** options are described here:

1. To compile MyODBC, you need to supply the MySQL client include and library files path using the `--with-mysql-path=`*`DIR`* option, where *`DIR`* is the directory where the MySQL is installed.

   MySQL compile options can be determined by running *`DIR`*`/bin/mysql_config`.

2. Supply the standard header and library files path for your ODBC Driver Manager(`iodbc` or `unixobc`).

- If you are using `iodbc` and `iodbc` is not installed in its default location (`/usr/local`), you might have to use the `--with-iodbc=DIR` option, where *`DIR`* is the directory where iodbc is installed.

  If the iodbc headers do not reside in *`DIR`*`/include`, you can use the `--with-iodbc-includes=INCDIR` option to specify their location.

  The applies to libraries. If they are not in *`DIR`*`/lib`, you can use the `--with-iodbc-libs=LIBDIR` option.

- If you are using `unixODBC`, use the `--with-unixODBC=DIR` option (case sensitive) to make **configure** look for `unixODBC` instead of `iodbc` by default, *`DIR`* is the directory where unixODBC is installed.

  If the unixODBC headers and libraries aren't located in *`DIR`*`/include` and *`DIR`*`/lib`, use the `--with-unixODBC-includes=INCDIR` and `--with-unixODBC-libs=LIBDIR` options.

3. You might want to specify an installation prefix other than `/usr/local`. For example, to install the MyODBC drivers in `/usr/local/odbc/lib`, use the `--prefix=/usr/local/odbc` option.

The final configuration command looks something like this:

```
shell> ./configure --prefix=/usr/local \
           --with-iodbc=/usr/local \
           --with-mysql-path=/usr/local/mysql
```

23.1.7.3. Thread-Safe Client

In order to link the driver with MySQL thread safe client libraries `libmysqlclient_r.so` or `libmysqlclient_r.a`, you must specify the following **configure** option:

```
--enable-thread-safe
```

and can be disabled(default) using

```
--disable-thread-safe
```

This option enables the building of driver thread-safe library `libmyodbc3_r.so` from by linking with mysql thread-safe client library `libmysqlclient_r.so` (The extensions are OS dependent).

In case while configuring with thread-safe option, and gotten into a configure error; then look at the `config.log` and see if it is due to the lack of thread-libraries in the system; and supply one with LIBS options i.e.

```
LIBS="-lpthread" ./configure ..
```

23.1.7.4. Shared or Static Options

You can enable or disable the shared and static versions using these options:

```
--enable-shared[=yes/no]
```

```
--disable-shared
--enable-static[=yes/no]
--disable-static
```

23.1.7.5. Enabling Debugging Information

By default, all the binary distributions are built as non-debugging versions (configured with `--without-debug`).

To enable debugging information, build the driver from source distribution and use the `--with-debug`) when you run **configure**.

23.1.7.6. Enabling the Documentation

This option is available only for `BK` clone trees; not for normal source distributions.

By default, the driver is built with (`--without-docs`); And in case if you want the documentation to be taken care in the normal build, then configure with:

```
--with-docs
```

23.1.7.7. Building and Compilation

To build the driver libraries, you have to just execute **make**, which takes care of everything.

```
shell> make
```

If any errors occur, correct them and continue the build process. If you aren't able to build, then send a detailed email to `<myodbc@lists.mysql.com>` for further assistance.

23.1.7.8. Building Shared Libraries

On most platforms, MySQL doesn't build or support `.so` (shared) client libraries by default, because building with shared libraries has caused us problems in the past.

In cases like this, you have to download the MySQL distribution and configure it with these options:

```
--without-server --enable-shared
```

To build shared driver libraries, you must specify the `--enable-shared` option for **configure**. By default, **configure** does not enable this option.

If you have configured with the `--disable-shared` option, you can build the `.so` file from the static libraries using the following commands:

```
shell> cd MyODBC-3.51.01
shell> make
shell> cd driver
shell> CC=/usr/bin/gcc \
       $CC -bundle -flat_namespace -undefined error \
           -o .libs/libmyodbc3-3.51.01.so \
           catalog.o connect.o cursor.o dll.o error.o execute.o \
           handle.o info.o misc.o myodbc3.o options.o prepare.o \
           results.o transact.o utility.o \
           -L/usr/local/mysql/lib/mysql/ \
```

```
          -L/usr/local/iodbc/lib/ \
          -lz -lc -lmysqlclient -liodbcinst
```

Make sure to change `-liodbcinst` to `-lodbcinst` if you are using unixODBC instead of iODBC, and configure the library paths accordingly.

This builds and places the `libmyodbc3-3.51.01.so` file in the `.libs` directory. Copy this file to MyODBC library directory (`/usr/local/lib` (or the `lib` directory under the installation directory that you supplied with the `--prefix`).

```
shell> cd .libs
shell> cp libmyodbc3-3.51.01.so /usr/local/lib
shell> cd /usr/local/lib
shell> ln -s libmyodbc3-3.51.01.so libmyodbc3.so
```

To build the thread-safe driver library:

```
shell> CC=/usr/bin/gcc \
       $CC -bundle -flat_namespace -undefined error
        -o .libs/libmyodbc3_r-3.51.01.so
        catalog.o connect.o cursor.o dll.o error.o execute.o
        handle.o info.o misc.o myodbc3.o options.o prepare.o
        results.o transact.o utility.o
        -L/usr/local/mysql/lib/mysql/
        -L/usr/local/iodbc/lib/
        -lz -lc -lmysqlclient_r -liodbcinst
```

23.1.7.9. Installing Driver Libraries

To install the driver libraries, execute the following command:

```
shell> make install
```

That command installs one of the following sets of libraries:

For MyODBC 3.51:

- `libmyodbc3.so`
- `libmyodbc3-3.51.01.so`, where 3.51.01 is the version of the driver
- `libmyodbc3.a`

For thread-safe MyODBC 3.51:

- `libmyodbc3_r.so`
- `libmyodbc3-3_r.51.01.so`
- `libmyodbc3_r.a`

For MyODBC 2.5.0:

- `libmyodbc.so`
- `libmyodbc-2.50.39.so`, where 2.50.39 is the version of the driver
- `libmyodbc.a`

For more information on build process, refer to the `INSTALL` file that comes with the source distribution. Note that if you are trying to use the **make** from Sun, you may end up with errors. On the other hand, GNU **gmake** should work fine on all platforms.

23.1.7.10. Testing MyODBC on Unix

To run the basic samples provided in the distribution with the libraries that you built, just execute:

```
shell> make test
```

Make sure the DSN 'myodbc3' is configured first in `odbc.ini` and environment variable `ODBCINI` is pointing to the right `odbc.ini` file; and MySQL server is running. You can find a sample `odbc.ini` with the driver distribution.

You can even modify the `samples/run-samples` script to pass the desired DSN, UID, and PASSWORD values as the command line arguments to each sample.

23.1.7.11. Mac OS X Notes

To build the driver on Mac OS X (Darwin), make use of the following **configure** example:

```
shell> ./configure --prefix=/usr/local
             --with-unixODBC=/usr/local
             --with-mysql-path=/usr/local/mysql
             --disable-shared
             --enable-gui=no
             --host=powerpc-apple
```

The command assumes that the unixODBC and MySQL are installed in the default locations. If not, configure accordingly.

On Mac OS X, `--enable-shared` builds `.dylib` files by default. You can build `.so` files like this:

```
shell> make
shell> cd driver
shell> CC=/usr/bin/gcc \
       $CC -bundle -flat_namespace -undefined error
           -o .libs/libmyodbc3-3.51.01.so *.o
           -L/usr/local/mysql/lib/
           -L/usr/local/iodbc/lib
           -liodbcinst -lmysqlclient -lz -lc
```

To build the thread-safe driver library:

```
shell> CC=/usr/bin/gcc \
       $CC -bundle -flat_namespace -undefined error
       -o .libs/libmyodbc3-3.51.01.so *.o
       -L/usr/local/mysql/lib/
       -L/usr/local/iodbc/lib
       -liodbcinst -lmysqlclienti_r -lz -lc -lpthread
```

Make sure to change the `-liodbcinst` to `-lodbcinst` in case of using unixODBC instead of iODBC and configure the libraries path accordingly.

In Apple's version of GCC, both **cc** and **gcc** are actually symbolic links to **gcc3**.

Copy this library to the `$prefix/lib` directory and symlink to `libmyodbc3.so`.

You can cross-check the output shared-library properties using this command:

```
shell> otool -LD .libs/libmyodbc3-3.51.01.so
```

23.1.7.12. HP-UX Notes

To build the driver on HP-UX 10.x or 11.x, make use of the following **configure** example:

If using **cc**:

```
shell> CC="cc" \
       CFLAGS="+z" \
       LDFLAGS="-Wl,+b:-Wl,+s" \
       ./configure --prefix=/usr/local
             --with-unixodbc=/usr/local
             --with-mysql-path=/usr/local/mysql/lib/mysql
             --enable-shared
             --enable-thread-safe
```

If using **gcc**:

```
shell> CC="gcc" \
       LDFLAGS="-Wl,+b:-Wl,+s" \
       ./configure --prefix=/usr/local
             --with-unixodbc=/usr/local
             --with-mysql-path=/usr/local/mysql
             --enable-shared
             --enable-thread-safe
```

Once the driver is built, cross-check its attributes using **chatr .libs/libmyodbc3.sl** to see whether or not you need to have the MySQL client libraries path using the `SHLIB_PATH` environment variable. For static versions, ignore all shared-library options and run **configure** with the `--disable-shared` option.

23.1.7.13. AIX Notes:

To build the driver on AIX, make use of the following **configure** example:

```
shell> ./configure --prefix=/usr/local
             --with-unixodbc=/usr/local
             --with-mysql-path=/usr/local/mysql
             --disable-shared
             --enable-thread-safe
```

NOTE: For more information about how to build and set up the static and shared libraries across the different platforms refer to ' Using static and shared libraries across platforms [http://www.fortran-2000.com/ArnaudRecipes/sharedlib.html]'.

23.1.8. Installing MyODBC from the BitKeeper Development Source Tree

Note: You should read this section only if you are interested in helping us test our new code.

To obtain our most recent development source tree, use these instructions:

1. Download BitKeeper from http://www.bitmover.com/cgi-bin/download.cgi. You need BitKeeper 3.0 or newer to access our repository.

2. Follow the instructions to install it.

3. After BitKeeper is installed, first go to the directory you want to work from, and then use this command if you want to clone the MyODBC 3.51 branch:

   ```
   shell> bk clone bk://mysql.bkbits.net/myodbc3 myodbc-3.51
   ```

   In the preceding example, the source tree is set up in the `myodbc-3.51/` or by default `myodbc3/` subdirectory of your current directory. If you are behind the firewall and can only initiate HTTP connections, you can also use BitKeeper via HTTP. If you are required to use a proxy server, simply set the environment variable `http_proxy` to point to your proxy:

   ```
   shell> export http_proxy="http://your.proxy.server:8080/"
   ```

   Replace the `bk://` with `http://` when doing a clone. Example:

   ```
   shell> bk clone http://mysql.bkbits.net/myodbc3 myodbc-3.51
   ```

   The initial download of the source tree may take a while, depending on the speed of your connection; be patient.

4. You need GNU `autoconf 2.52` (or newer), `automake 1.4`, `libtool 1.4`, and `m4` to run the next set of commands.

   ```
   shell> cd myodbc-3.51
   shell> bk -r edit
   shell> aclocal; autoheader; autoconf;  automake;
   shell> ./configure  # Add your favorite options here
   shell> make
   ```

   For more information on how to build, refer to `INSTALL` file located in the same directory. On Windows, make use of Windows Makefiles `WIN-Makefile` and `WIN-Makefile_debug` in building the driver, for more information, see Section 23.1.6, “Installing MyODBC from a Source Distribution on Windows”.

5. When the build is done, run **make install** to install the MyODBC 3.51 driver on your system.

6. If you have gotten to the **make** stage and the distribution does not compile, please report it to `<myodbc@lists.mysql.com>`.

7. After the initial **bk clone** operation to get the source tree, you should run **bk pull** periodically to get the updates.

8. You can examine the change history for the tree with all the diffs by using **bk sccstool**. If you see some funny diffs or code that you have a question about, do not hesitate to send e-mail to

`<myodbc@lists.mysql.com>`.

Also, if you think you have a better idea on how to do something, send an e-mail to the same address with a patch. **bk diffs** produces a patch for you after you have made changes to the source. If you do not have the time to code your idea, just send a description.

9. BitKeeper has a nice help utility that you can access via **bk helptool**.

You can also browse changesets, comments and source code online by browsing to http://mysql.bkbits.net:8080/myodbc3.

23.1.9. MyODBC Configuration

This section describes how to configure MyODBC, including DSN creation and the different arguments that the driver takes as an input arguments in the connection string. It also describes how to create an ODBC trace file.

23.1.9.1. What is a Data Source Name?

A "data source" is a place where data comes from. The data source must have a persistent identifier, the Data Source Name. Using the Data Source Name, MySQL can access initialization information. With the initialization information, MySQL knows where to access the database and what settings to use when the access starts.

In effect, the data source is the *path* to the data. In different contexts this might mean different things, but typically it identifies a running MySQL server (for example via a network address or service name), plus the default database for that server at connection time, plus necessary connection information such as the port. The MySQL drivers (and, on Windows systems, the ODBC Driver Manager) use the data source for connecting. An administrative utility called the Microsoft ODBC Data Source Administrator may be useful for this purpose.

There are two places where the initialization information might be: in the Windows registry (on a Windows system), or in a DSN file (on any system).

If the information is in the Windows registry, it is called a "Machine data source". It might be a "User data source", in which case only one user can see it. Or it might be a "System data source" in which case it is accessible to all users on the computer, or indeed to all users connected to the computer, if the users are connected by Microsoft Windows NT services. When you run the ODBC Data Administration program, you have a choice whether to use "User" or "System" -- there are separate tabs.

If the information is in a DSN file, it is called a "File data source". This is a text file. Its advantages are: (a) it is an option for any kind of computer, not just a computer with a Windows operating system; (b) its contents can be transmitted or copied relatively easily.

23.1.9.2. Configuring a MyODBC DSN on Windows

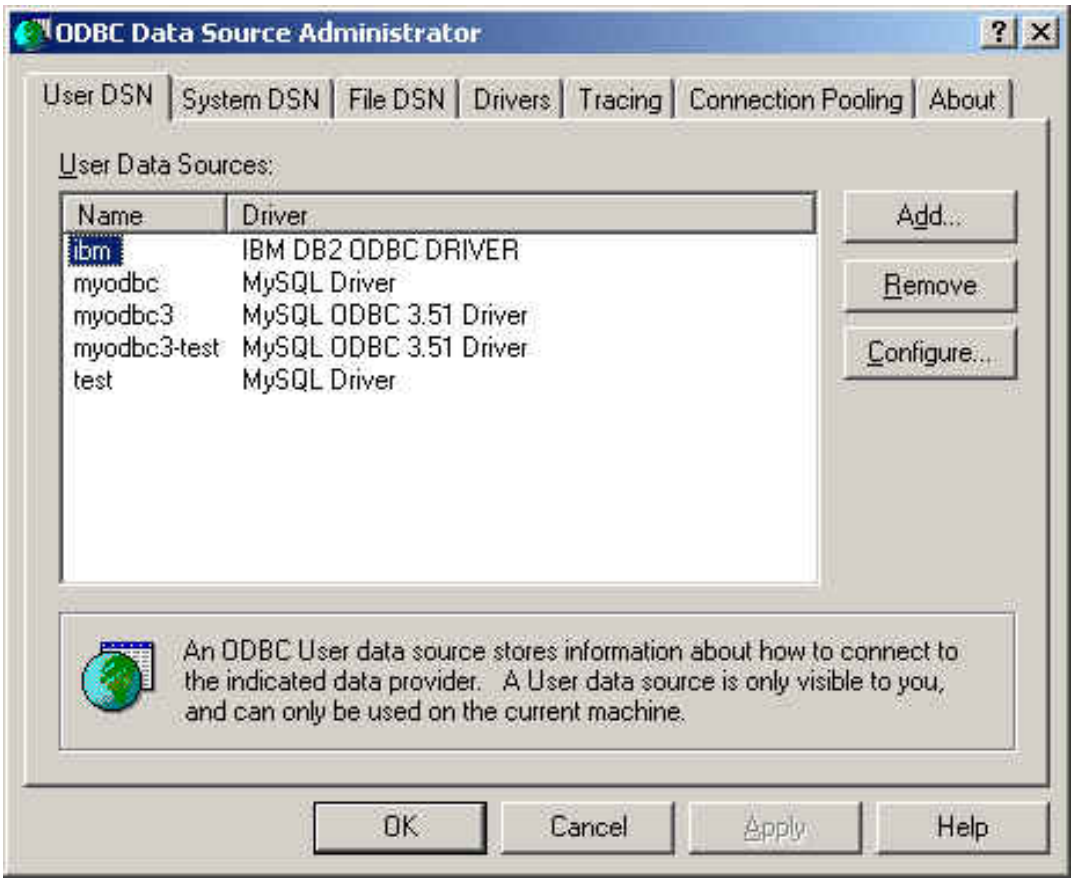
To add and configure a new MyODBC data source on Windows, use the `ODBC Data Source Administrator`. The `ODBC Administrator` updates your data source connection information. As you add data sources, the `ODBC Administrator` updates the registry information for you.

To open the `ODBC Administrator` from the Control Panel:

1. Click `Start`, point to `Settings`, and then click `Control Panel`.
2. On computers running Microsoft Windows 2000 or newer, double-click `Administrative`

`Tools`, and then double-click `Data Sources (ODBC)`. On computers running older versions of Windows, double-click `32-bit ODBC` or `ODBC`.

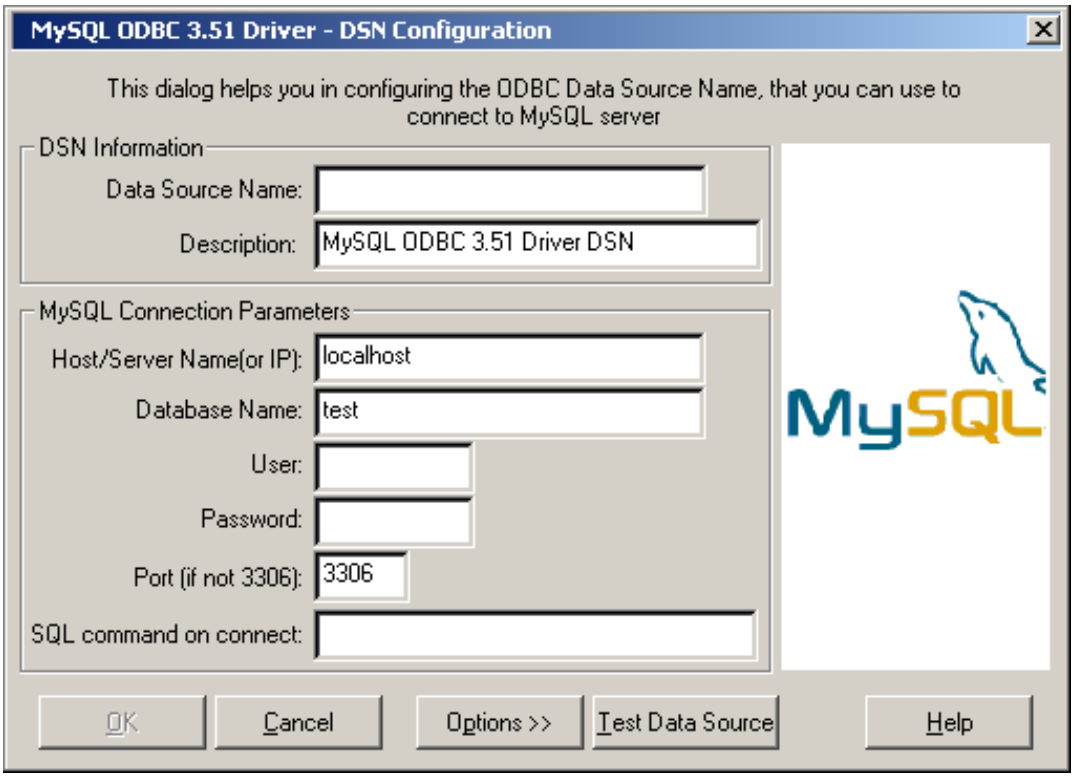
The `ODBC Data Source Administrator` dialog box appears, as shown here:



Click `Help` for detailed information about each tab of the `ODBC Data Source Administrator` dialog box.

To add a data source on Windows:

1. Open the `ODBC Data Source Administrator`.
2. In the `ODBC Data Source Administrator` dialog box, click `Add`. The `Create New Data Source` dialog box appears.
3. Select `MySQL ODBC 3.51 Driver`, and then click `Finish`. The `MySQL ODBC 3.51 Driver - DSN Configuration` dialog box appears, as shown here:



4. In the `Data Source Name` box, enter the name of the data source you want to access. It can be any valid name that you choose.

5. In the `Description` box, enter the description needed for the DSN.

6. For `Host or Server Name (or IP)` box, enter the name of the MySQL server host that you want to access. By default, it is `localhost`.

7. In the `Database Name` box, enter the name of the MySQL database that you want to use as the default database.

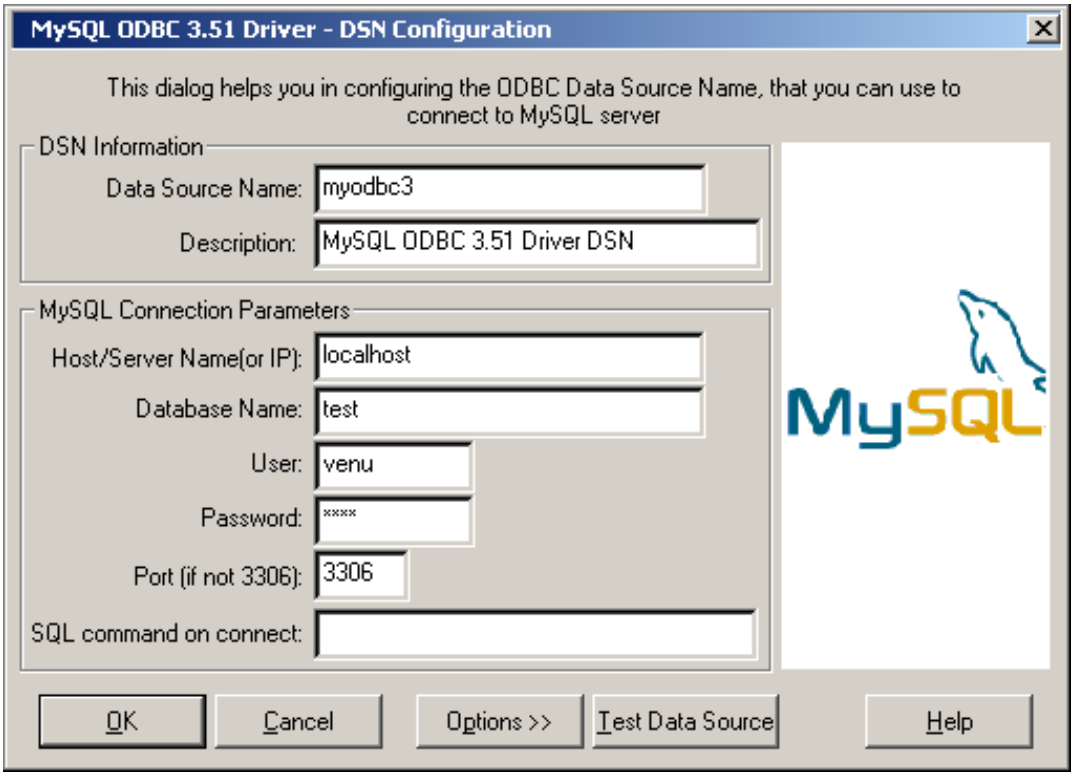
8. In the `User` box, enter your MySQL username (your database user ID).

9. In the `Password` box, enter your password.

10. In the `Port` box, enter the port number if it is not the default (3306).

11. In the `SQL Command` box, you can enter an optional SQL statement that you want to issue automatically after the connection has been established.

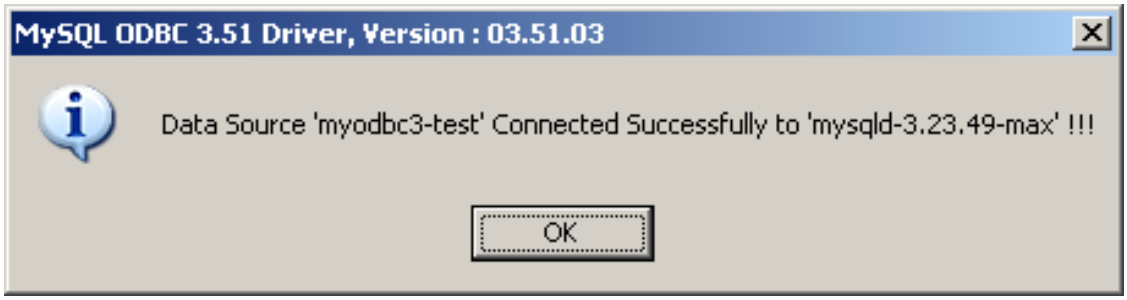
    The final dialog looks like this:



Click `OK` to add this data source.

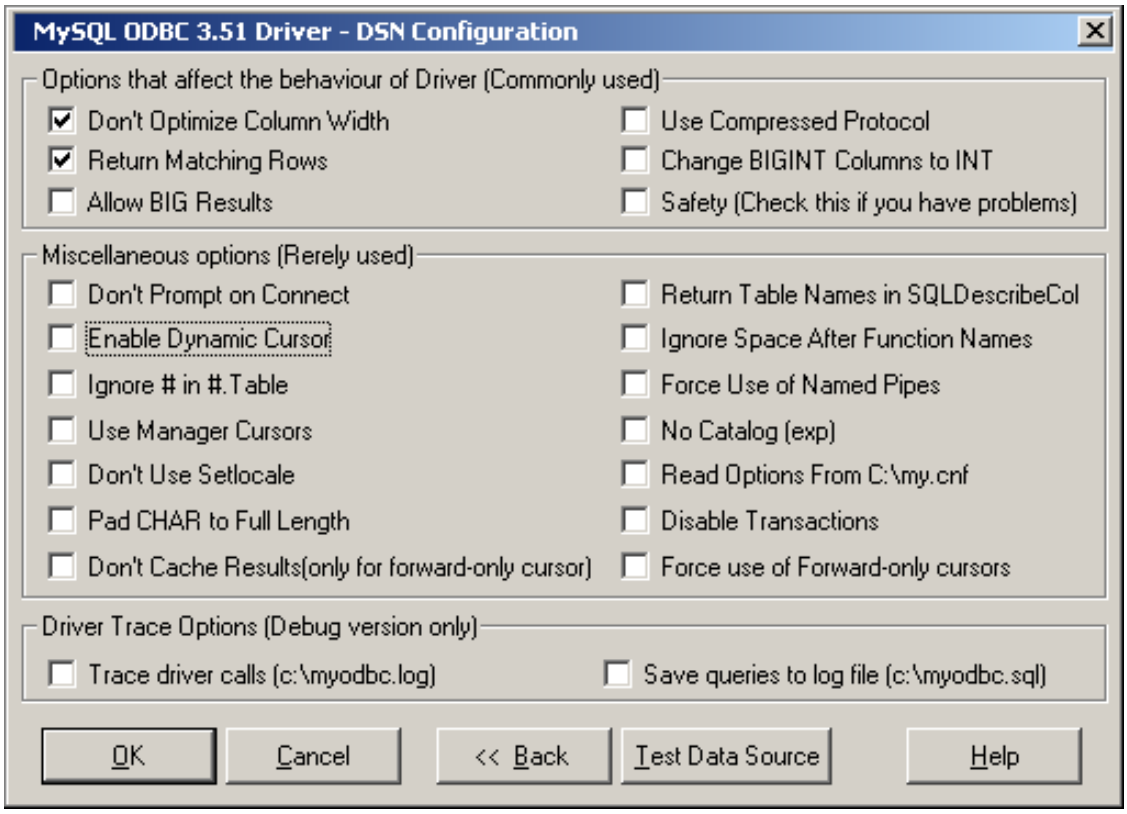
Note: Upon clicking `OK`, the `Data Sources` dialog box appears, and the `ODBC Administrator` updates the registry information. The username and connect string that you entered become the default connection values for this data source when you connect to it.

You can also test whether your settings are suitable for connecting to the server using the button `Test Data Source`. This feature is available only for the MyODBC 3.51 driver. A successful test results in the following window:



A failed test results in an error:

The DSN configuration dialog also has an `Options` button. If you select it, the following options dialog appears displaying that control driver behavior. Refer to Section 23.1.9.4, “Connection Parameters” for information about the meaning of these options.



Note: The options listed under `Driver Trace Options` are disabled (grayed out) unless you are using the debugging version of the driver DLL.

To modify a data source on Windows:

1. Open the `ODBC Data Source Administrator`. Click the appropriate DSN tab.
2. Select the MySQL data source that you want to modify and then click `Configure`. The `MySQL ODBC 3.51 Driver - DSN Configuration` dialog box appears.
3. Modify the applicable data source fields, and then click `OK`.

When you have finished modifying the information in this dialog box, the `ODBC Administrator` updates the registry information.

23.1.9.3. Configuring a MyODBC DSN on Unix

On `Unix`, you configure DSN entries directly in the `odbc.ini` file. Here is a typical `odbc.ini` file that configures `myodbc` and `myodbc3` as the DSN names for MyODBC 2.50 and MyODBC 3.51, respectively:

```
;
;  odbc.ini configuration for MyODBC and MyODBC 3.51 drivers
;

[ODBC Data Sources]
myodbc      = MyODBC 2.50 Driver DSN
myodbc3     = MyODBC 3.51 Driver DSN

[myodbc]
Driver       = /usr/local/lib/libmyodbc.so
Description  = MyODBC 2.50 Driver DSN
SERVER       = localhost
PORT         =
USER         = root
Password     =
Database     = test
OPTION       = 3
SOCKET       =

[myodbc3]
Driver       = /usr/local/lib/libmyodbc3.so
Description  = MyODBC 3.51 Driver DSN
SERVER       = localhost
PORT         =
USER         = root
Password     =
Database     = test
OPTION       = 3
SOCKET       =

[Default]
Driver       = /usr/local/lib/libmyodbc3.so
Description  = MyODBC 3.51 Driver DSN
SERVER       = localhost
PORT         =
USER         = root
Password     =
Database     = test
OPTION       = 3
SOCKET       =
```

Refer to the Section 23.1.9.4, “Connection Parameters”, for the list of connection parameters that can be supplied.

Note: If you are using unixODBC, you can use the following tools in order to set up the DSN:

- ODBCConfig GUI tool(HOWTO: ODBCConfig [http://www.unixodbc.org/config.html])
- odbcinst

In some cases when using unixODBC, you might get this error:

```
Data source name not found and no default driver specified
```

If this happens, make sure the `ODBCINI` and `ODBCSYSINI` environment variables are pointing to the right `odbc.ini` file. For example, if your `odbc.ini` file is located in `/usr/local/etc`, set the environment variables like this:

```
export ODBCINI=/usr/local/etc/odbc.ini
export ODBCSYSINI=/usr/local/etc
```

23.1.9.4. Connection Parameters

You can specify the following parameters for MyODBC in the `[Data Source Name]` section of an `ODBC.INI` file or through the `InConnectionString` argument in the `SQLDriverConnect()` call.

Parameter	Default Value	Comment
`user`	ODBC (on Windows)	The username used to connect to MySQL.
`server`	`localhost`	The hostname of the MySQL server.
`database`		The default database.
`option`	0	Options that specify how MyODBC should work. See below.
`port`	3306	The TCP/IP port to use if `server` is not `localhost`.
`stmt`		A statement to execute when connecting to MySQL.
`password`		The password for the `user` account on `server`.
`socket`		The Unix socket file or Windows named pipe to connect to if `server` is `localhost`.

The `option` argument is used to tell MyODBC that the client isn't 100% ODBC compliant. On Windows, you normally select options by toggling the checkboxes in the connection screen, but you can also select them in the `option` argument. The following options are listed in the order in which they appear in the MyODBC connect screen:

Value	Description
1	The client can't handle that MyODBC returns the real width of a column.
2	The client can't handle that MySQL returns the true value of affected rows. If this flag is set, MySQL returns ``found rows" instead. You must have MySQL 3.21.14 or newer to get this to work.
4	Make a debug log in `c:\myodbc.log`. This is the same as putting `MYSQL_DEBUG=d:t:O,c::\myodbc.log` in `AUTOEXEC.BAT`. (On Unix, the file is `/tmp/myodbc.log`.)
8	Don't set any packet limit for results and parameters.
16	Don't prompt for questions even if driver would like to prompt.
32	Enable or disable the dynamic cursor support. (Not allowed in MyODBC 2.50.)
64	Ignore use of database name in `db_name.tbl_name.col_name`.
128	Force use of ODBC manager cursors (experimental).
256	Disable the use of extended fetch (experimental).
512	Pad `CHAR` columns to full column length.
1024	`SQLDescribeCol()` returns fully qualified column names.

2048	Use the compressed client/server protocol.
4096	Tell server to ignore space after function name and before `'('` (needed by PowerBuilder). This makes all function names keywords.
8192	Connect with named pipes to a **mysqld** server running on NT.
16384	Change `LONGLONG` columns to `INT` columns (some applications can't handle `LONGLONG`).
32768	Return 'user' as `Table_qualifier` and `Table_owner` from `SQLTables` (experimental).
65536	Read parameters from the `[client]` and `[odbc]` groups from `my.cnf`.
131072	Add some extra safety checks (should not be needed but...).
262144	Disable transactions.
524288	Enable query logging to `c:\myodbc.sql`(`/tmp/myodbc.sql`) file. (Enabled only in debug mode.)
1048576	Do not cache the results locally in the driver, instead read from server (`mysql_use_result()`). This works only for forward-only cursors. This option is very important in dealing with large tables when you don't want the driver to cache the entire result set.
2097152	Force the use of `Forward-only` cursor type. In case of applications setting the default static/dynamic cursor type, and one wants the driver to use non-cache result sets, then this option ensures the forward-only cursor behavior.

To select multiple options, add together their values. For example, setting `option` to 12 (4+8) gives you debugging without packet limits.

The default `myodbc3.dll` is compiled for optimal performance. If you want to debug MyODBC 3.51 (for example, to enable tracing), you should instead use `myodbc3d.dll`. To install this file, copy `myodbc3d.dll` over the installed `myodbc3.dll` file. Make sure to revert back to the release version of the driver DLL once you are done with the debugging because the debug version may cause performance issues.

For MyODBC 2.50, `myodbc.dll` and `myodbcd.dll` are used instead.

The following table shows some recommended `option` values for various configurations:

Configuration	**Option Value**
Microsoft Access	3
Microsoft Visual Basic	3
Large tables with too many rows	2049
Driver trace generation (Debug mode)	4
Query log generation (Debug mode)	524288
Generate driver trace as well as query log (Debug mode)	524292
Large tables with no-cache results	3145731

23.1.9.5. Connecting Without a Predefined DSN

Yes. You can connect to the MySQL server using SQLDriverConnect, by specifying the `DRIVER` name field. Here are the connection strings for MyODBC using DSN-Less connection:

For MyODBC 2.50:

```
ConnectionString = "DRIVER={MySQL};\
                    SERVER=localhost;\
                    DATABASE=test;\
                    USER=venu;\
                    PASSWORD=venu;\
                    OPTION=3;"
```

For MyODBC 3.51:

```
ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};\
                    SERVER=localhost;\
                    DATABASE=test;\
                    USER=venu;\
                    PASSWORD=venu;\
                    OPTION=3;"
```

If your programming language converts backslash followed by whitespace to a space, it is preferable to specify the connection string as a single long string, or to use a concatenation of multiple strings that does not add spaces in between. For example:

```
ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};"
                    "SERVER=localhost;"
                    "DATABASE=test;"
                    "USER=venu;"
                    "PASSWORD=venu;"
                    "OPTION=3;"
```

Refer to the Section 23.1.9.4, “Connection Parameters”, for the list of connection parameters that can be supplied.

23.1.9.6. Establishing a Remote Connection to System A from System B

If you want to connect to system A from system B with a username and password of `myuser` and `mypassword`, here is a simple procedure.

On system A, follow these steps:

1. Start the MySQL server.
2. Use `GRANT` to set up an account with a username of `myuser` that can connect from system B using a password of `myuser`:

   ```
   GRANT ALL ON *.* to 'myuser'@'B' IDENTIFIED BY 'mypassword';
   ```

3. The `GRANT` statement grants all privileges to user `myuser` for connecting from system B using the password `mypassword`. To execute this statement, you should be either `root` on system A (or another user who has appropriate privileges). For more information about MySQL privileges, refer to Section 5.6, “MySQL User Account Management”.

On system B, follow these steps:

1. Configure a MyODBC DSN using the following connection parameters:

```
DSN             = remote_test
SERVER or HOST = A (or IP address of system A)
DATABASE        = test (The default database or an appropriate one)
USER            = myuser
PASSWORD        = mypassword
```

To set up a DSN-less connection, refer to Section 23.1.9.5, “Connecting Without a Predefined DSN”.

2. Check whether you are able to access system A from system B by using ping or other means. If you are not able to reach system A, check your network or Internet connections or contact your system administrator.

3. Try to connect using `DSN=remote_test`. If it fails, trace the MyODBC log, and take the further steps based on the error message from the log. If you need further assistance, send a detailed mail message to `<myodbc@lists.mysql.com>`.

You can also find a simple HOWTO at http://www.phphelp.com/tutorial/using-myodbc-to-connect-to-a-remote-database.html.

23.1.9.7. Getting an ODBC Trace File

If you encounter difficulties or problems with MyODBC, you should start by making a log file from the `ODBC Manager` (the log you get when requesting logs from `ODBC ADMIN`) and MyODBC.

To get an ODBC trace through Driver Manager, do the following:

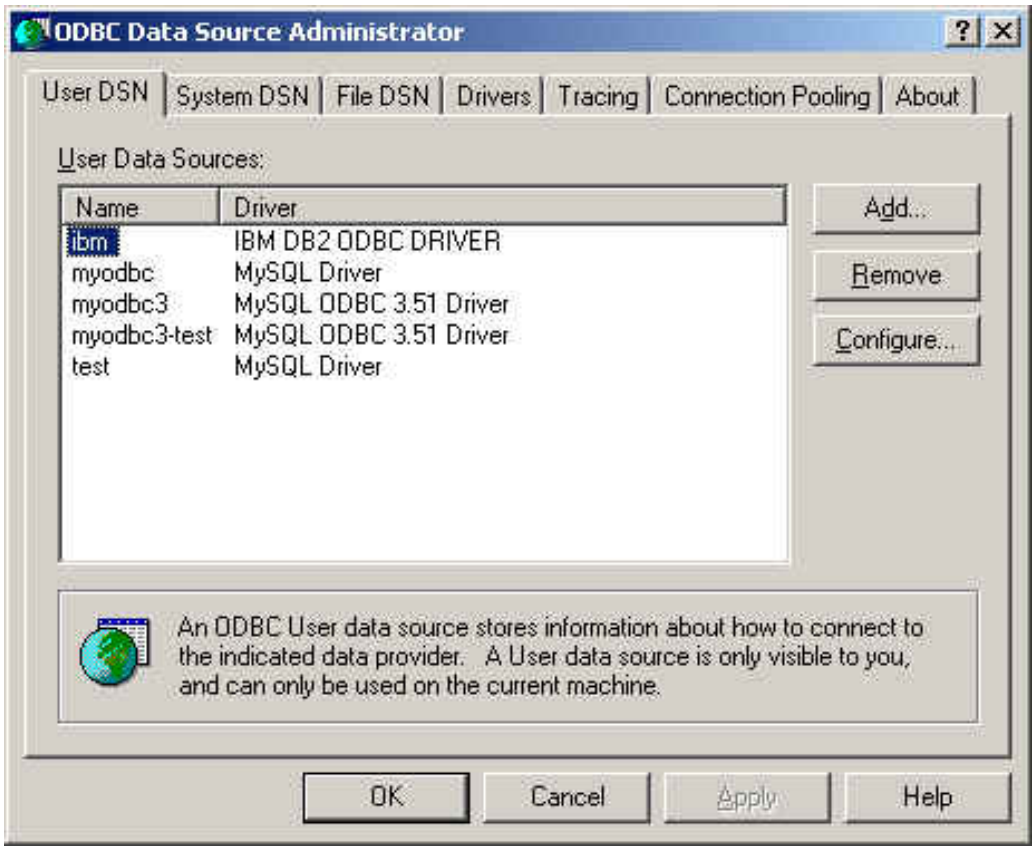
- Open ODBC Data source administrator:

  1. Click `Start`, point to `Settings`, and then click `Control Panel`.

  2. On computers running Microsoft Windows 2000, XP, or 2003, double-click `Administrative Tools`, and then double-click `Data Sources (ODBC)`, as shown below.

     On computers running an earlier version of Microsoft Windows, double-click `32-bit ODBC` or `ODBC` in the Control Panel.

  3. The `ODBC Data Source Administrator` dialog box appears, as shown below:



4. Click Help for detailed information about each tab of the ODBC Data Source Administrator dialog box.

- Enable the trace option. The procedure for this differs for Windows and Unix.

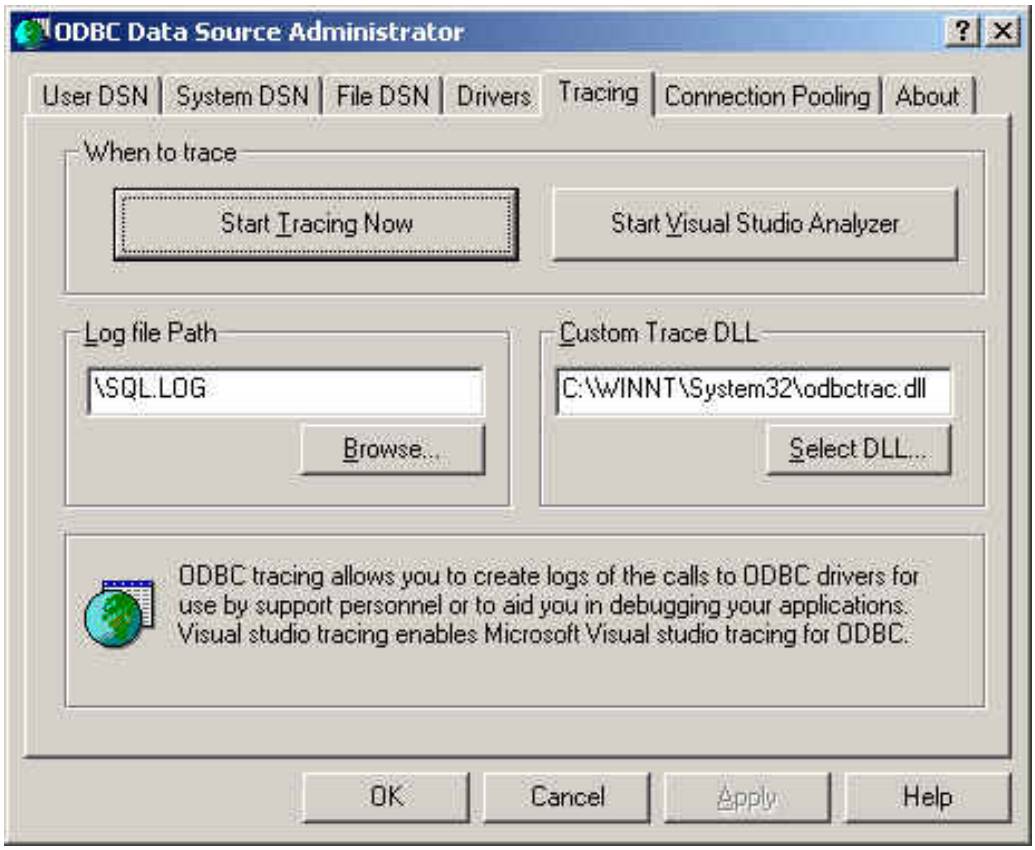
  To enable the trace option on Windows:

  1. The `Tracing` tab of the ODBC Data Source Administrator dialog box enables you to configure the way ODBC function calls are traced.

  2. When you activate tracing from the `Tracing` tab, the `Driver Manager` logs all ODBC function calls for all subsequently run applications.

  3. ODBC function calls from applications running before tracing is activated are not logged. ODBC function calls are recorded in a log file you specify.

  4. Tracing ceases only after you click `Stop Tracing Now`. Remember that while tracing is on, the log file continues to increase in size and that tracing affects the performance of all your ODBC applications.



To enable the trace option on Unix:

1. On Unix, you need to explicitly set the `Trace` option in the `ODBC.INI` file.

   Set the tracing `ON` or `OFF` by using `TraceFile` and `Trace` parameters in `odbc.ini` as shown below:

   ```
   TraceFile  = /tmp/odbc.trace
   Trace      = 1
   ```

   `TraceFile` specifies the name and full path of the trace file and `Trace` is set to `ON` or `OFF`. You can also use `1` or `YES` for `ON` and `0` or `NO` for `OFF`. If you are using **ODBCConfig** from `unixODBC`, then follow the instructions for tracing `unixODBC` calls at HOWTO-ODBCConfig [http://www.unixodbc.org/config.html].

To generate a MyODBC log, do the following:

1. Ensure that you are using the driver debug DLL (that is, `myodbc3d.dll` and not `myodbc3.dll` for MyODBC 3.51, and `myodbcd.dll` for MyODBC 2.50).

   The easiest way to do this is to get `myodbc3d.dll` (or `myodbcd.dll`) from the MyODBC 3.51 distribution and copy it over the `myodbc3.dll` (or `myodbc.dll`), which is probably in your `C:\windows\system32` or `C:\winnt\system32` directory. Note that you probably want to restore the old `myodbc.dll` file when you have finished testing, as this is a lot faster than `myodbc3d.dll` (or `myodbcd.dll`), so do keep a backup copy of original DLLs.

2. Enable the `Trace MyODBC` option flag in the MyODBC connect/configure screen. The log is

written to file `C:\myodbc.log`. If the trace option is not remembered when you are going back to the above screen, it means that you are not using the `myodbcd.dll` driver (see above). On Linux or if you are using DSN-Less connection, then you need to supply `OPTION=4` in the connection string.

3. Start your application and try to get it to fail. Then check the MyODBC trace file to find out what could be wrong.

   If you find out something is wrong, please send a mail message to `<myodbc@lists.mysql.com>` (or to `<support@mysql.com>` if you have a support contract from MySQL AB) with a brief description of the problem, with the following additional information:

   - MyODBC version
   - ODBC Driver Manager type and version
   - MySQL server version
   - ODBC trace from Driver Manager
   - MyODBC log file from MyODBC driver
   - Simple reproducible sample

   Remember that the more information you can supply to us, the more likely it is that we can fix the problem!

   Also, before posting the bug, check the MyODBC mailing list archive at http://lists.mysql.com/.

23.1.9.8. Applications Tested with MyODBC

MyODBC has been tested with the following applications:

- MS Access 95, 97, 2000, and 2002
- C++-Builder, Borland Builder 4
- Centura Team Developer (formerly Gupta SQL/Windows)
- ColdFusion (on Solaris and NT with service pack 5), How-to: MySQL and Coldfusion [http://www.njwtech.net/addons/coldfusion/mysql.html]. Troubleshooting Data Sources and Database Connectivity for UnixPlatforms [http://www.macromedia.com/v1/handlers/index.cfm?ID=11328&Method=Full&PageCall=/support/index.cfm].
- Crystal Reports
- DataJunction
- Delphi
- ERwin
- MS Excel

- iHTML
- FileMaker Pro
- FoxPro
- Notes 4.5/4.6
- MS Visio Enterprise 2000
- Vision
- Visual Objects
- Visual Interdev
- SBSS
- Perl DBD-ODBC
- Paradox
- Powerbuilder
- Powerdesigner 32-bit
- MS Visual C++
- Visual Basic
- ODBC.NET through CSharp(C#), VB and C++
- Data Architect(http://thekompany.com/products/dataarchitect/)
- SQLExpress for Xbase++(http://www.SQLExpress.net)
- Open Office (http://www.openoffice.org) How-to: MySQL + OpenOffice [http://dba.openoffice.org/proposals/MySQL_OOo.html]. How-to: OpenOffice + MyODBC + unix-ODBC [http://www.unixodbc.org/doc/OOoMySQL.pdf].
- Star Office (http://wwws.sun.com/software/star/staroffice/6.0/index.html)
- G2-ODBC bridge (http://www.gensym.com)
- Sambar Server (http://www.sambarserver.info) How-to: MyODBC + SambarServer + MySQL [http://www.sambarserver.info/article.php?sid=66].

If you know of any other applications that work with MyODBC, please send mail to `<myodbc@lists.mysql.com>` about them.

23.1.9.9. Programs Known to Work With MyODBC

Most programs should work with MyODBC, but for each of those listed here, we have tested it ourselves or received confirmation from some user that it works. Many of the descriptions provide workarounds for problems that you might encounter.

- **Program**

Comment

- Access

  To make Access work:

  - If you are using Access 2000, you should get and install the newest (version 2.6 or higher) Microsoft MDAC (`Microsoft Data Access Components`) from http://www.microsoft.com/data/. This fixes a bug in Access that when you export data to MySQL, the table and column names aren't specified. Another way to work around this bug is to upgrade to MyODBC 2.50.33 and MySQL 3.23.x, which together provide a workaround for the problem.

    You should also get and apply the Microsoft Jet 4.0 Service Pack 5 (SP5) which can be found at http://support.microsoft.com/default.aspx?scid=kb;EN-US;q239114. This fixes some cases where columns are marked as `#DELETED#` in Access.

    Note: If you are using MySQL 3.22, you must to apply the MDAC patch and use MyODBC 2.50.32 or 2.50.34 and up to work around this problem.

  - For all versions of Access, you should enable the MyODBC `Return matching rows` option. For Access 2.0, you should additionally enable the `Simulate ODBC 1.0` option.

  - You should have a timestamp in all tables that you want to be able to update. For maximum portability, don't use a length specification in the column declaration. That is, use `TIMESTAMP`, not `TIMESTAMP(`*n*`)`, *n* < 14.

  - You should have a primary key in the table. If not, new or updated rows may show up as `#DELETED#`.

  - Use only `DOUBLE` float fields. Access fails when comparing with single floats. The symptom usually is that new or updated rows may show up as `#DELETED#` or that you can't find or update rows.

  - If you are using MyODBC to link to a table that has a `BIGINT` column, the results are displayed as `#DELETED`. The work around solution is:

    - Have one more dummy column with `TIMESTAMP` as the data type.

    - Select the `Change BIGINT columns to INT` option in the connection dialog in ODBC DSN Administrator.

    - Delete the table link from Access and re-create it.

    Old records still display as `#DELETED#`, but newly added/updated records are displayed properly.

  - If you still get the error `Another user has changed your data` after adding a `TIMESTAMP` column, the following trick may help you:

    Don't use a `table` data sheet view. Instead, create a form with the fields you want, and use that `form` data sheet view. You should set the `DefaultValue` property for the `TIMESTAMP` column to `NOW()`. It may be a good idea to hide the `TIMESTAMP` column from view so your users are not confused.

  - In some cases, Access may generate illegal SQL statements that MySQL can't understand. You can fix this by selecting `"Query|SQLSpecific|Pass-Through"` from the Access menu.

  - On NT, Access reports `BLOB` columns as `OLE OBJECTS`. If you want to have `MEMO` columns

instead, you should change BLOB columns to TEXT with ALTER TABLE.

- Access can't always handle DATE columns properly. If you have a problem with these, change the columns to DATETIME.
- If you have in Access a column defined as BYTE, Access tries to export this as TINYINT instead of TINYINT UNSIGNED. This gives you problems if you have values larger than 127 in the column.

- ADO

  When you are coding with the ADO API and MyODBC, you need to pay attention to some default properties that aren't supported by the MySQL server. For example, using the CursorLocation Property as adUseServer returns a result of #1 for the RecordCount Property. To have the right value, you need to set this property to adUseClient, as shown in the VB code here:

```
Dim myconn As New ADODB.Connection
Dim myrs As New Recordset
Dim mySQL As String
Dim myrows As Long

myconn.Open "DSN=MyODBCsample"
mySQL = "SELECT * from user"
myrs.Source = mySQL
Set myrs.ActiveConnection = myconn
myrs.CursorLocation = adUseClient
myrs.Open
myrows = myrs.RecordCount

myrs.Close
myconn.Close
```

  Another workaround is to use a SELECT COUNT(*) statement for a similar query to get the correct row count.

- Active server pages (ASP)

  You should select the Return matching rows option.

- BDE applications

  To get these to work, you should select the Don't optimize column widths and Return matching rows options.

- Borland Builder 4

  When you start a query, you can use the Active property or the Open method. Note that Active starts by automatically issuing a SELECT * FROM ... query. That may not be a good thing if your tables are large.

- ColdFusion (On Unix)

  The following information is taken from the ColdFusion documentation:

  Use the following information to configure ColdFusion Server for Linux to use the unixODBC driver with MyODBC for MySQL data sources. Allaire has verified that MyODBC 2.50.26 works

with MySQL 3.22.27 and ColdFusion for Linux. (Any newer version should also work.) You can download MyODBC at http://dev.mysql.com/downloads/connector/odbc/.

ColdFusion Version 4.5.1 allows you to us the ColdFusion Administrator to add the MySQL data source. However, the driver is not included with ColdFusion Version 4.5.1. Before the MySQL driver appears in the ODBC datasources drop-down list, you must build and copy the MyODBC driver to `/opt/coldfusion/lib/libmyodbc.so`.

The Contrib directory contains the program `mydsn-xxx.zip` which allows you to build and remove the DSN registry file for the MyODBC driver on Coldfusion applications.

- DataJunction

  You have to change it to output `VARCHAR` rather than `ENUM`, as it exports the latter in a manner that causes MySQL problems.

- Excel

  Works. A few tips:

  - If you have problems with dates, try to select them as strings using the `CONCAT()` function. For example:

    ```
    SELECT CONCAT(rise_time), CONCAT(set_time)
        FROM sunrise_sunset;
    ```

    Values retrieved as strings this way should be correctly recognized as time values by Excel97.

    The purpose of `CONCAT()` in this example is to fool ODBC into thinking the column is of ``string type." Without the `CONCAT()`, ODBC knows the column is of time type, and Excel does not understand that.

    Note that this is a bug in Excel, because it automatically converts a string to a time. This would be great if the source was a text file, but is unfortunate when the source is an ODBC connection that reports exact types for each column.

- Word

  To retrieve data from MySQL to Word/Excel documents, you need to use the MyODBC driver and the Add-in Microsoft Query help.

  For example, create a database with a table containing two columns of text:

  - Insert rows using the **mysql** client command-line tool.
  - Create a DSN file using the ODBC manager, for example, `my` for the database that was just created.
  - Open the Word application.
  - Create a blank new document.
  - In the `Database` tool bar, press the `Insert Database` button.
  - Press the `Get Data` button.

- At the right hand of the `Get Data` screen, press the `Ms Query` button.
- In `Ms Query`, create a new data source using the `my` DSN file.
- Select the new query.
- Select the columns that you want.
- Make a filter if you want.
- Make a Sort if you want.
- Select `Return Data to Microsoft Word`.
- Click `Finish`.
- Click `Insert Data` and select the records.
- Click `OK` and you see the rows in your Word document.

- odbcadmin

  Test program for ODBC.

- Delphi

  You must use BDE Version 3.2 or newer. Select the `Don't optimize column width` option when connecting to MySQL.

  Also, here is some potentially useful Delphi code that sets up both an ODBC entry and a BDE entry for MyODBC. The BDE entry requires a BDE Alias Editor that is free at a Delphi Super Page near you. (Thanks to Bryan Brunton `<bryan@flesherfab.com>` for this):

```
fReg:= TRegistry.Create;
  fReg.OpenKey('\Software\ODBC\ODBC.INI\DocumentsFab', True);
  fReg.WriteString('Database', 'Documents');
  fReg.WriteString('Description', ' ');
  fReg.WriteString('Driver', 'C:\WINNT\System32\myodbc.dll');
  fReg.WriteString('Flag', '1');
  fReg.WriteString('Password', '');
  fReg.WriteString('Port', ' ');
  fReg.WriteString('Server', 'xmark');
  fReg.WriteString('User', 'winuser');
  fReg.OpenKey('\Software\ODBC\ODBC.INI\ODBC Data Sources', True);
  fReg.WriteString('DocumentsFab', 'MySQL');
  fReg.CloseKey;
  fReg.Free;

  Memo1.Lines.Add('DATABASE NAME=');
  Memo1.Lines.Add('USER NAME=');
  Memo1.Lines.Add('ODBC DSN=DocumentsFab');
  Memo1.Lines.Add('OPEN MODE=READ/WRITE');
  Memo1.Lines.Add('BATCH COUNT=200');
  Memo1.Lines.Add('LANGDRIVER=');
  Memo1.Lines.Add('MAX ROWS=-1');
  Memo1.Lines.Add('SCHEMA CACHE DIR=');
  Memo1.Lines.Add('SCHEMA CACHE SIZE=8');
  Memo1.Lines.Add('SCHEMA CACHE TIME=-1');
```

```
Memo1.Lines.Add('SQLPASSTHRU MODE=SHARED AUTOCOMMIT');
Memo1.Lines.Add('SQLQRYMODE=');
Memo1.Lines.Add('ENABLE SCHEMA CACHE=FALSE');
Memo1.Lines.Add('ENABLE BCD=FALSE');
Memo1.Lines.Add('ROWSET SIZE=20');
Memo1.Lines.Add('BLOBS TO CACHE=64');
Memo1.Lines.Add('BLOB SIZE=32');

AliasEditor.Add('DocumentsFab','MySQL',Memo1.Lines);
```

- C++ Builder

  Tested with BDE Version 3.0. The only known problem is that when the table schema changes, query fields are not updated. BDE, however, does not seem to recognize primary keys, only the index named `PRIMARY`, though this has not been a problem.

- Vision

  You should select the `Return matching rows` option.

- Visual Basic

  To be able to update a table, you must define a primary key for the table.

  Visual Basic with ADO can't handle big integers. This means that some queries like `SHOW PROCESSLIST` do not work properly. The fix is to use `OPTION=16384` in the ODBC connect string or to select the `Change BIGINT columns to INT` option in the MyODBC connect screen. You may also want to select the `Return matching rows` option.

- VisualInterDev

  If you have a `BIGINT` in your result, you may get the error `[Microsoft][ODBC Driver Manager] Driver does not support this parameter` Try selecting the `Change BIGINT columns to INT` option in the MyODBC connect screen.

- Visual Objects

  You should select the `Don't optimize column widths` option.

- MS Visio Enterprise 2000

  We made database model diagram by connecting from MS Vision Enterprise 2000 to MySQL via MyODBC (2.50.37 or greater) and using Visio's reverse engineer function to retrieve information about the DB (Visio shows all the column definitions, primary keys, Indexes and so on). Also we tested by designing new tables in Visio and exported them to MySQL via MyODBC.

23.1.10. MyODBC Connection-Related Issues

This section answers MyODBC connection-related questions.

23.1.10.1. While Configuring a MyODBC DSN, a `Could Not Load Translator or Setup Library` Error Occurs

For more information, refer to MS KnowledgeBase Article(Q260558)

[http://support.microsoft.com/default.aspx?scid=kb;EN-US;q260558]. Also, make sure you have the latest valid `ctl3d32.dll` in your system directory.

23.1.10.2. While Connecting, an `Access denied` Error Occurs

Refer to Section 5.5.8, “Causes of `Access denied` Errors”.

23.1.10.3. INFO: About ODBC Connection Pooling

Refer to this document about connection pooling: http://support.microsoft.com/default.aspx?scid=kb;EN-US;q169470.

23.1.11. MyODBC and Microsoft Access

This section of the document answers questions related to MyODBC with Microsoft Access.

23.1.11.1. How to Set Up Microsoft Access to Work with MySQL using MyODBC?

The following must be done on your client PC in order to make Microsoft Access work with MyODBC.

1. If you are using Access 2000, you should get and install the newest (version 2.6 or higher) Microsoft MDAC (`Microsoft Data Access Components`) from http://www.microsoft.com/data/. This fixes a bug in Access that when you export data to MySQL, the table and column names aren't specified. Another way to work around this bug is to upgrade to MyODBC 2.50.33 and MySQL 3.23.x, which together provide a workaround for the problem.

   You should also get and apply the Microsoft Jet 4.0 Service Pack 5 (SP5) which can be found at http://support.microsoft.com/default.aspx?scid=kb;EN-US;q239114. This fixes some cases where columns are marked as `#DELETED#` in Access.

   Note: If you are using MySQL 3.22, you must to apply the MDAC patch and use MyODBC 2.50.32 or 2.50.34 and up to work around this problem.

2. Install the latest version of MySQL from http://dev.mysql.com/downloads/.

3. Install the latest version of MyODBC 3.51 or 2.50 from http://dev.mysql.com/downloads/connector/odbc/.

4. For all Access versions, you should enable the `Return matching rows` option.

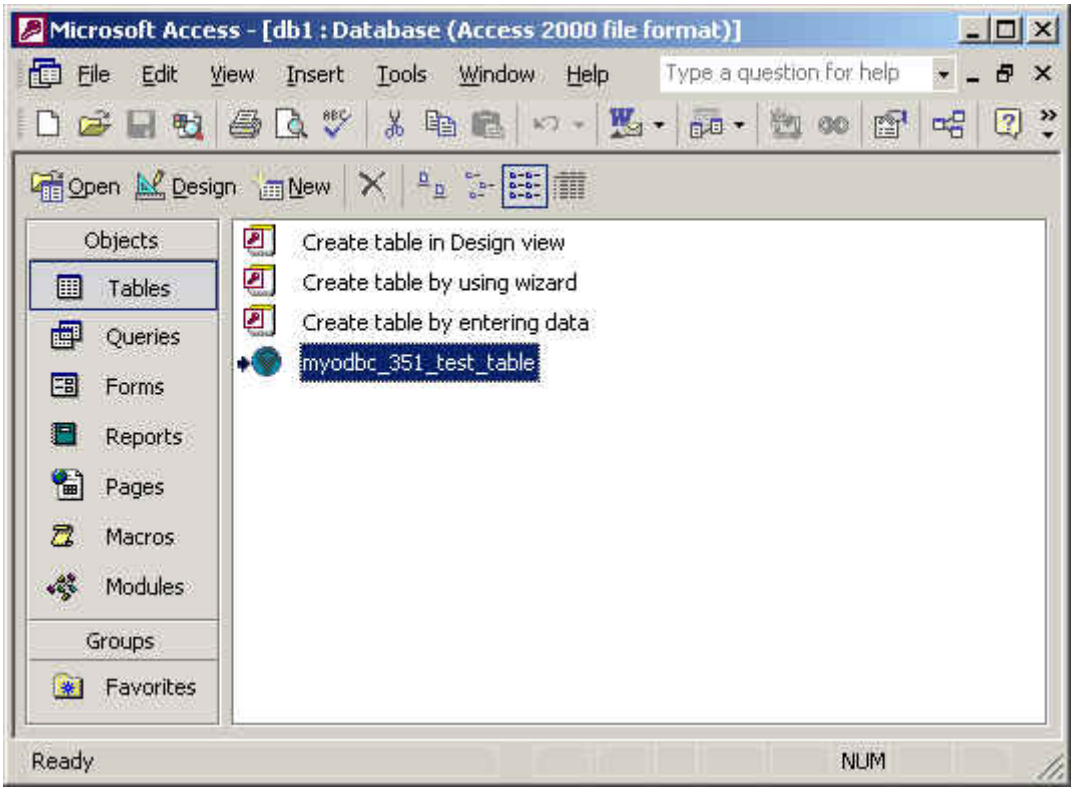
5. Start working with Access as the front-end for MySQL Server through MyODBC.

23.1.11.2. How to Export a Table or Query from Access to MySQL?

You cannot export a table or query to MySQL unless you have installed MyODBC.

To export a table from Access to MySQL, follow these instructions:

1. When you open an Access database or an Access project, a Database window appears. It displays shortcuts for creating new database objects and opening existing objects.



2. Click the name of the `table` or `query` you want to export, and then in the `File` menu, select `Export`.

3. In the `Export Object Type` *`Object name`* `To` dialog box, in the `Save As Type` box, select `ODBC Databases ()` as shown here:

4. In the `Export` dialog box, enter a name for the file (or use the suggested name), and then select `OK`.

5. The Select Data Source dialog box is displayed; it lists the defined data sources for any ODBC drivers installed on your computer. Click either the File Data Source or Machine Data Source tab, and then double-click the MyODBC or MyODBC 3.51 data source that you want to export to. To define a new data source for MyODBC, please Section 23.1.9.2, “Configuring a MyODBC DSN on Windows”.

Microsoft Access connects to the MySQL Server through this data source and exports new tables and or data.

23.1.11.3. How to Import or Link MySQL Database Tables to Access?

You cannot export a table or query to MySQL database unless you have installed the MyODBC.

To import or link a table(s) from MySQL to Access, follow the instructions:

1. Open a database, or switch to the Database window for the open database.
2. To import tables, on the `File` menu, point to `Get External Data`, and then click `Import`. To link tables, on the File menu, point to `Get External Data`, and then click `Link Tables`.
3. In the `Import` (or `Link`) dialog box, in the Files Of Type box, select `ODBC Databases ()`. The Select Data Source dialog box lists the defined data sources The Select Data Source dialog box is displayed; it lists the defined data sources for any ODBC drivers installed on your computer. Click either the File Data Source or Machine Data Source tab, and then double-click the MyODBC or MyODBC 3.51 data source that you want to export to. To define a new data source for the MyODBC or MyODBC 3.51 driver, please Section 23.1.9.2, “Configuring a MyODBC DSN on Windows”.
4. If the ODBC data source that you selected requires you to log on, enter your login ID and password (additional information might also be required), and then click `OK`.
5. Microsoft Access connects to the MySQL server through `ODBC data source` and displays the list of tables that you can `import` or `link`.
6. Click each table that you want to `import` or `link`, and then click `OK`. If you're linking a table and it doesn't have an index that uniquely identifies each record, then Microsoft Access displays a list of the fields in the linked table. Click a field or a combination of fields that uniquely identifies each record, and then click `OK`.

23.1.11.4. The Structure or Location of a Linked Table has been Changed. Can I See Those Changes Locally in Linked Tables?

Yes. Use the following procedure to view or to refresh links when the structure or location of a linked table has changed. The Linked Table Manager lists the paths to all currently linked tables.

To wiew or refresh links:

1. Open the database that contains links to tables.
2. On the `Tools` menu, point to `Add-ins`, and then click `Linked Table Manager`.
3. Select the check box for the tables whose links you want to refresh.
4. Click OK to refresh the links.

Microsoft Access confirms a successful refresh or, if the table wasn't found, displays the `Select New Location of` <table name> dialog box in which you can specify its the table's new location.If several selected tables have moved to the new location that you specify, the Linked Table Manager searches that location for all selected tables, and updates all links in one step.

To change the path for a set of linked tables:

1. Open the database that contains links to tables.
2. On the `Tools` menu, point to `Add-ins`, and then click `Linked Table Manager`.
3. Select the `Always Prompt For A New Location` check box.
4. Select the check box for the tables whose links you want to change, and then click `OK`.
5. In the `Select New Location of` <table name> dialog box, specify the new location, click `Open`, and then click `OK`.

23.1.11.5. When I Insert or Update a Record in Linked Tables, I Get #DELETED#

If the inserted or updated records are shown as `#DELETED#` in the access, then:

- If you are using Access 2000, you should get and install the newest (version 2.6 or higher) Microsoft MDAC (`Microsoft Data Access Components`) from http://www.microsoft.com/data/. This fixes a bug in Access that when you export data to MySQL, the table and column names aren't specified. Another way to work around this bug is to upgrade to MyODBC 2.50.33 and MySQL 3.23.x, which together provide a workaround for the problem.

  You should also get and apply the Microsoft Jet 4.0 Service Pack 5 (SP5) which can be found at http://support.microsoft.com/default.aspx?scid=kb;EN-US;q239114. This fixes some cases where columns are marked as `#DELETED#` in Access.

  Note: If you are using MySQL 3.22, you must to apply the MDAC patch and use MyODBC 2.50.32 or 2.50.34 and up to work around this problem.

- For all versions of Access, you should enable the MyODBC `Return matching rows` option. For Access 2.0, you should additionally enable the `Simulate ODBC 1.0` option.
- You should have a timestamp in all tables that you want to be able to update. For maximum portability, don't use a length specification in the column declaration. That is, use `TIMESTAMP`, not `TIMESTAMP(`*n*`)`, *n* < 14.
- You should have a primary key in the table. If not, new or updated rows may show up as `#DELETED#`.
- Use only `DOUBLE` float fields. Access fails when comparing with single floats. The symptom usually is that new or updated rows may show up as `#DELETED#` or that you can't find or update rows.
- If you are using MyODBC to link to a table that has a `BIGINT` column, the results are displayed as `#DELETED`. The work around solution is:
  - Have one more dummy column with `TIMESTAMP` as the data type.
  - Select the `Change BIGINT columns to INT` option in the connection dialog in ODBC DSN Administrator.
  - Delete the table link from Access and re-create it.

  Old records still display as `#DELETED#`, but newly added/updated records are displayed properly.

23.1.11.6. How Do I Handle Write Conflicts or Row Location Errors?

If you see the following errors, select the `Return Matching Rows` option in the DSN configuration dialog, or specify `OPTION=2`, as the connection parameter:

```
Write Conflict. Another user has changed your data.
```

```
Row cannot be located for updating. Some values may have been changed
since it was last read.
```

23.1.11.7. Whenever I Export a Table from Access 97, a Strange Syntax Error Occurs

This is a strange issue from Access 97, and doesn't appear with Access 2000 or 2002. You can overcome this by upgrading the MyODBC driver to at least MyODBC 3.51.02.

23.1.11.8. Access Returns `Another user has modified the record that you have modified` While Editing Records

With some programs, this error may occur: `Another user has modified the record that you have modified`. In most cases, this can be solved by doing one of the following things:

- Add a primary key for the table if one doesn't exist.
- Add a timestamp column if one doesn't exist.
- Only use double float fields. Some programs may fail when they compare single floats.

If these strategies don't help, you should start by making a log file from the ODBC manager (the log you get when requesting logs from ODBCADMIN) and a MyODBC log to help you figure out why things go wrong. For instructions, see Section 23.1.9.7, “Getting an ODBC Trace File”.

23.1.11.9. How to Trap ODBC Login Error Messages in Access?

Read ``How to Trap ODBC Login Error Messages in Access'' at http://support.microsoft.com/support/kb/articles/Q124/9/01.asp?LN=EN-US&SD=gn&FR=0%3CP%3E.

23.1.11.10. How Do I Optimize Access for Performance with MyODBC?

- Optimizing for Client/Server Performance [http://support.microsoft.com/default.aspx?scid=kb;en-us;128808]
- Tips for Converting Applications to Using ODBCDirect [http://support.microsoft.com/default.aspx?scid=kb;en-us;164481]
- Tips for Optimizing Queries on Attached SQL Tables [http://support.microsoft.com/default.aspx?scid=kb;EN-US;q99321]

23.1.11.11. I Have Very Long Tables. What is the Best Configuration for MyODBC to Access These Tables?

If you have very large (long) tables in Access, it might take a very long time to open them. Or you might

run low on virtual memory and eventually get an `ODBC Query Failed` error and the table cannot open. To deal with this, select the following options:

- Return Matching Rows (2)
- Allow BIG Results (8).

These add up to a value of 10 (`OPTION=10`).

23.1.11.12. How to Set the QueryTimeout Value for ODBC Connections?

Read ``Set the QueryTimeout Value for ODBC Connections'' at http://support.microsoft.com/default.aspx?scid=kb%3Ben-us%3B153756.

23.1.11.13. INFO: Tools to Export/Import from/to Access to/from MySQL

Refer to converters [http://www.mysql.com/portal/software/convertors/] section for list of available tools.

23.1.12. MyODBC and Microsoft VBA and ASP

This section answers questions related to using MyODBC with Microsoft Visual Basic(ADO, DAO & RDO) and ASP.

23.1.12.1. Why Does `SELECT COUNT(*) FROM` *`tbl_name`* Return an Error?

It's because the `COUNT(*)` expression is returning a `BIGINT`, and ADO can't make sense of a number this big. Select the `Change BIGINT columns to INT` option (option value 16384).

23.1.12.2. Whenever I Use the `AppendChunk()` or `GetChunk()` ADO Methods, I Get an Error `Multiple-step operation generated errors. Check each status value.`

The `GetChunk()` and `AppendChunk()` methods from ADO doesn't work as expected when the cursor location is specified as adUseServer. On the other hand, you can overcome this error by using adUseClient.

A simple example can be found from, http://www.dwam.net/iishelp/ado/docs/adomth02_4.htm

23.1.12.3. How to Find the Total Number of Rows Affected by a Particular SQL Statement in ADO?

You can make use of `RecordsAffected` property in the ADO execute method. For more information on the usage of execute method, refer to http://msdn.microsoft.com/library/default.asp?url=/library/en-us/ado270/htm/mdmthcnnexecute.asp.

23.1.12.4. How Do I Handle Blob Data in Visual Basic?

Here is an excellent article from Mike Hillyer (<m.hillyer@telusplanet.net>); explaining how to insert and/or fetch data from blob columns through MyODBC from ADO: MySQL BLOB columns and Visual Basic 6 [http://www.dynamergy.com/mike/articles/blobaccessvb.html].

23.1.12.5. How Do I Map Visual Basic Data Types to MySQL Types?

Here is yet another good article from Mike Hillyer (<m.hillyer@telusplanet.net>): How to map Visual basic data type to MySQL types [http://www.dynamergy.com/mike/articles/vbmysqldatatypes.php].

23.1.12.6. SAMPLES: VB with ADO, DAO and RDO

A simple examples for the usage of ADO, DAO and RDO with VB can be found her:

- ADO sample: Section 23.1.19, “MyODBC With VB: ADO, DAO and RDO”
- DAO sample: Section 23.1.19, “MyODBC With VB: ADO, DAO and RDO”
- RDO sample: Section 23.1.19, “MyODBC With VB: ADO, DAO and RDO”

If you find any other good example or HOW-TO on ADO/DAO/RDO, then please send the details to <myodbc@lists.mysql.com>

23.1.12.7. ASP and MySQL with MyODBC

For more information about how to access MySQL via ASP using MyODBC, refer to the following articles:

- Using MyODBC To Access Your MySQL Database Via ASP [http://www.devarticles.com/c/a/ASP/Using-MyODBC-To-Access-Your-MySQL-Database-Via-ASP/]
- ASP and MySQL at DWAM.NT [http://www.dwam.net/mysql/asp_myodbc.asp]

A Frequently Asked Questions list for ASP can be found at http://support.microsoft.com/default.aspx?scid=/Support/ActiveServer/faq/data/adofaq.asp.

23.1.12.8. INFO: Frequently Asked Questions on ActiveX Data Objects (ADO)

For information, see ActiveX Data Objects(ADO) Frequently Asked Questions [http://support.microsoft.com/default.aspx?scid=kb;EN-US;q183606].

23.1.13. MyODBC and Third-Party ODBC Tools

This section answers questions related to MyODBC with various ODBC-related tools; such as Microsoft Word, Excel and ColdFusion.

23.1.13.1. How to Retrieve Data from MySQL into MS-Word/Excel Documents?

To retrieve data from MySQL to Word/Excel documents, you need to use the MyODBC driver and the Add-in Microsoft Query help.

For example, create a database with a table containing two columns of text:

- Insert rows using the **mysql** client command-line tool.
- Create a DSN file using the ODBC manager, for example, `my` for the database that was just created.
- Open the Word application.
- Create a blank new document.
- In the `Database` tool bar, press the `Insert Database` button.
- Press the `Get Data` button.
- At the right hand of the `Get Data` screen, press the `Ms Query` button.
- In `Ms Query`, create a new data source using the `my` DSN file.
- Select the new query.
- Select the columns that you want.
- Make a filter if you want.
- Make a Sort if you want.
- Select `Return Data to Microsoft Word`.
- Click `Finish`.
- Click `Insert Data` and select the records.
- Click `OK` and you see the rows in your Word document.

23.1.13.2. Exporting Tables from MS DTS to MySQL Using MyODBC Results in a Syntax Error

This is an issue similar to that of Access 97 when your table consists of `TEXT` or `VARCHAR` data types. You can fix this error by upgrading your MyODBC driver to version 3.51.02 or higher.

23.1.13.3. HOWTO: Configure MySQL+MyODBC+unixODBC+ColdFusion on Solaris

Refer to MySQL ColdFusion unixODBC MyODBC and Solaris - how to succeed [http://dbforums.com/showthread.php?threadid=174934]

23.1.14. MyODBC General Functionality

This section of the document answers questions related to MyODBC general functionality.

23.1.14.1. How to Get the Value of an `AUTO_INCREMENT` Column in

ODBC

A common problem is how to get the value of an automatically generated ID from an `INSERT` statement. With ODBC, you can do something like this (assuming that `auto` is an `AUTO_INCREMENT` field):

```
INSERT INTO tbl (auto,text) VALUES(NULL,'text');
SELECT LAST_INSERT_ID();
```

Or, if you are just going to insert the ID into another table, you can do this:

```
INSERT INTO tbl (auto,text) VALUES(NULL,'text');
INSERT INTO tbl2 (id,text) VALUES(LAST_INSERT_ID(),'text');
```

See Section 22.2.13.3, “How to Get the Unique ID for the Last Inserted Row”.

For the benefit of some ODBC applications (at least Delphi and Access), the following query can be used to find a newly inserted row:

```
SELECT * FROM tbl WHERE auto IS NULL;
```

23.1.14.2. Does MyODBC Support Dynamic Cursor Type?

Yes. MyODBC 3.51 supports `Dynamic cursor` type along with `Forward-only` and `static`.

Due to the performance issues, the driver does not support this feature by default. You can enable this by specifying the connection option flag as `OPTION=32` or by checking the `Enable Dynamic Cursor` option from the DSN configuration.

23.1.14.3. What Causes `Transactions are not enabled` Errors?

The driver returns this error when an application issues any transactional call but the underlying MySQL server either does not support transactions or they are not enabled.

To avoid this problem, you must use a server that has either or both of the `InnoDB` or `BDB` storage engines enabled, and use tables of those types. MySQL servers from version 4.0 and up support `InnoDB` by default. MySQL-Max servers also support `BDB` on platforms where `BDB` is available.

Also, if your server supports transactional table types (`InnoDB` and `BDB`) make sure the `disable transactions` option is not set from the DSN configuration.

23.1.14.4. What Causes `Cursor not found` Errors?

This is because the application is using old MyODBC 2.50 version, and it did not set the cursor name explicitly through SQLSetCursorName. The fix is to upgrade to MyODBC 3.51 version.

23.1.14.5. Can I Use MyODBC 2.50 Applications with MyODBC 3.51?

Yes. If you find something is not working with MyODBC 3.51 that works with MyODBC 2.50, then send a mail message to `<myodbc@lists.mysql.com>`

23.1.14.6. Can I Access MySQL from .NET Environment Using MyODBC?

Yes. You can make use of **odbc.net** to connect to MySQL through MyODBC. Here are the few basic samples to connect to MySQL from VC.NET and VB.NET.

- See Section 23.1.20.1, “ODBC.NET: CSHARP(C#)”
- See Section 23.1.20.2, “ODBC.NET: VB”

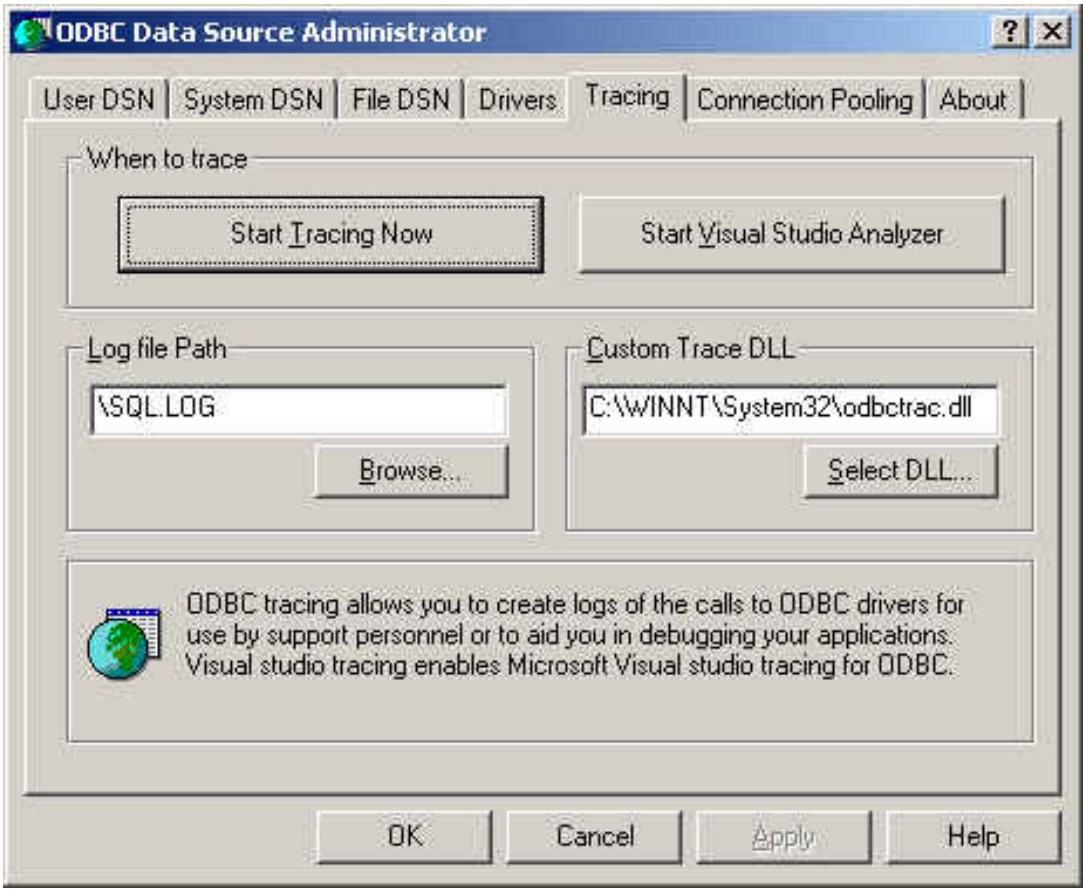
Here is yet another excellent article "Exploring MySQL on .NET environment [http://www.mysql.com/newsletter/2002-07/a0000000035.html]" by **Venu** (MyODBC developer) that covers about all MySQL .NET interfaces along with some useful examples.

Caution: Using ODBC.NET with MyODBC, while fetching empty string (0 length), it starts giving the SQL_NO_DATA exception. You can get the patch for this from http://support.microsoft.com/default.aspx?scid=kb;EN-US;q319243.

23.1.14.7. Why Does MyODBC Perform Poorly, and Also Make a Lot of Disk Activity for Relatively Small Queries?

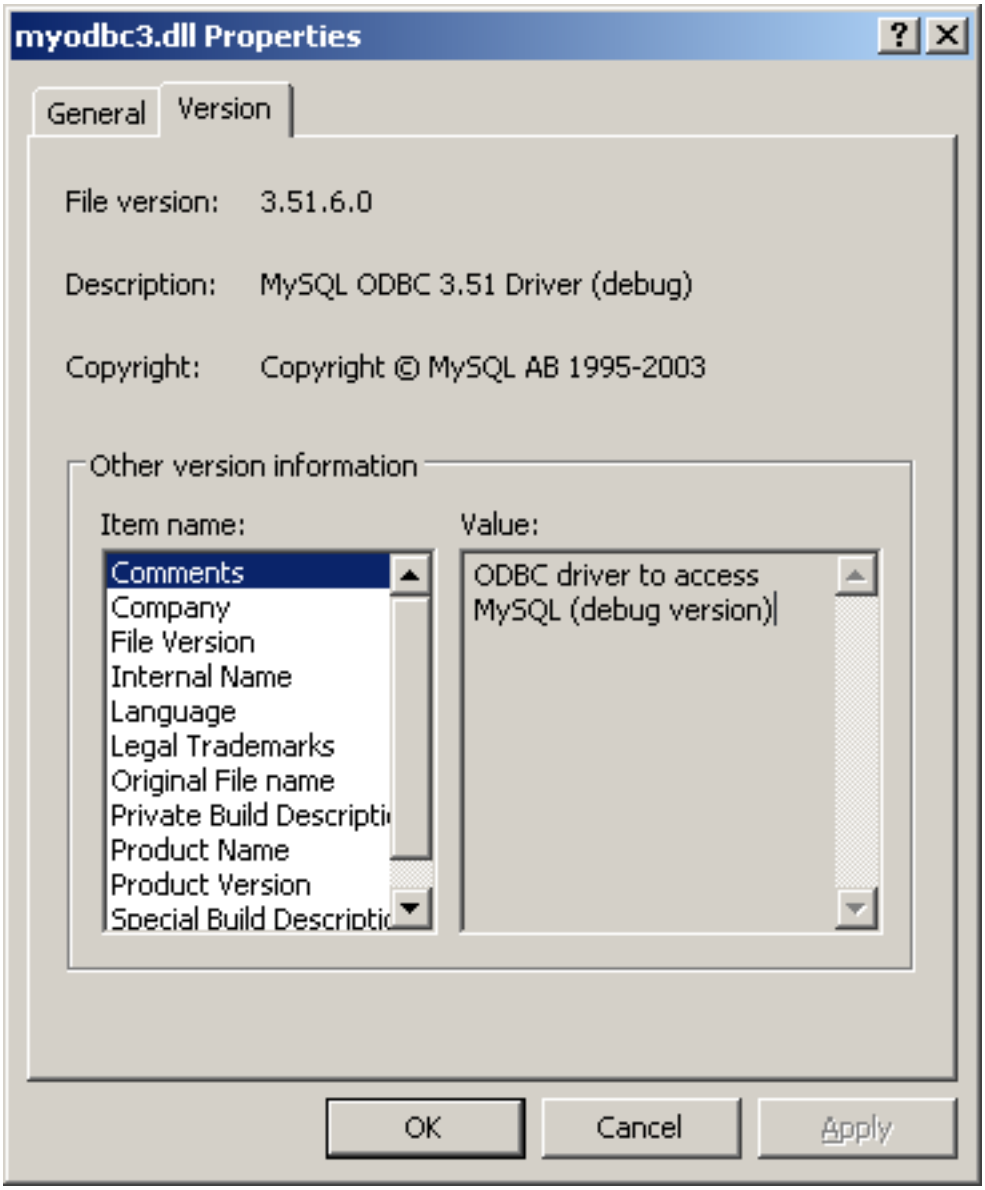
MyODBC is a lot faster than any other ODBC driver. Slowness might be due to not using the following options.

- The **ODBC Tracing** option is turned on. You can cross-check whether this option is not turned on by following the instructions from here [http://support.microsoft.com/default.aspx?scid=kb;EN-US;268591].

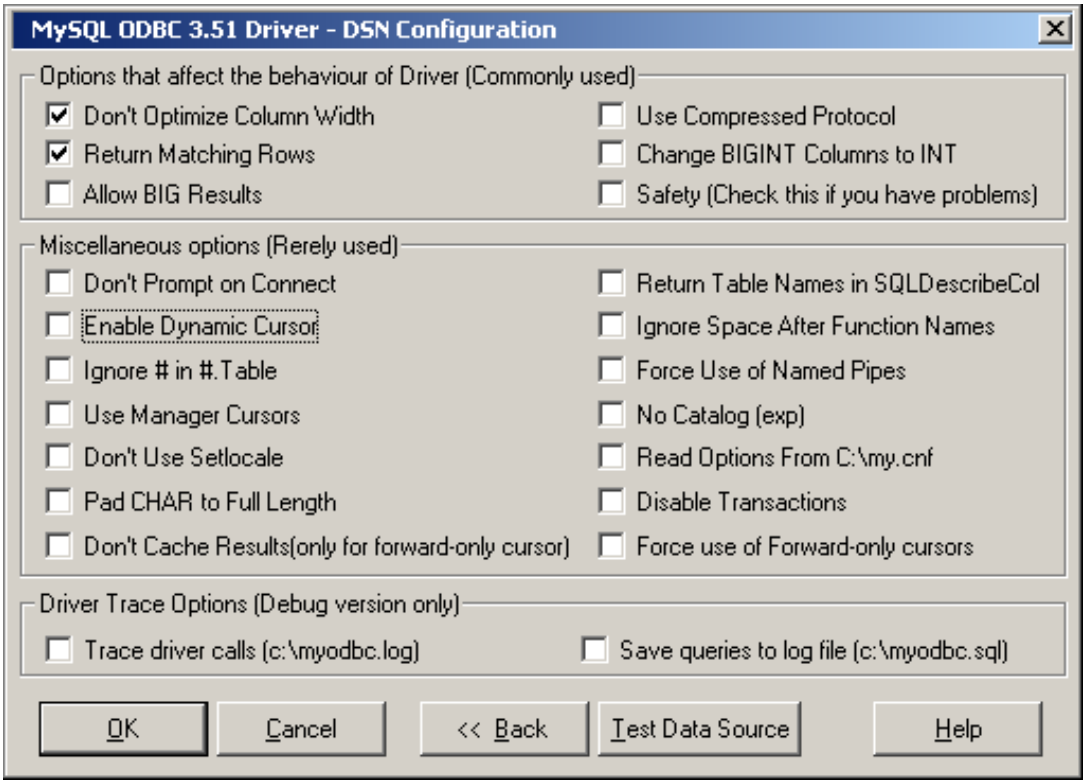


As shown in the above image, the 'When to trace' option from the ODBC Data Source Administrator 'Tracing' tab should always point to 'Start Tracing Now', instead of 'Stop Tracing Now'.

- The **Debug version** of the driver is used. If you are using the debug version of the driver DLL, it can also relatively slow down the query processing time. You can cross-check whether you are using the debug or release version of the DLL from the 'Comments' section of the driver DLL properties (from the system directory, right click on the driver DLL and click on properties) as shown below:



- The **Driver trace and query logs** are enabled. Even if you intent to use the debug version of the driver (you should always use the release version in the production environment), make sure the driver trace and query log options(OPTION=4,524288 respectively) are not enabled as shown below:

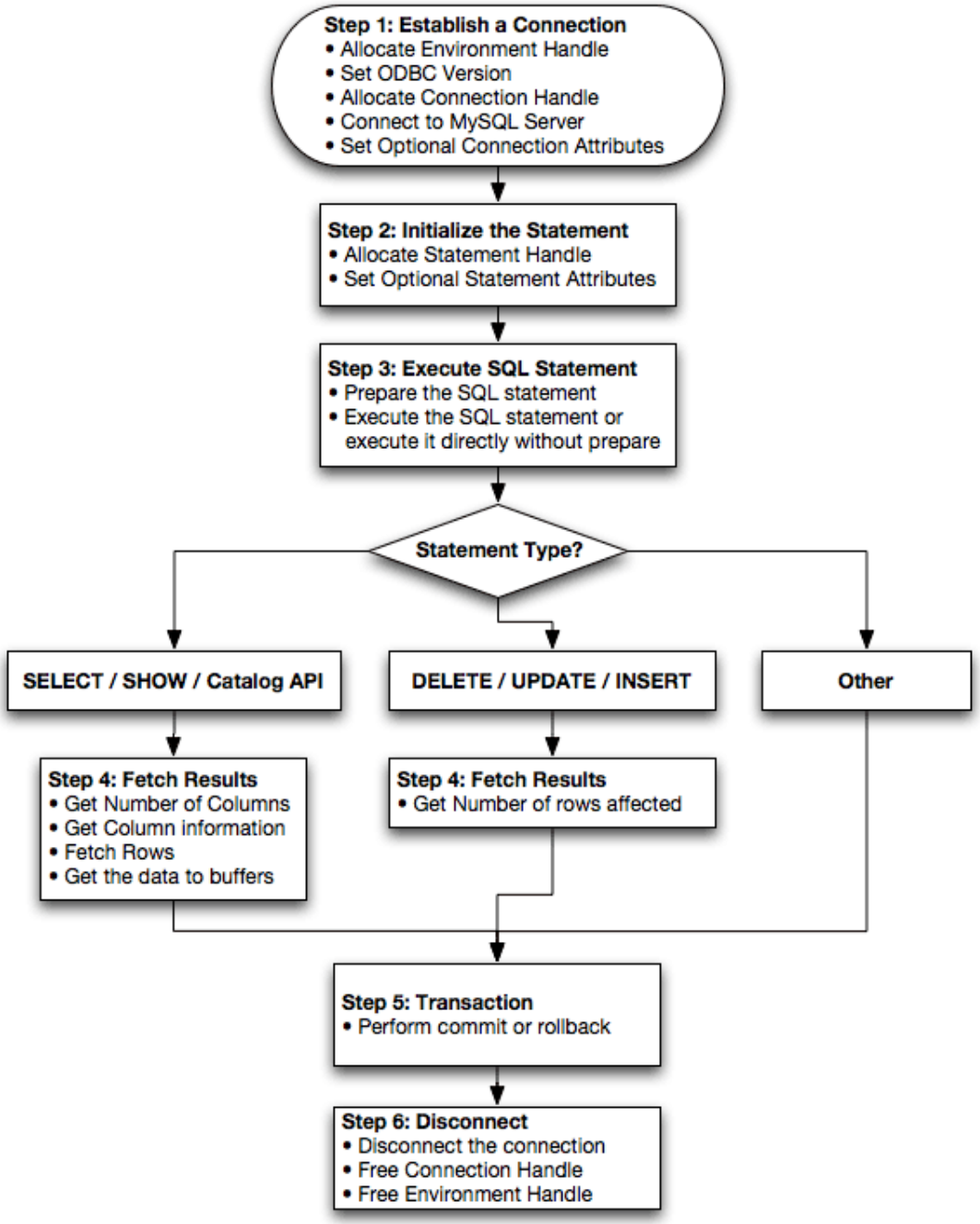


23.1.15. Basic MyODBC Application Steps

Interacting with a MySQL server from MyODBC applications involves the following operations:

- Configure the MyODBC DSN
- Connect to MySQL server
- Initialization operations
- Execute SQL statements
- Retrieve results
- Perform Transactions
- Disconnect from the server

Most applications use some variation of these steps. The basic application steps are shown in the following diagram:



23.1.16. MyODBC API Reference

This section summarizes ODBC routines, categorized by functionality.

For the complete ODBC API reference, please refer to the ODBC Programer's Reference at http://msdn.microsoft.com/library/en-us/odbc/htm/odbcabout_this_manual.asp.

An application can call `SQLGetInfo` function to obtain conformance information about MyODBC. To obtain information about support for a specific function in the driver, an application can call `SQLGetFunctions`.

Note: For backward compatibility, the MyODBC 3.51 driver supports all deprecated functions.

The following tables list MyODBC API calls grouped by task:

Connecting to a data source:

Function name	MyODBC	MyODBC	Conformance	Purpose
	2.50	**3.51**		
`SQLAllocHandle`	No	Yes	ISO 92	Obtains an environment, connection, statement, or descriptor handle.
`SQLConnect`	Yes	Yes	ISO 92	Connects to a specific driver by data source name, user ID, and password.
`SQLDriverConnect`	Yes	Yes	ODBC	Connects to a specific driver by connection string or requests that the Driver Manager and driver display connection dialog boxes for the user.
`SQLAllocEnv`	Yes	Yes	Deprecated	Obtains an environment handle allocated from driver.
`SQLAllocConnect`	Yes	Yes	Deprecated	Obtains a connection handle

Obtaining information about a driver and data source:

Function name	MyODBC	MyODBC	Conformance	Purpose
	2.50	**3.51**		
`SQLDataSources`	No	No	ISO 92	Returns the list of available data sources, handled by the Driver Manager
`SQLDrivers`	No	No	ODBC	Returns the list of installed drivers and their attributes, handles by Driver Manager
`SQLGetInfo`	Yes	Yes	ISO 92	Returns information about a specific driver and data source.
`SQLGetFunctions`	Yes	Yes	ISO 92	Returns supported driver functions.
`SQLGetTypeInfo`	Yes	Yes	ISO 92	Returns information about supported data types.

Setting and retrieving driver attributes:

Function name	MyODBC	MyODBC	Conformance	Purpose
	2.50	**3.51**		
`SQLSetConnectAttr`	No	Yes	ISO 92	Sets a connection attribute.
`SQLGetConnectAttr`	No	Yes	ISO 92	Returns the value of a connection attribute.

SQLSetConnect-Option	Yes	Yes	Deprecated	Sets a connection option
SQLGetConnect-Option	Yes	Yes	Deprecated	Returns the value of a connection option
SQLSetEnvAttr	No	Yes	ISO 92	Sets an environment attribute.
SQLGetEnvAttr	No	Yes	ISO 92	Returns the value of an environment attribute.
SQLSetStmtAttr	No	Yes	ISO 92	Sets a statement attribute.
SQLGetStmtAttr	No	Yes	ISO 92	Returns the value of a statement attribute.
SQLSetStmtOp-tion	Yes	Yes	Deprecated	Sets a statement option
SQLGetStmtOp-tion	Yes	Yes	Deprecated	Returns the value of a statement option

Preparing SQL requests:

Function name	MyODBC	MyODBC	Conformance	Purpose
	2.50	**3.51**		
SQLAllocStmt	Yes	Yes	Deprecated	Allocates a statement handle
SQLPrepare	Yes	Yes	ISO 92	Prepares an SQL statement for later execution.
SQLBindPara-meter	Yes	Yes	ODBC	Assigns storage for a parameter in an SQL statement.
SQLGetCursor-Name	Yes	Yes	ISO 92	Returns the cursor name associated with a statement handle.
SQLSetCursor-Name	Yes	Yes	ISO 92	Specifies a cursor name.
SQLSetScrol-lOptions	Yes	Yes	ODBC	Sets options that control cursor behavior.

Submitting requests:

Function name	MyODBC	MyODBC	Conformance	Purpose
	2.50	**3.51**		
SQLExecute	Yes	Yes	ISO 92	Executes a prepared statement.
SQLExecDirect	Yes	Yes	ISO 92	Executes a statement
SQLNativeSql	Yes	Yes	ODBC	Returns the text of an SQL statement as translated by the driver.
SQLDescribe-Param	Yes	Yes	ODBC	Returns the description for a specific parameter in a statement.
SQLNumParams	Yes	Yes	ISO 92	Returns the number of parameters in a statement.
SQLParamData	Yes	Yes	ISO 92	Used in conjunction with SQLPutData to supply parameter data at execution time. (Useful for long data values.)

SQLPutData	Yes	Yes	ISO 92	Sends part or all of a data value for a parameter. (Useful for long data values.)

Retrieving results and information about results:

Function name	MyODBC	MyODBC	Conformance	Purpose
	2.50	**3.51**		
SQLRowCount	Yes	Yes	ISO 92	Returns the number of rows affected by an insert, update, or delete request.
SQLNumResult-Cols	Yes	Yes	ISO 92	Returns the number of columns in the result set.
SQLDescribeCol	Yes	Yes	ISO 92	Describes a column in the result set.
SQLColAttrib-ute	No	Yes	ISO 92	Describes attributes of a column in the result set.
SQLColAttrib-utes	Yes	Yes	Deprecated	Describes attributes of a column in the result set.
SQLFetch	Yes	Yes	ISO 92	Returns multiple result rows.
SQLFetchScroll	No	Yes	ISO 92	Returns scrollable result rows.
SQLExtended-Fetch	Yes	Yes	Deprecated	Returns scrollable result rows.
SQLSetPos	Yes	Yes	ODBC	Positions a cursor within a fetched block of data and allows an application to refresh data in the rowset or to update or delete data in the result set.
SQLBulkOpera-tions	No	Yes	ODBC	Performs bulk insertions and bulk bookmark operations, including update, delete, and fetch by bookmark.

Retrieving error or diagnostic information:

Function name	MyODBC	MyODBC	Conformance	Purpose
	2.50	**3.51**		
SQLError	Yes	Yes	Deprecated	Returns additional error or status information
SQLGet-DiagField	Yes	Yes	ISO 92	Returns additional diagnostic information (a single field of the diagnostic data structure).
SQLGetDiagRec	Yes	Yes	ISO 92	Returns additional diagnostic information (multiple fields of the diagnostic data structure).

Obtaining information about the data source's system tables (catalog functions) item:

Function name	MyODBC	MyODBC	Conformance	Purpose
	2.50	**3.51**		
`SQLColumnPrivileges`	Yes	Yes	ODBC	Returns a list of columns and associated privileges for one or more tables.
`SQLColumns`	Yes	Yes	X/Open	Returns the list of column names in specified tables.
`SQLForeignKeys`	Yes	Yes	ODBC	Returns a list of column names that make up foreign keys, if they exist for a specified table.
`SQLPrimaryKeys`	Yes	Yes	ODBC	Returns the list of column names that make up the primary key for a table.
`SQLSpecialColumns`	Yes	Yes	X/Open	Returns information about the optimal set of columns that uniquely identifies a row in a specified table, or the columns that are automatically updated when any value in the row is updated by a transaction.
`SQLStatistics`	Yes	Yes	ISO 92	Returns statistics about a single table and the list of indexes associated with the table.
`SQLTablePrivileges`	Yes	Yes	ODBC	Returns a list of tables and the privileges associated with each table.
`SQLTables`	Yes	Yes	X/Open	Returns the list of table names stored in a specific data source.

Performing transactions:

Function name	MyODBC	MyODBC	Conformance	Purpose
	2.50	**3.51**		
`SQLTransact`	Yes	Yes	Deprecated	Commits or rolls back a transaction
`SQLEndTran`	No	Yes	ISO 92	Commits or rolls back a transaction.

Terminating a statement:

Function name	MyODBC	MyODBC	Conformance	Purpose
	2.50	**3.51**		
`SQLFreeStmt`	Yes	Yes	ISO 92	Ends statement processing, discards pending results, and, optionally, frees all resources associated with the statement handle.
`SQLCloseCursor`	Yes	Yes	ISO 92	Closes a cursor that has been opened on a statement handle.
`SQLCancel`	Yes	Yes	ISO 92	Cancels an SQL statement.

Terminating a connection:

Function name	MyODBC	MyODBC	Conformance	Purpose
	2.50	**3.51**		
SQLDisconnect	Yes	Yes	ISO 92	Closes the connection.
SQLFreeHandle	No	Yes	ISO 92	Releases an environment, connection, statement, or descriptor handle.
SQLFreeConnect	Yes	Yes	Deprecated	Releases connection handle
SQLFreeEnv	Yes	Yes	Deprecated	Releases an environment handle

23.1.17. MyODBC Data Types

The following table illustrates how driver maps the server data types to default SQL and C data types:

Native Value	SQL Type	C Type
bit	SQL_BIT	SQL_C_BIT
tinyint	SQL_TINYINT	SQL_C_STINYINT
tinyint unsigned	SQL_TINYINT	SQL_C_UTINYINT
bigint	SQL_BIGINT	SQL_C_SBIGINT
bigint unsigned	SQL_BIGINT	SQL_C_UBIGINT
long varbinary	SQL_LONGVARBINARY	SQL_C_BINARY
blob	SQL_LONGVARBINARY	SQL_C_BINARY
longblob	SQL_LONGVARBINARY	SQL_C_BINARY
tinyblob	SQL_LONGVARBINARY	SQL_C_BINARY
mediumblob	SQL_LONGVARBINARY	SQL_C_BINARY
long varchar	SQL_LONGVARCHAR	SQL_C_CHAR
text	SQL_LONGVARCHAR	SQL_C_CHAR
mediumtext	SQL_LONGVARCHAR	SQL_C_CHAR
char	SQL_CHAR	SQL_C_CHAR
numeric	SQL_NUMERIC	SQL_C_CHAR
decimal	SQL_DECIMAL	SQL_C_CHAR
integer	SQL_INTEGER	SQL_C_SLONG
integer unsigned	SQL_INTEGER	SQL_C_ULONG
int	SQL_INTEGER	SQL_C_SLONG
int unsigned	SQL_INTEGER	SQL_C_ULONG
mediumint	SQL_INTEGER	SQL_C_SLONG
mediumint unsigned	SQL_INTEGER	SQL_C_ULONG
smallint	SQL_SMALLINT	SQL_C_SSHORT
smallint unsigned	SQL_SMALLINT	SQL_C_USHORT
real	SQL_FLOAT	SQL_C_DOUBLE
double	SQL_FLOAT	SQL_C_DOUBLE
float	SQL_REAL	SQL_C_FLOAT

double precision	SQL_DOUBLE	SQL_C_DOUBLE
date	SQL_DATE	SQL_C_DATE
time	SQL_TIME	SQL_C_TIME
year	SQL_SMALLINT	SQL_C_SHORT
datetime	SQL_TIMESTAMP	SQL_C_TIMESTAMP
timestamp	SQL_TIMESTAMP	SQL_C_TIMESTAMP
text	SQL_VARCHAR	SQL_C_CHAR
varchar	SQL_VARCHAR	SQL_C_CHAR
enum	SQL_VARCHAR	SQL_C_CHAR
set	SQL_VARCHAR	SQL_C_CHAR
bit	SQL_CHAR	SQL_C_CHAR
bool	SQL_CHAR	SQL_C_CHAR

23.1.18. MyODBC Error Codes

The following tables lists the error codes returned by the driver apart from the server errors.

Native Code	SQLSTATE 2	SQLSTATE 3	Error Message
500	01000	01000	General warning
501	01004	01004	String data, right truncated
502	01S02	01S02	Option value changed
503	01S03	01S03	No rows updated/deleted
504	01S04	01S04	More than one row updated/deleted
505	01S06	01S06	Attempt to fetch before the result set returned the first row set
506	07001	07002	SQLBindParameter not used for all parameters
507	07005	07005	Prepared statement not a cursor-specification
508	07009	07009	Invalid descriptor index
509	08002	08002	Connection name in use
510	08003	08003	Connection does not exist
511	24000	24000	Invalid cursor state
512	25000	25000	Invalid transaction state
513	25S01	25S01	Transaction state unknown
514	34000	34000	Invalid cursor name
515	S1000	HY000	General driver defined error
516	S1001	HY001	Memory allocation error
517	S1002	HY002	Invalid column number
518	S1003	HY003	Invalid application buffer type
519	S1004	HY004	Invalid SQL data type
520	S1009	HY009	Invalid use of null pointer
521	S1010	HY010	Function sequence error

522	S1011	HY011	Attribute can not be set now
523	S1012	HY012	Invalid transaction operation code
524	S1013	HY013	Memory management error
525	S1015	HY015	No cursor name available
526	S1024	HY024	Invalid attribute value
527	S1090	HY090	Invalid string or buffer length
528	S1091	HY091	Invalid descriptor field identifier
529	S1092	HY092	Invalid attribute/option identifier
530	S1093	HY093	Invalid parameter number
531	S1095	HY095	Function type out of range
532	S1106	HY106	Fetch type out of range
533	S1117	HY117	Row value out of range
534	S1109	HY109	Invalid cursor position
535	S1C00	HYC00	Optional feature not implemented
0	21S01	21S01	Column count does not match value count
0	23000	23000	Integrity constraint violation
0	42000	42000	Syntax error or access violation
0	42S02	42S02	Base table or view not found
0	42S12	42S12	Index not found
0	42S21	42S21	Column already exists
0	42S22	42S22	Column not found
0	08S01	08S01	Communication link failure

23.1.19. MyODBC With VB: ADO, DAO and RDO

This section contains simple examples of the use of MySQL ODBC 3.51 Driver with ADO, DAO and RDO.

23.1.19.1. ADO: `rs.addNew`, `rs.delete`, and `rs.update`

The following ADO (ActiveX Data Objects) example creates a table `my_ado` and demonstrates the use of `rs.addNew`, `rs.delete`, and `rs.update`.

```
Private Sub myodbc_ado_Click()

  Dim conn As ADODB.Connection
  Dim rs As ADODB.Recordset
  Dim fld As ADODB.Field
  Dim sql As String

  'connect to MySQL server using MySQL ODBC 3.51 Driver
  Set conn = New ADODB.Connection
  conn.ConnectionString = "DRIVER={MySQL ODBC 3.51 Driver};"_
                        & "SERVER=localhost;"_
                        & " DATABASE=test;"_
                        & "UID=venu;PWD=venu; OPTION=3"

  conn.Open
```

```
'create table
conn.Execute "DROP TABLE IF EXISTS my_ado"
conn.Execute "CREATE TABLE my_ado(id int not null primary key, name varchar(20),
                              & "txt text, dt date, tm time, ts timestamp)"

'direct insert
conn.Execute "INSERT INTO my_ado(id,name,txt) values(1,100,'venu')"
conn.Execute "INSERT INTO my_ado(id,name,txt) values(2,200,'MySQL')"
conn.Execute "INSERT INTO my_ado(id,name,txt) values(3,300,'Delete')"

Set rs = New ADODB.Recordset
rs.CursorLocation = adUseServer

'fetch the initial table ..
rs.Open "SELECT * FROM my_ado", conn
  Debug.Print rs.RecordCount
  rs.MoveFirst
  Debug.Print String(50, "-") & "Initial my_ado Result Set " & String(50, "-")
  For Each fld In rs.Fields
    Debug.Print fld.Name,
    Next
    Debug.Print

    Do Until rs.EOF
    For Each fld In rs.Fields
    Debug.Print fld.Value,
    Next
    rs.MoveNext
    Debug.Print
  Loop
rs.Close

'rs insert
rs.Open "select * from my_ado", conn, adOpenDynamic, adLockOptimistic
rs.AddNew
rs!Name = "Monty"
rs!txt = "Insert row"
rs.Update
rs.Close

'rs update
rs.Open "SELECT * FROM my_ado"
rs!Name = "update"
rs!txt = "updated-row"
rs.Update
rs.Close

'rs update second time..
rs.Open "SELECT * FROM my_ado"
rs!Name = "update"
rs!txt = "updated-second-time"
rs.Update
rs.Close

'rs delete
rs.Open "SELECT * FROM my_ado"
rs.MoveNext
rs.MoveNext
rs.Delete
rs.Close

'fetch the updated table ..
rs.Open "SELECT * FROM my_ado", conn
  Debug.Print rs.RecordCount
```

```
    rs.MoveFirst
    Debug.Print String(50, "-") & "Updated my_ado Result Set " & String(50, "-")
    For Each fld In rs.Fields
      Debug.Print fld.Name,
      Next
      Debug.Print

      Do Until rs.EOF
      For Each fld In rs.Fields
      Debug.Print fld.Value,
      Next
      rs.MoveNext
      Debug.Print
    Loop
  rs.Close
  conn.Close
End Sub
```

23.1.19.2. DAO: `rs.addNew`, `rs.update`, and Scrolling

The following DAO (Data Access Objects) example creates a table `my_dao` and demonstrates the use of `rs.addNew`, `rs.update`, and result set scrolling.

```
Private Sub myodbc_dao_Click()

  Dim ws As Workspace
  Dim conn As Connection
  Dim queryDef As queryDef
  Dim str As String

  'connect to MySQL using MySQL ODBC 3.51 Driver
  Set ws = DBEngine.CreateWorkspace("", "venu", "venu", dbUseODBC)
  str = "odbc;DRIVER={MySQL ODBC 3.51 Driver};"_
                       & "SERVER=localhost;"_
                       & " DATABASE=test;"_
                       & "UID=venu;PWD=venu; OPTION=3"
  Set conn = ws.OpenConnection("test", dbDriverNoPrompt, False, str)

  'Create table my_dao
  Set queryDef = conn.CreateQueryDef("", "drop table if exists my_dao")
  queryDef.Execute

  Set queryDef = conn.CreateQueryDef("", "create table my_dao(Id INT AUTO_INCREMEN
                                                            & "Ts TIMESTAMP(14) NOT
  queryDef.Execute

  'Insert new records using rs.addNew
  Set rs = conn.OpenRecordset("my_dao")
  Dim i As Integer

    For i = 10 To 15
    rs.AddNew
    rs!Name = "insert record" & i
    rs!Id2 = i
    rs.Update
    Next i
             rs.Close

  'rs update..
  Set rs = conn.OpenRecordset("my_dao")
  rs.Edit
  rs!Name = "updated-string"
```

```
  rs.Update
  rs.Close

  'fetch the table back...
  Set rs = conn.OpenRecordset("my_dao", dbOpenDynamic)
  str = "Results:"
  rs.MoveFirst
  While Not rs.EOF
  str = " " & rs!Id & " , " & rs!Name & ", " & rs!Ts & ", " & rs!Id2
  Debug.Print "DATA:" & str
  rs.MoveNext
  Wend

  'rs Scrolling
  rs.MoveFirst
  str = " FIRST ROW: " & rs!Id & " , " & rs!Name & ", " & rs!Ts & ", " & rs!Id2
  Debug.Print str

  rs.MoveLast
  str = " LAST ROW: " & rs!Id & " , " & rs!Name & ", " & rs!Ts & ", " & rs!Id2
  Debug.Print str

  rs.MovePrevious
  str = " LAST-1 ROW: " & rs!Id & " , " & rs!Name & ", " & rs!Ts & ", " & rs!Id2
  Debug.Print str

  'free all resources
  rs.Close
  queryDef.Close
  conn.Close
  ws.Close

End Sub
```

23.1.19.3. RDO: `rs.addNew` and `rs.update`

The following RDO (Remote Data Objects) example creates a table `my_rdo` and demonstrates the use of `rs.addNew` and `rs.update`.

```
Dim rs As rdoResultset
    Dim cn As New rdoConnection
    Dim cl As rdoColumn
    Dim SQL As String

    'cn.Connect = "DSN=test;"
    cn.Connect = "DRIVER={MySQL ODBC 3.51 Driver};"_
                        & "SERVER=localhost;"_
                        & " DATABASE=test;"_
                        & "UID=venu;PWD=venu; OPTION=3"

    cn.CursorDriver = rdUseOdbc
    cn.EstablishConnection rdDriverPrompt


    'drop table my_rdo
    SQL = "drop table if exists my_rdo"
    cn.Execute SQL, rdExecDirect

    'create table my_rdo
    SQL = "create table my_rdo(id int, name varchar(20))"
    cn.Execute SQL, rdExecDirect
```

```
    'insert - direct
    SQL = "insert into my_rdo values (100,'venu')"
    cn.Execute SQL, rdExecDirect

    SQL = "insert into my_rdo values (200,'MySQL')"
    cn.Execute SQL, rdExecDirect

    'rs insert
    SQL = "select * from my_rdo"
    Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
    rs.AddNew
    rs!id = 300
    rs!Name = "Insert1"
    rs.Update
    rs.Close

    'rs insert
    SQL = "select * from my_rdo"
    Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
    rs.AddNew
    rs!id = 400
    rs!Name = "Insert 2"
    rs.Update
    rs.Close

    'rs update
    SQL = "select * from my_rdo"
    Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
    rs.Edit
    rs!id = 999
    rs!Name = "updated"
    rs.Update
    rs.Close

    'fetch back...
    SQL = "select * from my_rdo"
    Set rs = cn.OpenResultset(SQL, rdOpenStatic, rdConcurRowVer, rdExecDirect)
    Do Until rs.EOF
    For Each cl In rs.rdoColumns
                Debug.Print cl.Value,
      Next
      rs.MoveNext
      Debug.Print
               Loop
    Debug.Print "Row count="; rs.RowCount

    'close
    rs.Close
    cn.Close

End Sub
```

23.1.20. MyODBC with Microsoft .NET

This section contains simple examples that demonstrate the use of MyODBC drivers with ODBC.NET.

23.1.20.1. ODBC.NET: CSHARP(C#)

The following sample creates a table `my_odbc_net` and demonstrates the use in C#.

```
/**
* @sample    : mycon.cs
* @purpose   : Demo sample for ODBC.NET using MyODBC
* @author    : Venu, <venu@mysql.com>
*
* (C) Copyright MySQL AB, 1995-2004
*
**/

/* build command
 *
 *  csc /t:exe
 *      /out:mycon.exe mycon.cs
 *      /r:Microsoft.Data.Odbc.dll
 */

using Console = System.Console;
using Microsoft.Data.Odbc;

namespace myodbc3
{
 class mycon
 {
   static void Main(string[] args)
   {
     try
     {
       //Connection string for MyODBC 2.50
       /*string MyConString = "DRIVER={MySQL};" +
                            "SERVER=localhost;" +
                            "DATABASE=test;" +
                            "UID=venu;" +
                            "PASSWORD=venu;" +
                            "OPTION=3";
       */
       //Connection string for MyODBC 3.51
       string MyConString = "DRIVER={MySQL ODBC 3.51 Driver};" +
                            "SERVER=localhost;" +
                            "DATABASE=test;" +
                            "UID=venu;" +
                            "PASSWORD=venu;" +
                            "OPTION=3";

       //Connect to MySQL using MyODBC
       OdbcConnection MyConnection = new OdbcConnection(MyConString);
       MyConnection.Open();

       Console.WriteLine("\n !!! success, connected successfully !!!\n");

       //Display connection information
       Console.WriteLine("Connection Information:");
       Console.WriteLine("\tConnection String:" + MyConnection.ConnectionString);
       Console.WriteLine("\tConnection Timeout:" + MyConnection.ConnectionTimeout
       Console.WriteLine("\tDatabase:" + MyConnection.Database);
       Console.WriteLine("\tDataSource:" + MyConnection.DataSource);
       Console.WriteLine("\tDriver:" + MyConnection.Driver);
       Console.WriteLine("\tServerVersion:" + MyConnection.ServerVersion);

       //Create a sample table
       OdbcCommand MyCommand = new OdbcCommand("DROP TABLE IF EXISTS my_odbc_net"
       MyCommand.ExecuteNonQuery();
       MyCommand.CommandText = "CREATE TABLE my_odbc_net(id int, name varchar(20)
       MyCommand.ExecuteNonQuery();
```

```
        //Insert
        MyCommand.CommandText = "INSERT INTO my_odbc_net VALUES(10,'venu', 300)";
        Console.WriteLine("INSERT, Total rows affected:" + MyCommand.ExecuteNonQue

        //Insert
        MyCommand.CommandText = "INSERT INTO my_odbc_net VALUES(20,'mysql',400)";
        Console.WriteLine("INSERT, Total rows affected:" + MyCommand.ExecuteNonQue

        //Insert
        MyCommand.CommandText = "INSERT INTO my_odbc_net VALUES(20,'mysql',500)";
        Console.WriteLine("INSERT, Total rows affected:" + MyCommand.ExecuteNonQue

        //Update
        MyCommand.CommandText = "UPDATE my_odbc_net SET id=999 WHERE id=20";
        Console.WriteLine("Update, Total rows affected:" + MyCommand.ExecuteNonQue

        //COUNT(*)
        MyCommand.CommandText = "SELECT COUNT(*) as TRows FROM my_odbc_net";
        Console.WriteLine("Total Rows:" + MyCommand.ExecuteScalar());

        //Fetch
        MyCommand.CommandText = "SELECT * FROM my_odbc_net";
        OdbcDataReader MyDataReader;
        MyDataReader =  MyCommand.ExecuteReader();
        while (MyDataReader.Read())
        {
         if(string.Compare(MyConnection.Driver,"myodbc3.dll") == 0) {
           Console.WriteLine("Data:" + MyDataReader.GetInt32(0) + " " +
                                       MyDataReader.GetString(1) + " " +
                                       MyDataReader.GetInt64(2)); //Supported only
         }
         else {
           Console.WriteLine("Data:" + MyDataReader.GetInt32(0) + " " +
                                       MyDataReader.GetString(1) + " " +
                                       MyDataReader.GetInt32(2)); //BIGINTs not su
         }
        }

        //Close all resources
        MyDataReader.Close();
        MyConnection.Close();
      }
      catch (OdbcException MyOdbcException)//Catch any ODBC exception ..
      {
        for (int i=0; i < MyOdbcException.Errors.Count; i++)
        {
          Console.Write("ERROR #" + i + "\n" +
            "Message: " + MyOdbcException.Errors[i].Message + "\n" +
            "Native: " + MyOdbcException.Errors[i].NativeError.ToString() + "\n" +
            "Source: " + MyOdbcException.Errors[i].Source + "\n" +
            "SQL: " + MyOdbcException.Errors[i].SQLState + "\n");
        }
      }
    }
  }
}
```

23.1.20.2. ODBC.NET: VB

The following sample creates a table `my_vb_net` and demonstrates the use in VB.

```
' @sample    : myvb.vb
' @purpose   : Demo sample for ODBC.NET using MyODBC
```

```
' @author    : Venu, <venu@mysql.com>
'
' (C) Copyright MySQL AB, 1995-2004
'
'

'
' build command
'
' vbc /target:exe
'     /out:myvb.exe
'     /r:Microsoft.Data.Odbc.dll
'     /r:System.dll
'     /r:System.Data.dll
'

Imports Microsoft.Data.Odbc
Imports System

Module myvb
    Sub Main()
        Try

            'MyODBC 3.51 connection string
            Dim MyConString As String = "DRIVER={MySQL ODBC 3.51 Driver};" & _
                           "SERVER=localhost;" & _
                           "DATABASE=test;" & _
                           "UID=venu;" & _
                           "PASSWORD=venu;" & _
                           "OPTION=3;"

            'Connection
            Dim MyConnection As New OdbcConnection(MyConString)
            MyConnection.Open()

            Console.WriteLine ("Connection State::" & MyConnection.State.ToString)

            'Drop
            Console.WriteLine ("Dropping table")
            Dim MyCommand As New OdbcCommand()
            MyCommand.Connection = MyConnection
            MyCommand.CommandText = "DROP TABLE IF EXISTS my_vb_net"
            MyCommand.ExecuteNonQuery()

            'Create
            Console.WriteLine ("Creating....")
            MyCommand.CommandText = "CREATE TABLE my_vb_net(id int, name varchar(3
            MyCommand.ExecuteNonQuery()

            'Insert
            MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(10,'venu')"
            Console.WriteLine("INSERT, Total rows affected:" & MyCommand.ExecuteNo

            'Insert
            MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(20,'mysql')"
            Console.WriteLine("INSERT, Total rows affected:" & MyCommand.ExecuteNo

            'Insert
            MyCommand.CommandText = "INSERT INTO my_vb_net VALUES(20,'mysql')"
            Console.WriteLine("INSERT, Total rows affected:" & MyCommand.ExecuteNo

            'Insert
            MyCommand.CommandText = "INSERT INTO my_vb_net(id) VALUES(30)"
            Console.WriteLine("INSERT, Total rows affected:" & MyCommand.ExecuteNo
```

```
            'Update
            MyCommand.CommandText = "UPDATE my_vb_net SET id=999 WHERE id=20"
            Console.WriteLine("Update, Total rows affected:" & MyCommand.ExecuteNo

            'COUNT(*)
            MyCommand.CommandText = "SELECT COUNT(*) as TRows FROM my_vb_net"
            Console.WriteLine("Total Rows:" & MyCommand.ExecuteScalar())

            'Select
            Console.WriteLine ("Select * FROM my_vb_net")
            MyCommand.CommandText = "SELECT * FROM my_vb_net"
            Dim MyDataReader As OdbcDataReader
            MyDataReader = MyCommand.ExecuteReader
            While MyDataReader.Read
                If MyDataReader("name") Is DBNull.Value Then
                    Console.WriteLine ("id = " & CStr(MyDataReader("id")) & "  nam
                      "NULL")
                Else
                    Console.WriteLine ("id = " & CStr(MyDataReader("id")) & "  nam
                                            CStr(MyDataReader("name")))
                End If
            End While

        'Catch ODBC Exception
        Catch MyOdbcException As OdbcException
            Dim i As Integer
            Console.WriteLine (MyOdbcException.ToString)

        'Catch program exception
        Catch MyException As Exception
            Console.WriteLine (MyException.ToString)
    End Try
    End Sub
End Module
```

23.1.21. Credits

These are the developers that have worked on the MyODBC and MyODBC 3.51 Drivers from MySQL AB.

- Micheal (Monty) Widenius
- Venu Anuganti
- Peter Harvey

23.2. MySQL Java Connectivity (JDBC)

There are two supported JDBC drivers for MySQL:

- `MySQL Connector/J` from MySQL AB, implemented in 100% native Java. This product was formerly known as the `mm.mysql` driver. You can download `MySQL Connector/J` from http://dev.mysql.com/downloads/connector/j/.
- The Resin JDBC driver, which can be found at http://www.caucho.com/projects/jdbc-mysql/index.xtp.

For more information, consult any general JDBC documentation, plus each driver's own documentation for MySQL-specific features.

Documentation for `MySQL Connector/J` is available online at the MySQL AB Web site at http://dev.mysql.com/doc/.

Chapter 24. Error Handling in MySQL

This chapter lists the errors that may appear when you call MySQL from any host language. The first list displays server error messages. The second list displays client program messages.

Server error information comes from the following files:

- The Error values and the symbols in parentheses correspond to definitions in the `include/mysqld_error.h` MySQL source file.
- The SQLSTATE values correspond to definitions in the `include/sql_state.h` MySQL source file.

  SQLSTATE error codes are displayed only if you use MySQL version 4.1 and up. SQLSTATE codes were added for compatibility with X/Open, ANSI, and ODBC behavior.
- The Message values correspond to the error messages that are listed in the `sql/share/english/errmsg.txt` file. %d or %s represent numbers or strings that are substituted into the messages %when they are displayed.

Because updates are frequent, it is possible that these files contain additional error information not listed here.

- Error: `1000` SQLSTATE: `HY000` (`ER_HASHCHK`)

  Message: hashchk
- Error: `1001` SQLSTATE: `HY000` (`ER_NISAMCHK`)

  Message: isamchk
- Error: `1002` SQLSTATE: `HY000` (`ER_NO`)

  Message: NO
- Error: `1003` SQLSTATE: `HY000` (`ER_YES`)

  Message: YES
- Error: `1004` SQLSTATE: `HY000` (`ER_CANT_CREATE_FILE`)

  Message: Can't create file '%s' (errno: %d)
- Error: `1005` SQLSTATE: `HY000` (`ER_CANT_CREATE_TABLE`)

  Message: Can't create table '%s' (errno: %d)
- Error: `1006` SQLSTATE: `HY000` (`ER_CANT_CREATE_DB`)

  Message: Can't create database '%s' (errno: %d)
- Error: `1007` SQLSTATE: `HY000` (`ER_DB_CREATE_EXISTS`)

  Message: Can't create database '%s'; database exists

- Error: `1008` SQLSTATE: `HY000` (`ER_DB_DROP_EXISTS`)

  Message: Can't drop database '%s'; database doesn't exist
- Error: `1009` SQLSTATE: `HY000` (`ER_DB_DROP_DELETE`)

  Message: Error dropping database (can't delete '%s', errno: %d)
- Error: `1010` SQLSTATE: `HY000` (`ER_DB_DROP_RMDIR`)

  Message: Error dropping database (can't rmdir '%s', errno: %d)
- Error: `1011` SQLSTATE: `HY000` (`ER_CANT_DELETE_FILE`)

  Message: Error on delete of '%s' (errno: %d)
- Error: `1012` SQLSTATE: `HY000` (`ER_CANT_FIND_SYSTEM_REC`)

  Message: Can't read record in system table
- Error: `1013` SQLSTATE: `HY000` (`ER_CANT_GET_STAT`)

  Message: Can't get status of '%s' (errno: %d)
- Error: `1014` SQLSTATE: `HY000` (`ER_CANT_GET_WD`)

  Message: Can't get working directory (errno: %d)
- Error: `1015` SQLSTATE: `HY000` (`ER_CANT_LOCK`)

  Message: Can't lock file (errno: %d)
- Error: `1016` SQLSTATE: `HY000` (`ER_CANT_OPEN_FILE`)

  Message: Can't open file: '%s' (errno: %d)
- Error: `1017` SQLSTATE: `HY000` (`ER_FILE_NOT_FOUND`)

  Message: Can't find file: '%s' (errno: %d)
- Error: `1018` SQLSTATE: `HY000` (`ER_CANT_READ_DIR`)

  Message: Can't read dir of '%s' (errno: %d)
- Error: `1019` SQLSTATE: `HY000` (`ER_CANT_SET_WD`)

  Message: Can't change dir to '%s' (errno: %d)
- Error: `1020` SQLSTATE: `HY000` (`ER_CHECKREAD`)

  Message: Record has changed since last read in table '%s'
- Error: `1021` SQLSTATE: `HY000` (`ER_DISK_FULL`)

  Message: Disk full (%s); waiting for someone to free some space...
- Error: `1022` SQLSTATE: `23000` (`ER_DUP_KEY`)

  Message: Can't write; duplicate key in table '%s'

- Error: `1023` SQLSTATE: `HY000` (`ER_ERROR_ON_CLOSE`)

  Message: Error on close of '%s' (errno: %d)
- Error: `1024` SQLSTATE: `HY000` (`ER_ERROR_ON_READ`)

  Message: Error reading file '%s' (errno: %d)
- Error: `1025` SQLSTATE: `HY000` (`ER_ERROR_ON_RENAME`)

  Message: Error on rename of '%s' to '%s' (errno: %d)
- Error: `1026` SQLSTATE: `HY000` (`ER_ERROR_ON_WRITE`)

  Message: Error writing file '%s' (errno: %d)
- Error: `1027` SQLSTATE: `HY000` (`ER_FILE_USED`)

  Message: '%s' is locked against change
- Error: `1028` SQLSTATE: `HY000` (`ER_FILSORT_ABORT`)

  Message: Sort aborted
- Error: `1029` SQLSTATE: `HY000` (`ER_FORM_NOT_FOUND`)

  Message: View '%s' doesn't exist for '%s'
- Error: `1030` SQLSTATE: `HY000` (`ER_GET_ERRNO`)

  Message: Got error %d from storage engine
- Error: `1031` SQLSTATE: `HY000` (`ER_ILLEGAL_HA`)

  Message: Table storage engine for '%s' doesn't have this option
- Error: `1032` SQLSTATE: `HY000` (`ER_KEY_NOT_FOUND`)

  Message: Can't find record in '%s'
- Error: `1033` SQLSTATE: `HY000` (`ER_NOT_FORM_FILE`)

  Message: Incorrect information in file: '%s'
- Error: `1034` SQLSTATE: `HY000` (`ER_NOT_KEYFILE`)

  Message: Incorrect key file for table '%s'; try to repair it
- Error: `1035` SQLSTATE: `HY000` (`ER_OLD_KEYFILE`)

  Message: Old key file for table '%s'; repair it!
- Error: `1036` SQLSTATE: `HY000` (`ER_OPEN_AS_READONLY`)

  Message: Table '%s' is read only
- Error: `1037` SQLSTATE: `HY001` (`ER_OUTOFMEMORY`)

  Message: Out of memory; restart server and try again (needed %d bytes)

- Error: `1038` SQLSTATE: `HY001` (`ER_OUT_OF_SORTMEMORY`)

  Message: Out of sort memory; increase server sort buffer size

- Error: `1039` SQLSTATE: `HY000` (`ER_UNEXPECTED_EOF`)

  Message: Unexpected EOF found when reading file '%s' (errno: %d)

- Error: `1040` SQLSTATE: `08004` (`ER_CON_COUNT_ERROR`)

  Message: Too many connections

- Error: `1041` SQLSTATE: `HY000` (`ER_OUT_OF_RESOURCES`)

  Message: Out of memory; check if mysqld or some other process uses all available memory; if not, you may have to use 'ulimit' to allow mysqld to use more memory or you can add more swap space

- Error: `1042` SQLSTATE: `08S01` (`ER_BAD_HOST_ERROR`)

  Message: Can't get hostname for your address

- Error: `1043` SQLSTATE: `08S01` (`ER_HANDSHAKE_ERROR`)

  Message: Bad handshake

- Error: `1044` SQLSTATE: `42000` (`ER_DBACCESS_DENIED_ERROR`)

  Message: Access denied for user '%s'@'%s' to database '%s'

- Error: `1045` SQLSTATE: `28000` (`ER_ACCESS_DENIED_ERROR`)

  Message: Access denied for user '%s'@'%s' (using password: %s)

- Error: `1046` SQLSTATE: `3D000` (`ER_NO_DB_ERROR`)

  Message: No database selected

- Error: `1047` SQLSTATE: `08S01` (`ER_UNKNOWN_COM_ERROR`)

  Message: Unknown command

- Error: `1048` SQLSTATE: `23000` (`ER_BAD_NULL_ERROR`)

  Message: Column '%s' cannot be null

- Error: `1049` SQLSTATE: `42000` (`ER_BAD_DB_ERROR`)

  Message: Unknown database '%s'

- Error: `1050` SQLSTATE: `42S01` (`ER_TABLE_EXISTS_ERROR`)

  Message: Table '%s' already exists

- Error: `1051` SQLSTATE: `42S02` (`ER_BAD_TABLE_ERROR`)

  Message: Unknown table '%s'

- Error: `1052` SQLSTATE: `23000` (`ER_NON_UNIQ_ERROR`)

  Message: Column '%s' in %s is ambiguous

- Error: 1053 SQLSTATE: 08S01 (ER_SERVER_SHUTDOWN)

  Message: Server shutdown in progress
- Error: 1054 SQLSTATE: 42S22 (ER_BAD_FIELD_ERROR)

  Message: Unknown column '%s' in '%s'
- Error: 1055 SQLSTATE: 42000 (ER_WRONG_FIELD_WITH_GROUP)

  Message: '%s' isn't in GROUP BY
- Error: 1056 SQLSTATE: 42000 (ER_WRONG_GROUP_FIELD)

  Message: Can't group on '%s'
- Error: 1057 SQLSTATE: 42000 (ER_WRONG_SUM_SELECT)

  Message: Statement has sum functions and columns in same statement
- Error: 1058 SQLSTATE: 21S01 (ER_WRONG_VALUE_COUNT)

  Message: Column count doesn't match value count
- Error: 1059 SQLSTATE: 42000 (ER_TOO_LONG_IDENT)

  Message: Identifier name '%s' is too long
- Error: 1060 SQLSTATE: 42S21 (ER_DUP_FIELDNAME)

  Message: Duplicate column name '%s'
- Error: 1061 SQLSTATE: 42000 (ER_DUP_KEYNAME)

  Message: Duplicate key name '%s'
- Error: 1062 SQLSTATE: 23000 (ER_DUP_ENTRY)

  Message: Duplicate entry '%s' for key %d
- Error: 1063 SQLSTATE: 42000 (ER_WRONG_FIELD_SPEC)

  Message: Incorrect column specifier for column '%s'
- Error: 1064 SQLSTATE: 42000 (ER_PARSE_ERROR)

  Message: %s near '%s' at line %d
- Error: 1065 SQLSTATE: 42000 (ER_EMPTY_QUERY)

  Message: Query was empty
- Error: 1066 SQLSTATE: 42000 (ER_NONUNIQ_TABLE)

  Message: Not unique table/alias: '%s'
- Error: 1067 SQLSTATE: 42000 (ER_INVALID_DEFAULT)

  Message: Invalid default value for '%s'

- Error: 1068 SQLSTATE: 42000 (ER_MULTIPLE_PRI_KEY)

  Message: Multiple primary key defined

- Error: 1069 SQLSTATE: 42000 (ER_TOO_MANY_KEYS)

  Message: Too many keys specified; max %d keys allowed

- Error: 1070 SQLSTATE: 42000 (ER_TOO_MANY_KEY_PARTS)

  Message: Too many key parts specified; max %d parts allowed

- Error: 1071 SQLSTATE: 42000 (ER_TOO_LONG_KEY)

  Message: Specified key was too long; max key length is %d bytes

- Error: 1072 SQLSTATE: 42000 (ER_KEY_COLUMN_DOES_NOT_EXITS)

  Message: Key column '%s' doesn't exist in table

- Error: 1073 SQLSTATE: 42000 (ER_BLOB_USED_AS_KEY)

  Message: BLOB column '%s' can't be used in key specification with the used table type

- Error: 1074 SQLSTATE: 42000 (ER_TOO_BIG_FIELDLENGTH)

  Message: Column length too big for column '%s' (max = %d); use BLOB instead

- Error: 1075 SQLSTATE: 42000 (ER_WRONG_AUTO_KEY)

  Message: Incorrect table definition; there can be only one auto column and it must be defined as a key

- Error: 1076 SQLSTATE: HY000 (ER_READY)

  Message: %s: ready for connections. Version: '%s' socket: '%s' port: %d

- Error: 1077 SQLSTATE: HY000 (ER_NORMAL_SHUTDOWN)

  Message: %s: Normal shutdown

- Error: 1078 SQLSTATE: HY000 (ER_GOT_SIGNAL)

  Message: %s: Got signal %d. Aborting!

- Error: 1079 SQLSTATE: HY000 (ER_SHUTDOWN_COMPLETE)

  Message: %s: Shutdown complete

- Error: 1080 SQLSTATE: 08S01 (ER_FORCING_CLOSE)

  Message: %s: Forcing close of thread %ld user: '%s'

- Error: 1081 SQLSTATE: 08S01 (ER_IPSOCK_ERROR)

  Message: Can't create IP socket

- Error: 1082 SQLSTATE: 42S12 (ER_NO_SUCH_INDEX)

  Message: Table '%s' has no index like the one used in CREATE INDEX; recreate the table

- Error: 1083 SQLSTATE: 42000 (ER_WRONG_FIELD_TERMINATORS)

  Message: Field separator argument is not what is expected; check the manual

- Error: 1084 SQLSTATE: 42000 (ER_BLOBS_AND_NO_TERMINATED)

  Message: You can't use fixed rowlength with BLOBs; please use 'fields terminated by'

- Error: 1085 SQLSTATE: HY000 (ER_TEXTFILE_NOT_READABLE)

  Message: The file '%s' must be in the database directory or be readable by all

- Error: 1086 SQLSTATE: HY000 (ER_FILE_EXISTS_ERROR)

  Message: File '%s' already exists

- Error: 1087 SQLSTATE: HY000 (ER_LOAD_INFO)

  Message: Records: %ld Deleted: %ld Skipped: %ld Warnings: %ld

- Error: 1088 SQLSTATE: HY000 (ER_ALTER_INFO)

  Message: Records: %ld Duplicates: %ld

- Error: 1089 SQLSTATE: HY000 (ER_WRONG_SUB_KEY)

  Message: Incorrect sub part key; the used key part isn't a string, the used length is longer than the key part, or the storage engine doesn't support unique sub keys

- Error: 1090 SQLSTATE: 42000 (ER_CANT_REMOVE_ALL_FIELDS)

  Message: You can't delete all columns with ALTER TABLE; use DROP TABLE instead

- Error: 1091 SQLSTATE: 42000 (ER_CANT_DROP_FIELD_OR_KEY)

  Message: Can't DROP '%s'; check that column/key exists

- Error: 1092 SQLSTATE: HY000 (ER_INSERT_INFO)

  Message: Records: %ld Duplicates: %ld Warnings: %ld

- Error: 1093 SQLSTATE: HY000 (ER_UPDATE_TABLE_USED)

  Message: You can't specify target table '%s' for update in FROM clause

- Error: 1094 SQLSTATE: HY000 (ER_NO_SUCH_THREAD)

  Message: Unknown thread id: %lu

- Error: 1095 SQLSTATE: HY000 (ER_KILL_DENIED_ERROR)

  Message: You are not owner of thread %lu

- Error: 1096 SQLSTATE: HY000 (ER_NO_TABLES_USED)

  Message: No tables used

- Error: 1097 SQLSTATE: HY000 (ER_TOO_BIG_SET)

  Message: Too many strings for column %s and SET

- Error: `1098` SQLSTATE: `HY000` (`ER_NO_UNIQUE_LOGFILE`)

  Message: Can't generate a unique log-filename %s.(1-999)

- Error: `1099` SQLSTATE: `HY000` (`ER_TABLE_NOT_LOCKED_FOR_WRITE`)

  Message: Table '%s' was locked with a READ lock and can't be updated

- Error: `1100` SQLSTATE: `HY000` (`ER_TABLE_NOT_LOCKED`)

  Message: Table '%s' was not locked with LOCK TABLES

- Error: `1101` SQLSTATE: `42000` (`ER_BLOB_CANT_HAVE_DEFAULT`)

  Message: BLOB/TEXT column '%s' can't have a default value

- Error: `1102` SQLSTATE: `42000` (`ER_WRONG_DB_NAME`)

  Message: Incorrect database name '%s'

- Error: `1103` SQLSTATE: `42000` (`ER_WRONG_TABLE_NAME`)

  Message: Incorrect table name '%s'

- Error: `1104` SQLSTATE: `42000` (`ER_TOO_BIG_SELECT`)

  Message: The SELECT would examine more than MAX_JOIN_SIZE rows; check your WHERE and use SET SQL_BIG_SELECTS=1 or SET SQL_MAX_JOIN_SIZE=# if the SELECT is okay

- Error: `1105` SQLSTATE: `HY000` (`ER_UNKNOWN_ERROR`)

  Message: Unknown error

- Error: `1106` SQLSTATE: `42000` (`ER_UNKNOWN_PROCEDURE`)

  Message: Unknown procedure '%s'

- Error: `1107` SQLSTATE: `42000` (`ER_WRONG_PARAMCOUNT_TO_PROCEDURE`)

  Message: Incorrect parameter count to procedure '%s'

- Error: `1108` SQLSTATE: `HY000` (`ER_WRONG_PARAMETERS_TO_PROCEDURE`)

  Message: Incorrect parameters to procedure '%s'

- Error: `1109` SQLSTATE: `42S02` (`ER_UNKNOWN_TABLE`)

  Message: Unknown table '%s' in %s

- Error: `1110` SQLSTATE: `42000` (`ER_FIELD_SPECIFIED_TWICE`)

  Message: Column '%s' specified twice

- Error: `1111` SQLSTATE: `HY000` (`ER_INVALID_GROUP_FUNC_USE`)

  Message: Invalid use of group function

- Error: `1112` SQLSTATE: `42000` (`ER_UNSUPPORTED_EXTENSION`)

  Message: Table '%s' uses an extension that doesn't exist in this MySQL version

- Error: 1113 SQLSTATE: 42000 (ER_TABLE_MUST_HAVE_COLUMNS)

  Message: A table must have at least 1 column

- Error: 1114 SQLSTATE: HY000 (ER_RECORD_FILE_FULL)

  Message: The table '%s' is full

- Error: 1115 SQLSTATE: 42000 (ER_UNKNOWN_CHARACTER_SET)

  Message: Unknown character set: '%s'

- Error: 1116 SQLSTATE: HY000 (ER_TOO_MANY_TABLES)

  Message: Too many tables; MySQL can only use %d tables in a join

- Error: 1117 SQLSTATE: HY000 (ER_TOO_MANY_FIELDS)

  Message: Too many columns

- Error: 1118 SQLSTATE: 42000 (ER_TOO_BIG_ROWSIZE)

  Message: Row size too large. The maximum row size for the used table type, not counting BLOBs, is %ld. You have to change some columns to TEXT or BLOBs

- Error: 1119 SQLSTATE: HY000 (ER_STACK_OVERRUN)

  Message: Thread stack overrun: Used: %ld of a %ld stack. Use 'mysqld -O thread_stack=#' to specify a bigger stack if needed

- Error: 1120 SQLSTATE: 42000 (ER_WRONG_OUTER_JOIN)

  Message: Cross dependency found in OUTER JOIN; examine your ON conditions

- Error: 1121 SQLSTATE: 42000 (ER_NULL_COLUMN_IN_INDEX)

  Message: Column '%s' is used with UNIQUE or INDEX but is not defined as NOT NULL

- Error: 1122 SQLSTATE: HY000 (ER_CANT_FIND_UDF)

  Message: Can't load function '%s'

- Error: 1123 SQLSTATE: HY000 (ER_CANT_INITIALIZE_UDF)

  Message: Can't initialize function '%s'; %s

- Error: 1124 SQLSTATE: HY000 (ER_UDF_NO_PATHS)

  Message: No paths allowed for shared library

- Error: 1125 SQLSTATE: HY000 (ER_UDF_EXISTS)

  Message: Function '%s' already exists

- Error: 1126 SQLSTATE: HY000 (ER_CANT_OPEN_LIBRARY)

  Message: Can't open shared library '%s' (errno: %d %s)

- Error: 1127 SQLSTATE: HY000 (ER_CANT_FIND_DL_ENTRY)

Message: Can't find function '%s' in library'

- Error: `1128` SQLSTATE: `HY000` (`ER_FUNCTION_NOT_DEFINED`)

  Message: Function '%s' is not defined

- Error: `1129` SQLSTATE: `HY000` (`ER_HOST_IS_BLOCKED`)

  Message: Host '%s' is blocked because of many connection errors; unblock with 'mysqladmin flush-hosts'

- Error: `1130` SQLSTATE: `HY000` (`ER_HOST_NOT_PRIVILEGED`)

  Message: Host '%s' is not allowed to connect to this MySQL server

- Error: `1131` SQLSTATE: `42000` (`ER_PASSWORD_ANONYMOUS_USER`)

  Message: You are using MySQL as an anonymous user and anonymous users are not allowed to change passwords

- Error: `1132` SQLSTATE: `42000` (`ER_PASSWORD_NOT_ALLOWED`)

  Message: You must have privileges to update tables in the mysql database to be able to change passwords for others

- Error: `1133` SQLSTATE: `42000` (`ER_PASSWORD_NO_MATCH`)

  Message: Can't find any matching row in the user table

- Error: `1134` SQLSTATE: `HY000` (`ER_UPDATE_INFO`)

  Message: Rows matched: %ld Changed: %ld Warnings: %ld

- Error: `1135` SQLSTATE: `HY000` (`ER_CANT_CREATE_THREAD`)

  Message: Can't create a new thread (errno %d); if you are not out of available memory, you can consult the manual for a possible OS-dependent bug

- Error: `1136` SQLSTATE: `21S01` (`ER_WRONG_VALUE_COUNT_ON_ROW`)

  Message: Column count doesn't match value count at row %ld

- Error: `1137` SQLSTATE: `HY000` (`ER_CANT_REOPEN_TABLE`)

  Message: Can't reopen table: '%s'

- Error: `1138` SQLSTATE: `22004` (`ER_INVALID_USE_OF_NULL`)

  Message: Invalid use of NULL value

- Error: `1139` SQLSTATE: `42000` (`ER_REGEXP_ERROR`)

  Message: Got error '%s' from regexp

- Error: `1140` SQLSTATE: `42000` (`ER_MIX_OF_GROUP_FUNC_AND_FIELDS`)

  Message: Mixing of GROUP columns (MIN(),MAX(),COUNT(),...) with no GROUP columns is illegal if there is no GROUP BY clause

- Error: `1141` SQLSTATE: `42000` (`ER_NONEXISTING_GRANT`)

  Message: There is no such grant defined for user '%s' on host '%s'

- Error: `1142` SQLSTATE: `42000` (`ER_TABLEACCESS_DENIED_ERROR`)

  Message: %s command denied to user '%s'@'%s' for table '%s'

- Error: `1143` SQLSTATE: `42000` (`ER_COLUMNACCESS_DENIED_ERROR`)

  Message: %s command denied to user '%s'@'%s' for column '%s' in table '%s'

- Error: `1144` SQLSTATE: `42000` (`ER_ILLEGAL_GRANT_FOR_TABLE`)

  Message: Illegal GRANT/REVOKE command; please consult the manual to see which privileges can be used

- Error: `1145` SQLSTATE: `42000` (`ER_GRANT_WRONG_HOST_OR_USER`)

  Message: The host or user argument to GRANT is too long

- Error: `1146` SQLSTATE: `42S02` (`ER_NO_SUCH_TABLE`)

  Message: Table '%s.%s' doesn't exist

- Error: `1147` SQLSTATE: `42000` (`ER_NONEXISTING_TABLE_GRANT`)

  Message: There is no such grant defined for user '%s' on host '%s' on table '%s'

- Error: `1148` SQLSTATE: `42000` (`ER_NOT_ALLOWED_COMMAND`)

  Message: The used command is not allowed with this MySQL version

- Error: `1149` SQLSTATE: `42000` (`ER_SYNTAX_ERROR`)

  Message: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use

- Error: `1150` SQLSTATE: `HY000` (`ER_DELAYED_CANT_CHANGE_LOCK`)

  Message: Delayed insert thread couldn't get requested lock for table %s

- Error: `1151` SQLSTATE: `HY000` (`ER_TOO_MANY_DELAYED_THREADS`)

  Message: Too many delayed threads in use

- Error: `1152` SQLSTATE: `08S01` (`ER_ABORTING_CONNECTION`)

  Message: Aborted connection %ld to db: '%s' user: '%s' (%s)

- Error: `1153` SQLSTATE: `08S01` (`ER_NET_PACKET_TOO_LARGE`)

  Message: Got a packet bigger than 'max_allowed_packet' bytes

- Error: `1154` SQLSTATE: `08S01` (`ER_NET_READ_ERROR_FROM_PIPE`)

  Message: Got a read error from the connection pipe

- Error: `1155` SQLSTATE: `08S01` (`ER_NET_FCNTL_ERROR`)

Message: Got an error from fcntl()

- Error: `1156` SQLSTATE: `08S01` (`ER_NET_PACKETS_OUT_OF_ORDER`)

  Message: Got packets out of order

- Error: `1157` SQLSTATE: `08S01` (`ER_NET_UNCOMPRESS_ERROR`)

  Message: Couldn't uncompress communication packet

- Error: `1158` SQLSTATE: `08S01` (`ER_NET_READ_ERROR`)

  Message: Got an error reading communication packets

- Error: `1159` SQLSTATE: `08S01` (`ER_NET_READ_INTERRUPTED`)

  Message: Got timeout reading communication packets

- Error: `1160` SQLSTATE: `08S01` (`ER_NET_ERROR_ON_WRITE`)

  Message: Got an error writing communication packets

- Error: `1161` SQLSTATE: `08S01` (`ER_NET_WRITE_INTERRUPTED`)

  Message: Got timeout writing communication packets

- Error: `1162` SQLSTATE: `42000` (`ER_TOO_LONG_STRING`)

  Message: Result string is longer than 'max_allowed_packet' bytes

- Error: `1163` SQLSTATE: `42000` (`ER_TABLE_CANT_HANDLE_BLOB`)

  Message: The used table type doesn't support BLOB/TEXT columns

- Error: `1164` SQLSTATE: `42000` (`ER_TABLE_CANT_HANDLE_AUTO_INCREMENT`)

  Message: The used table type doesn't support AUTO_INCREMENT columns

- Error: `1165` SQLSTATE: `HY000` (`ER_DELAYED_INSERT_TABLE_LOCKED`)

  Message: INSERT DELAYED can't be used with table '%s' because it is locked with LOCK TABLES

- Error: `1166` SQLSTATE: `42000` (`ER_WRONG_COLUMN_NAME`)

  Message: Incorrect column name '%s'

- Error: `1167` SQLSTATE: `42000` (`ER_WRONG_KEY_COLUMN`)

  Message: The used storage engine can't index column '%s'

- Error: `1168` SQLSTATE: `HY000` (`ER_WRONG_MRG_TABLE`)

  Message: All tables in the MERGE table are not identically defined

- Error: `1169` SQLSTATE: `23000` (`ER_DUP_UNIQUE`)

  Message: Can't write, because of unique constraint, to table '%s'

- Error: `1170` SQLSTATE: `42000` (`ER_BLOB_KEY_WITHOUT_LENGTH`)

  Message: BLOB/TEXT column '%s' used in key specification without a key length

- Error: `1171` SQLSTATE: `42000` (`ER_PRIMARY_CANT_HAVE_NULL`)

  Message: All parts of a PRIMARY KEY must be NOT NULL; if you need NULL in a key, use UNIQUE instead

- Error: `1172` SQLSTATE: `42000` (`ER_TOO_MANY_ROWS`)

  Message: Result consisted of more than one row

- Error: `1173` SQLSTATE: `42000` (`ER_REQUIRES_PRIMARY_KEY`)

  Message: This table type requires a primary key

- Error: `1174` SQLSTATE: `HY000` (`ER_NO_RAID_COMPILED`)

  Message: This version of MySQL is not compiled with RAID support

- Error: `1175` SQLSTATE: `HY000` (`ER_UPDATE_WITHOUT_KEY_IN_SAFE_MODE`)

  Message: You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column

- Error: `1176` SQLSTATE: `HY000` (`ER_KEY_DOES_NOT_EXITS`)

  Message: Key '%s' doesn't exist in table '%s'

- Error: `1177` SQLSTATE: `42000` (`ER_CHECK_NO_SUCH_TABLE`)

  Message: Can't open table

- Error: `1178` SQLSTATE: `42000` (`ER_CHECK_NOT_IMPLEMENTED`)

  Message: The storage engine for the table doesn't support %s

- Error: `1179` SQLSTATE: `25000` (`ER_CANT_DO_THIS_DURING_AN_TRANSACTION`)

  Message: You are not allowed to execute this command in a transaction

- Error: `1180` SQLSTATE: `HY000` (`ER_ERROR_DURING_COMMIT`)

  Message: Got error %d during COMMIT

- Error: `1181` SQLSTATE: `HY000` (`ER_ERROR_DURING_ROLLBACK`)

  Message: Got error %d during ROLLBACK

- Error: `1182` SQLSTATE: `HY000` (`ER_ERROR_DURING_FLUSH_LOGS`)

  Message: Got error %d during FLUSH_LOGS

- Error: `1183` SQLSTATE: `HY000` (`ER_ERROR_DURING_CHECKPOINT`)

  Message: Got error %d during CHECKPOINT

- Error: `1184` SQLSTATE: `08S01` (`ER_NEW_ABORTING_CONNECTION`)

Message: Aborted connection %ld to db: '%s' user: '%s' host: `%s' (%s)

- Error: `1185` SQLSTATE: `HY000` (`ER_DUMP_NOT_IMPLEMENTED`)

  Message: The storage engine for the table does not support binary table dump

- Error: `1186` SQLSTATE: `HY000` (`ER_FLUSH_MASTER_BINLOG_CLOSED`)

  Message: Binlog closed, cannot RESET MASTER

- Error: `1187` SQLSTATE: `HY000` (`ER_INDEX_REBUILD`)

  Message: Failed rebuilding the index of dumped table '%s'

- Error: `1188` SQLSTATE: `HY000` (`ER_MASTER`)

  Message: Error from master: '%s'

- Error: `1189` SQLSTATE: `08S01` (`ER_MASTER_NET_READ`)

  Message: Net error reading from master

- Error: `1190` SQLSTATE: `08S01` (`ER_MASTER_NET_WRITE`)

  Message: Net error writing to master

- Error: `1191` SQLSTATE: `HY000` (`ER_FT_MATCHING_KEY_NOT_FOUND`)

  Message: Can't find FULLTEXT index matching the column list

- Error: `1192` SQLSTATE: `HY000` (`ER_LOCK_OR_ACTIVE_TRANSACTION`)

  Message: Can't execute the given command because you have active locked tables or an active transaction

- Error: `1193` SQLSTATE: `HY000` (`ER_UNKNOWN_SYSTEM_VARIABLE`)

  Message: Unknown system variable '%s'

- Error: `1194` SQLSTATE: `HY000` (`ER_CRASHED_ON_USAGE`)

  Message: Table '%s' is marked as crashed and should be repaired

- Error: `1195` SQLSTATE: `HY000` (`ER_CRASHED_ON_REPAIR`)

  Message: Table '%s' is marked as crashed and last (automatic?) repair failed

- Error: `1196` SQLSTATE: `HY000` (`ER_WARNING_NOT_COMPLETE_ROLLBACK`)

  Message: Some non-transactional changed tables couldn't be rolled back

- Error: `1197` SQLSTATE: `HY000` (`ER_TRANS_CACHE_FULL`)

  Message: Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage; increase this mysqld variable and try again

- Error: `1198` SQLSTATE: `HY000` (`ER_SLAVE_MUST_STOP`)

  Message: This operation cannot be performed with a running slave; run STOP SLAVE first

- Error: `1199` SQLSTATE: `HY000` (`ER_SLAVE_NOT_RUNNING`)

  Message: This operation requires a running slave; configure slave and do START SLAVE

- Error: `1200` SQLSTATE: `HY000` (`ER_BAD_SLAVE`)

  Message: The server is not configured as slave; fix in config file or with CHANGE MASTER TO

- Error: `1201` SQLSTATE: `HY000` (`ER_MASTER_INFO`)

  Message: Could not initialize master info structure; more error messages can be found in the MySQL error log

- Error: `1202` SQLSTATE: `HY000` (`ER_SLAVE_THREAD`)

  Message: Could not create slave thread; check system resources

- Error: `1203` SQLSTATE: `42000` (`ER_TOO_MANY_USER_CONNECTIONS`)

  Message: User %s already has more than 'max_user_connections' active connections

- Error: `1204` SQLSTATE: `HY000` (`ER_SET_CONSTANTS_ONLY`)

  Message: You may only use constant expressions with SET

- Error: `1205` SQLSTATE: `HY000` (`ER_LOCK_WAIT_TIMEOUT`)

  Message: Lock wait timeout exceeded; try restarting transaction

- Error: `1206` SQLSTATE: `HY000` (`ER_LOCK_TABLE_FULL`)

  Message: The total number of locks exceeds the lock table size

- Error: `1207` SQLSTATE: `25000` (`ER_READ_ONLY_TRANSACTION`)

  Message: Update locks cannot be acquired during a READ UNCOMMITTED transaction

- Error: `1208` SQLSTATE: `HY000` (`ER_DROP_DB_WITH_READ_LOCK`)

  Message: DROP DATABASE not allowed while thread is holding global read lock

- Error: `1209` SQLSTATE: `HY000` (`ER_CREATE_DB_WITH_READ_LOCK`)

  Message: CREATE DATABASE not allowed while thread is holding global read lock

- Error: `1210` SQLSTATE: `HY000` (`ER_WRONG_ARGUMENTS`)

  Message: Incorrect arguments to %s

- Error: `1211` SQLSTATE: `42000` (`ER_NO_PERMISSION_TO_CREATE_USER`)

  Message: '%s'@'%s' is not allowed to create new users

- Error: `1212` SQLSTATE: `HY000` (`ER_UNION_TABLES_IN_DIFFERENT_DIR`)

  Message: Incorrect table definition; all MERGE tables must be in the same database

- Error: `1213` SQLSTATE: `40001` (`ER_LOCK_DEADLOCK`)

  Message: Deadlock found when trying to get lock; try restarting transaction

- Error: `1214` SQLSTATE: `HY000` (`ER_TABLE_CANT_HANDLE_FT`)

  Message: The used table type doesn't support FULLTEXT indexes
- Error: `1215` SQLSTATE: `HY000` (`ER_CANNOT_ADD_FOREIGN`)

  Message: Cannot add foreign key constraint
- Error: `1216` SQLSTATE: `23000` (`ER_NO_REFERENCED_ROW`)

  Message: Cannot add or update a child row: a foreign key constraint fails
- Error: `1217` SQLSTATE: `23000` (`ER_ROW_IS_REFERENCED`)

  Message: Cannot delete or update a parent row: a foreign key constraint fails
- Error: `1218` SQLSTATE: `08S01` (`ER_CONNECT_TO_MASTER`)

  Message: Error connecting to master: %s
- Error: `1219` SQLSTATE: `HY000` (`ER_QUERY_ON_MASTER`)

  Message: Error running query on master: %s
- Error: `1220` SQLSTATE: `HY000` (`ER_ERROR_WHEN_EXECUTING_COMMAND`)

  Message: Error when executing command %s: %s
- Error: `1221` SQLSTATE: `HY000` (`ER_WRONG_USAGE`)

  Message: Incorrect usage of %s and %s
- Error: `1222` SQLSTATE: `21000` (`ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT`)

  Message: The used SELECT statements have a different number of columns
- Error: `1223` SQLSTATE: `HY000` (`ER_CANT_UPDATE_WITH_READLOCK`)

  Message: Can't execute the query because you have a conflicting read lock
- Error: `1224` SQLSTATE: `HY000` (`ER_MIXING_NOT_ALLOWED`)

  Message: Mixing of transactional and non-transactional tables is disabled
- Error: `1225` SQLSTATE: `HY000` (`ER_DUP_ARGUMENT`)

  Message: Option '%s' used twice in statement
- Error: `1226` SQLSTATE: `42000` (`ER_USER_LIMIT_REACHED`)

  Message: User '%s' has exceeded the '%s' resource (current value: %ld)
- Error: `1227` SQLSTATE: `HY000` (`ER_SPECIFIC_ACCESS_DENIED_ERROR`)

  Message: Access denied; you need the %s privilege for this operation
- Error: `1228` SQLSTATE: `HY000` (`ER_LOCAL_VARIABLE`)

  Message: Variable '%s' is a SESSION variable and can't be used with SET GLOBAL

- Error: 1229 SQLSTATE: HY000 (ER_GLOBAL_VARIABLE)

  Message: Variable '%s' is a GLOBAL variable and should be set with SET GLOBAL
- Error: 1230 SQLSTATE: 42000 (ER_NO_DEFAULT)

  Message: Variable '%s' doesn't have a default value
- Error: 1231 SQLSTATE: 42000 (ER_WRONG_VALUE_FOR_VAR)

  Message: Variable '%s' can't be set to the value of '%s'
- Error: 1232 SQLSTATE: 42000 (ER_WRONG_TYPE_FOR_VAR)

  Message: Incorrect argument type to variable '%s'
- Error: 1233 SQLSTATE: HY000 (ER_VAR_CANT_BE_READ)

  Message: Variable '%s' can only be set, not read
- Error: 1234 SQLSTATE: 42000 (ER_CANT_USE_OPTION_HERE)

  Message: Incorrect usage/placement of '%s'
- Error: 1235 SQLSTATE: 42000 (ER_NOT_SUPPORTED_YET)

  Message: This version of MySQL doesn't yet support '%s'
- Error: 1236 SQLSTATE: HY000 (ER_MASTER_FATAL_ERROR_READING_BINLOG)

  Message: Got fatal error %d: '%s' from master when reading data from binary log
- Error: 1237 SQLSTATE: HY000 (ER_SLAVE_IGNORED_TABLE)

  Message: Slave SQL thread ignored the query because of replicate-*-table rules
- Error: 1238 SQLSTATE: HY000 (ER_INCORRECT_GLOBAL_LOCAL_VAR)

  Message: Variable '%s' is a %s variable
- Error: 1239 SQLSTATE: 42000 (ER_WRONG_FK_DEF)

  Message: Incorrect foreign key definition for '%s': %s
- Error: 1240 SQLSTATE: HY000 (ER_KEY_REF_DO_NOT_MATCH_TABLE_REF)

  Message: Key reference and table reference don't match
- Error: 1241 SQLSTATE: 21000 (ER_OPERAND_COLUMNS)

  Message: Operand should contain %d column(s)
- Error: 1242 SQLSTATE: 21000 (ER_SUBQUERY_NO_1_ROW)

  Message: Subquery returns more than 1 row
- Error: 1243 SQLSTATE: HY000 (ER_UNKNOWN_STMT_HANDLER)

  Message: Unknown prepared statement handler (%.*s) given to %s

- Error: 1244 SQLSTATE: HY000 (ER_CORRUPT_HELP_DB)

  Message: Help database is corrupt or does not exist

- Error: 1245 SQLSTATE: HY000 (ER_CYCLIC_REFERENCE)

  Message: Cyclic reference on subqueries

- Error: 1246 SQLSTATE: HY000 (ER_AUTO_CONVERT)

  Message: Converting column '%s' from %s to %s

- Error: 1247 SQLSTATE: 42S22 (ER_ILLEGAL_REFERENCE)

  Message: Reference '%s' not supported (%s)

- Error: 1248 SQLSTATE: 42000 (ER_DERIVED_MUST_HAVE_ALIAS)

  Message: Every derived table must have its own alias

- Error: 1249 SQLSTATE: 01000 (ER_SELECT_REDUCED)

  Message: Select %u was reduced during optimization

- Error: 1250 SQLSTATE: 42000 (ER_TABLENAME_NOT_ALLOWED_HERE)

  Message: Table '%s' from one of the SELECTs cannot be used in %s

- Error: 1251 SQLSTATE: 08004 (ER_NOT_SUPPORTED_AUTH_MODE)

  Message: Client does not support authentication protocol requested by server; consider upgrading MySQL client

- Error: 1252 SQLSTATE: 42000 (ER_SPATIAL_CANT_HAVE_NULL)

  Message: All parts of a SPATIAL index must be NOT NULL

- Error: 1253 SQLSTATE: 42000 (ER_COLLATION_CHARSET_MISMATCH)

  Message: COLLATION '%s' is not valid for CHARACTER SET '%s'

- Error: 1254 SQLSTATE: HY000 (ER_SLAVE_WAS_RUNNING)

  Message: Slave is already running

- Error: 1255 SQLSTATE: HY000 (ER_SLAVE_WAS_NOT_RUNNING)

  Message: Slave already has been stopped

- Error: 1256 SQLSTATE: HY000 (ER_TOO_BIG_FOR_UNCOMPRESS)

  Message: Uncompressed data size too large; the maximum size is %d (probably, length of uncompressed data was corrupted)

- Error: 1257 SQLSTATE: HY000 (ER_ZLIB_Z_MEM_ERROR)

  Message: ZLIB: Not enough memory

- Error: 1258 SQLSTATE: HY000 (ER_ZLIB_Z_BUF_ERROR)

Message: ZLIB: Not enough room in the output buffer (probably, length of uncompressed data was corrupted)

- Error: `1259` SQLSTATE: `HY000` (`ER_ZLIB_Z_DATA_ERROR`)

  Message: ZLIB: Input data corrupted

- Error: `1260` SQLSTATE: `HY000` (`ER_CUT_VALUE_GROUP_CONCAT`)

  Message: %d line(s) were cut by GROUP_CONCAT()

- Error: `1261` SQLSTATE: `01000` (`ER_WARN_TOO_FEW_RECORDS`)

  Message: Row %ld doesn't contain data for all columns

- Error: `1262` SQLSTATE: `01000` (`ER_WARN_TOO_MANY_RECORDS`)

  Message: Row %ld was truncated; it contained more data than there were input columns

- Error: `1263` SQLSTATE: `22004` (`ER_WARN_NULL_TO_NOTNULL`)

  Message: Column set to default value; NULL supplied to NOT NULL column '%s' at row %ld

- Error: `1264` SQLSTATE: `22003` (`ER_WARN_DATA_OUT_OF_RANGE`)

  Message: Out of range value adjusted for column '%s' at row %ld

- Error: `1265` SQLSTATE: `01000` (`WARN_DATA_TRUNCATED`)

  Message: Data truncated for column '%s' at row %ld

- Error: `1266` SQLSTATE: `HY000` (`ER_WARN_USING_OTHER_HANDLER`)

  Message: Using storage engine %s for table '%s'

- Error: `1267` SQLSTATE: `HY000` (`ER_CANT_AGGREGATE_2COLLATIONS`)

  Message: Illegal mix of collations (%s,%s) and (%s,%s) for operation '%s'

- Error: `1268` SQLSTATE: `HY000` (`ER_DROP_USER`)

  Message: Cannot drop one or more of the requested users

- Error: `1269` SQLSTATE: `HY000` (`ER_REVOKE_GRANTS`)

  Message: Can't revoke all privileges, grant for one or more of the requested users

- Error: `1270` SQLSTATE: `HY000` (`ER_CANT_AGGREGATE_3COLLATIONS`)

  Message: Illegal mix of collations (%s,%s), (%s,%s), (%s,%s) for operation '%s'

- Error: `1271` SQLSTATE: `HY000` (`ER_CANT_AGGREGATE_NCOLLATIONS`)

  Message: Illegal mix of collations for operation '%s'

- Error: `1272` SQLSTATE: `HY000` (`ER_VARIABLE_IS_NOT_STRUCT`)

  Message: Variable '%s' is not a variable component (can't be used as XXXX.variable_name)

- Error: 1273 SQLSTATE: HY000 (ER_UNKNOWN_COLLATION)

  Message: Unknown collation: '%s'

- Error: 1274 SQLSTATE: HY000 (ER_SLAVE_IGNORED_SSL_PARAMS)

  Message: SSL parameters in CHANGE MASTER are ignored because this MySQL slave was compiled without SSL support; they can be used later if MySQL slave with SSL is started

- Error: 1275 SQLSTATE: HY000 (ER_SERVER_IS_IN_SECURE_AUTH_MODE)

  Message: Server is running in --secure-auth mode, but '%s'@'%s' has a password in the old format; please change the password to the new format

- Error: 1276 SQLSTATE: HY000 (ER_WARN_FIELD_RESOLVED)

  Message: Field or reference '%s%s%s%s%s' of SELECT #%d was resolved in SELECT #%d

- Error: 1277 SQLSTATE: HY000 (ER_BAD_SLAVE_UNTIL_COND)

  Message: Incorrect parameter or combination of parameters for START SLAVE UNTIL

- Error: 1278 SQLSTATE: HY000 (ER_MISSING_SKIP_SLAVE)

  Message: It is recommended to use --skip-slave-start when doing step-by-step replication with START SLAVE UNTIL; otherwise, you will get problems if you get an unexpected slave's mysqld restart

- Error: 1279 SQLSTATE: HY000 (ER_UNTIL_COND_IGNORED)

  Message: SQL thread is not to be started so UNTIL options are ignored

- Error: 1280 SQLSTATE: 42000 (ER_WRONG_NAME_FOR_INDEX)

  Message: Incorrect index name '%s'

- Error: 1281 SQLSTATE: 42000 (ER_WRONG_NAME_FOR_CATALOG)

  Message: Incorrect catalog name '%s'

- Error: 1282 SQLSTATE: HY000 (ER_WARN_QC_RESIZE)

  Message: Query cache failed to set size %lu; new query cache size is %lu

- Error: 1283 SQLSTATE: HY000 (ER_BAD_FT_COLUMN)

  Message: Column '%s' cannot be part of FULLTEXT index

- Error: 1284 SQLSTATE: HY000 (ER_UNKNOWN_KEY_CACHE)

  Message: Unknown key cache '%s'

- Error: 1285 SQLSTATE: HY000 (ER_WARN_HOSTNAME_WONT_WORK)

  Message: MySQL is started in --skip-name-resolve mode; you must restart it without this switch for this grant to work

- Error: 1286 SQLSTATE: 42000 (ER_UNKNOWN_STORAGE_ENGINE)

  Message: Unknown table engine '%s'

- Error: 1287 SQLSTATE: HY000 (ER_WARN_DEPRECATED_SYNTAX)

  Message: '%s' is deprecated; use '%s' instead

- Error: 1288 SQLSTATE: HY000 (ER_NON_UPDATABLE_TABLE)

  Message: The target table %s of the %s is not updatable

- Error: 1289 SQLSTATE: HY000 (ER_FEATURE_DISABLED)

  Message: The '%s' feature is disabled; you need MySQL built with '%s' to have it working

- Error: 1290 SQLSTATE: HY000 (ER_OPTION_PREVENTS_STATEMENT)

  Message: The MySQL server is running with the %s option so it cannot execute this statement

- Error: 1291 SQLSTATE: HY000 (ER_DUPLICATED_VALUE_IN_TYPE)

  Message: Column '%s' has duplicated value '%s' in %s

- Error: 1292 SQLSTATE: 22007 (ER_TRUNCATED_WRONG_VALUE)

  Message: Truncated incorrect %s value: '%s'

- Error: 1293 SQLSTATE: HY000 (ER_TOO_MUCH_AUTO_TIMESTAMP_COLS)

  Message: Incorrect table definition; there can be only one TIMESTAMP column with CURRENT_TIMESTAMP in DEFAULT or ON UPDATE clause

- Error: 1294 SQLSTATE: HY000 (ER_INVALID_ON_UPDATE)

  Message: Invalid ON UPDATE clause for '%s' column

- Error: 1295 SQLSTATE: HY000 (ER_UNSUPPORTED_PS)

  Message: This command is not supported in the prepared statement protocol yet

- Error: 1296 SQLSTATE: HY000 (ER_GET_ERRMSG)

  Message: Got error %d '%s' from %s

- Error: 1297 SQLSTATE: HY000 (ER_GET_TEMPORARY_ERRMSG)

  Message: Got temporary error %d '%s' from %s

- Error: 1298 SQLSTATE: HY000 (ER_UNKNOWN_TIME_ZONE)

  Message: Unknown or incorrect time zone: '%s'

- Error: 1299 SQLSTATE: HY000 (ER_WARN_INVALID_TIMESTAMP)

  Message: Invalid TIMESTAMP value in column '%s' at row %ld

- Error: 1300 SQLSTATE: HY000 (ER_INVALID_CHARACTER_STRING)

  Message: Invalid %s character string: '%s'

- Error: 1301 SQLSTATE: HY000 (ER_WARN_ALLOWED_PACKET_OVERFLOWED)

  Message: Result of %s() was larger than max_allowed_packet (%ld) - truncated

- Error: 1302 SQLSTATE: HY000 (ER_CONFLICTING_DECLARATIONS)

  Message: Conflicting declarations: '%s%s' and '%s%s'

- Error: 1303 SQLSTATE: 2F003 (ER_SP_NO_RECURSIVE_CREATE)

  Message: Can't create a %s from within another stored routine

- Error: 1304 SQLSTATE: 42000 (ER_SP_ALREADY_EXISTS)

  Message: %s %s already exists

- Error: 1305 SQLSTATE: 42000 (ER_SP_DOES_NOT_EXIST)

  Message: %s %s does not exist

- Error: 1306 SQLSTATE: HY000 (ER_SP_DROP_FAILED)

  Message: Failed to DROP %s %s

- Error: 1307 SQLSTATE: HY000 (ER_SP_STORE_FAILED)

  Message: Failed to CREATE %s %s

- Error: 1308 SQLSTATE: 42000 (ER_SP_LILABEL_MISMATCH)

  Message: %s with no matching label: %s

- Error: 1309 SQLSTATE: 42000 (ER_SP_LABEL_REDEFINE)

  Message: Redefining label %s

- Error: 1310 SQLSTATE: 42000 (ER_SP_LABEL_MISMATCH)

  Message: End-label %s without match

- Error: 1311 SQLSTATE: 01000 (ER_SP_UNINIT_VAR)

  Message: Referring to uninitialized variable %s

- Error: 1312 SQLSTATE: 0A000 (ER_SP_BADSELECT)

  Message: SELECT in a stored procedure must have INTO

- Error: 1313 SQLSTATE: 42000 (ER_SP_BADRETURN)

  Message: RETURN is only allowed in a FUNCTION

- Error: 1314 SQLSTATE: 0A000 (ER_SP_BADSTATEMENT)

  Message: LOCK and UNLOCK tables are not allowed in stored procedures

- Error: 1315 SQLSTATE: 42000 (ER_UPDATE_LOG_DEPRECATED_IGNORED)

  Message: The update log is deprecated and replaced by the binary log; SET SQL_LOG_UPDATE has been ignored

- Error: 1316 SQLSTATE: 42000 (ER_UPDATE_LOG_DEPRECATED_TRANSLATED)

  Message: The update log is deprecated and replaced by the binary log; SET SQL_LOG_UPDATE

has been translated to SET SQL_LOG_BIN

- Error: `1317` SQLSTATE: `70100` (`ER_QUERY_INTERRUPTED`)

  Message: Query execution was interrupted
- Error: `1318` SQLSTATE: `42000` (`ER_SP_WRONG_NO_OF_ARGS`)

  Message: Incorrect number of arguments for %s %s; expected %u, got %u
- Error: `1319` SQLSTATE: `42000` (`ER_SP_COND_MISMATCH`)

  Message: Undefined CONDITION: %s
- Error: `1320` SQLSTATE: `42000` (`ER_SP_NORETURN`)

  Message: No RETURN found in FUNCTION %s
- Error: `1321` SQLSTATE: `2F005` (`ER_SP_NORETURNEND`)

  Message: FUNCTION %s ended without RETURN
- Error: `1322` SQLSTATE: `42000` (`ER_SP_BAD_CURSOR_QUERY`)

  Message: Cursor statement must be a SELECT
- Error: `1323` SQLSTATE: `42000` (`ER_SP_BAD_CURSOR_SELECT`)

  Message: Cursor SELECT must not have INTO
- Error: `1324` SQLSTATE: `42000` (`ER_SP_CURSOR_MISMATCH`)

  Message: Undefined CURSOR: %s
- Error: `1325` SQLSTATE: `24000` (`ER_SP_CURSOR_ALREADY_OPEN`)

  Message: Cursor is already open
- Error: `1326` SQLSTATE: `24000` (`ER_SP_CURSOR_NOT_OPEN`)

  Message: Cursor is not open
- Error: `1327` SQLSTATE: `42000` (`ER_SP_UNDECLARED_VAR`)

  Message: Undeclared variable: %s
- Error: `1328` SQLSTATE: `HY000` (`ER_SP_WRONG_NO_OF_FETCH_ARGS`)

  Message: Incorrect number of FETCH variables
- Error: `1329` SQLSTATE: `02000` (`ER_SP_FETCH_NO_DATA`)

  Message: No data to FETCH
- Error: `1330` SQLSTATE: `42000` (`ER_SP_DUP_PARAM`)

  Message: Duplicate parameter: %s
- Error: `1331` SQLSTATE: `42000` (`ER_SP_DUP_VAR`)

Message: Duplicate variable: %s

- Error: 1332 SQLSTATE: 42000 (ER_SP_DUP_COND)

  Message: Duplicate condition: %s

- Error: 1333 SQLSTATE: 42000 (ER_SP_DUP_CURS)

  Message: Duplicate cursor: %s

- Error: 1334 SQLSTATE: HY000 (ER_SP_CANT_ALTER)

  Message: Failed to ALTER %s %s

- Error: 1335 SQLSTATE: 0A000 (ER_SP_SUBSELECT_NYI)

  Message: Subselect value not supported

- Error: 1336 SQLSTATE: 42000 (ER_SP_NO_USE)

  Message: USE is not allowed in a stored procedure

- Error: 1337 SQLSTATE: 42000 (ER_SP_VARCOND_AFTER_CURSHNDLR)

  Message: Variable or condition declaration after cursor or handler declaration

- Error: 1338 SQLSTATE: 42000 (ER_SP_CURSOR_AFTER_HANDLER)

  Message: Cursor declaration after handler declaration

- Error: 1339 SQLSTATE: 20000 (ER_SP_CASE_NOT_FOUND)

  Message: Case not found for CASE statement

- Error: 1340 SQLSTATE: HY000 (ER_FPARSER_TOO_BIG_FILE)

  Message: Configuration file '%s' is too big

- Error: 1341 SQLSTATE: HY000 (ER_FPARSER_BAD_HEADER)

  Message: Malformed file type header in file '%s'

- Error: 1342 SQLSTATE: HY000 (ER_FPARSER_EOF_IN_COMMENT)

  Message: Unexpected end of file while parsing comment '%s'

- Error: 1343 SQLSTATE: HY000 (ER_FPARSER_ERROR_IN_PARAMETER)

  Message: Error while parsing parameter '%s' (line: '%s')

- Error: 1344 SQLSTATE: HY000 (ER_FPARSER_EOF_IN_UNKNOWN_PARAMETER)

  Message: Unexpected end of file while skipping unknown parameter '%s'

- Error: 1345 SQLSTATE: HY000 (ER_VIEW_NO_EXPLAIN)

  Message: EXPLAIN/SHOW can not be issued; lacking privileges for underlying table

- Error: 1346 SQLSTATE: HY000 (ER_FRM_UNKNOWN_TYPE)

Message: File '%s' has unknown type '%s' in its header

- Error: 1347 SQLSTATE: HY000 (ER_WRONG_OBJECT)

  Message: '%s.%s' is not %s

- Error: 1348 SQLSTATE: HY000 (ER_NONUPDATEABLE_COLUMN)

  Message: Column '%s' is not updatable

- Error: 1349 SQLSTATE: HY000 (ER_VIEW_SELECT_DERIVED)

  Message: View's SELECT contains a subquery in the FROM clause

- Error: 1350 SQLSTATE: HY000 (ER_VIEW_SELECT_CLAUSE)

  Message: View's SELECT contains a '%s' clause

- Error: 1351 SQLSTATE: HY000 (ER_VIEW_SELECT_VARIABLE)

  Message: View's SELECT contains a variable or parameter

- Error: 1352 SQLSTATE: HY000 (ER_VIEW_SELECT_TMPTABLE)

  Message: View's SELECT contains a temporary table '%s'

- Error: 1353 SQLSTATE: HY000 (ER_VIEW_WRONG_LIST)

  Message: View's SELECT and view's field list have different column counts

- Error: 1354 SQLSTATE: HY000 (ER_WARN_VIEW_MERGE)

  Message: View merge algorithm can't be used here for now (assumed undefined algorithm)

- Error: 1355 SQLSTATE: HY000 (ER_WARN_VIEW_WITHOUT_KEY)

  Message: View being updated does not have complete key of underlying table in it

- Error: 1356 SQLSTATE: HY000 (ER_VIEW_INVALID)

  Message: View '%s.%s' references invalid table(s) or column(s) or function(s)

- Error: 1357 SQLSTATE: HY000 (ER_SP_NO_DROP_SP)

  Message: Can't drop a %s from within another stored routine

- Error: 1358 SQLSTATE: HY000 (ER_SP_GOTO_IN_HNDLR)

  Message: GOTO is not allowed in a stored procedure handler

- Error: 1359 SQLSTATE: HY000 (ER_TRG_ALREADY_EXISTS)

  Message: Trigger already exists

- Error: 1360 SQLSTATE: HY000 (ER_TRG_DOES_NOT_EXIST)

  Message: Trigger does not exist

- Error: 1361 SQLSTATE: HY000 (ER_TRG_ON_VIEW_OR_TEMP_TABLE)

Message: Trigger's '%s' is view or temporary table

- Error: 1362 SQLSTATE: HY000 (ER_TRG_CANT_CHANGE_ROW)

  Message: Updating of %s row is not allowed in %strigger

- Error: 1363 SQLSTATE: HY000 (ER_TRG_NO_SUCH_ROW_IN_TRG)

  Message: There is no %s row in %s trigger

- Error: 1364 SQLSTATE: HY000 (ER_NO_DEFAULT_FOR_FIELD)

  Message: Field '%s' doesn't have a default value

- Error: 1365 SQLSTATE: 22012 (ER_DIVISION_BY_ZERO)

  Message: Division by 0

- Error: 1366 SQLSTATE: HY000 (ER_TRUNCATED_WRONG_VALUE_FOR_FIELD)

  Message: Incorrect %s value: '%s' for column '%s' at row %ld

- Error: 1367 SQLSTATE: 22007 (ER_ILLEGAL_VALUE_FOR_TYPE)

  Message: Illegal %s '%s' value found during parsing

- Error: 1368 SQLSTATE: HY000 (ER_VIEW_NONUPD_CHECK)

  Message: CHECK OPTION on non-updatable view '%s.%s'

- Error: 1369 SQLSTATE: HY000 (ER_VIEW_CHECK_FAILED)

  Message: CHECK OPTION failed '%s.%s'

- Error: 1370 SQLSTATE: 42000 (ER_PROCACCESS_DENIED_ERROR)

  Message: %s command denied to user '%s'@'%s' for routine '%s'

- Error: 1371 SQLSTATE: HY000 (ER_RELAY_LOG_FAIL)

  Message: Failed purging old relay logs: %s

- Error: 1372 SQLSTATE: HY000 (ER_PASSWD_LENGTH)

  Message: Password hash should be a %d-digit hexadecimal number

- Error: 1373 SQLSTATE: HY000 (ER_UNKNOWN_TARGET_BINLOG)

  Message: Target log not found in binlog index

- Error: 1374 SQLSTATE: HY000 (ER_IO_ERR_LOG_INDEX_READ)

  Message: I/O error reading log index file

- Error: 1375 SQLSTATE: HY000 (ER_BINLOG_PURGE_PROHIBITED)

  Message: Server configuration does not permit binlog purge

- Error: 1376 SQLSTATE: HY000 (ER_FSEEK_FAIL)

Message: Failed on fseek()

- Error: 1377 SQLSTATE: HY000 (ER_BINLOG_PURGE_FATAL_ERR)

  Message: Fatal error during log purge

- Error: 1378 SQLSTATE: HY000 (ER_LOG_IN_USE)

  Message: A purgeable log is in use, will not purge

- Error: 1379 SQLSTATE: HY000 (ER_LOG_PURGE_UNKNOWN_ERR)

  Message: Unknown error during log purge

- Error: 1380 SQLSTATE: HY000 (ER_RELAY_LOG_INIT)

  Message: Failed initializing relay log position: %s

- Error: 1381 SQLSTATE: HY000 (ER_NO_BINARY_LOGGING)

  Message: You are not using binary logging

- Error: 1382 SQLSTATE: HY000 (ER_RESERVED_SYNTAX)

  Message: The '%s' syntax is reserved for purposes internal to the MySQL server

- Error: 1383 SQLSTATE: HY000 (ER_WSAS_FAILED)

  Message: WSAStartup Failed

- Error: 1384 SQLSTATE: HY000 (ER_DIFF_GROUPS_PROC)

  Message: Can't handle procedures with differents groups yet

- Error: 1385 SQLSTATE: HY000 (ER_NO_GROUP_FOR_PROC)

  Message: Select must have a group with this procedure

- Error: 1386 SQLSTATE: HY000 (ER_ORDER_WITH_PROC)

  Message: Can't use ORDER clause with this procedure

- Error: 1387 SQLSTATE: HY000 (ER_LOGGING_PROHIBIT_CHANGING_OF)

  Message: Binary logging and replication forbid changing the global server %s

- Error: 1388 SQLSTATE: HY000 (ER_NO_FILE_MAPPING)

  Message: Can't map file: %s, errno: %d

- Error: 1389 SQLSTATE: HY000 (ER_WRONG_MAGIC)

  Message: Wrong magic in %s

- Error: 1390 SQLSTATE: HY000 (ER_PS_MANY_PARAM)

  Message: Prepared statement contains too many placeholders

- Error: 1391 SQLSTATE: HY000 (ER_KEY_PART_0)

Message: Key part '%s' length cannot be 0

- Error: 1392 SQLSTATE: HY000 (ER_VIEW_CHECKSUM)

  Message: View text checksum failed

- Error: 1393 SQLSTATE: HY000 (ER_VIEW_MULTIUPDATE)

  Message: Can not modify more than one base table through a join view '%s.%s'

- Error: 1394 SQLSTATE: HY000 (ER_VIEW_NO_INSERT_FIELD_LIST)

  Message: Can not insert into join view '%s.%s' without fields list

- Error: 1395 SQLSTATE: HY000 (ER_VIEW_DELETE_MERGE_VIEW)

  Message: Can not delete from join view '%s.%s'

- Error: 1396 SQLSTATE: HY000 (ER_CANNOT_USER)

  Message: Operation %s failed for %s

- Error: 1397 SQLSTATE: XAE04 (ER_XAER_NOTA)

  Message: XAER_NOTA: Unknown XID

- Error: 1398 SQLSTATE: XAE05 (ER_XAER_INVAL)

  Message: XAER_INVAL: Invalid arguments (or unsupported command)

- Error: 1399 SQLSTATE: XAE07 (ER_XAER_RMFAIL)

  Message: XAER_RMFAIL: The command cannot be executed in the %s state

- Error: 1400 SQLSTATE: XAE09 (ER_XAER_OUTSIDE)

  Message: XAER_OUTSIDE: Some work is done outside global transaction

- Error: 1401 SQLSTATE: XAE03 (ER_XAER_RMERR)

  Message: XAER_RMERR: Fatal error occurred in the transaction branch - check your data for consistency

- Error: 1402 SQLSTATE: XA100 (ER_XA_RBROLLBACK)

  Message: XA_RBROLLBACK: Transaction branch was rolled back

- Error: 1403 SQLSTATE: 42000 (ER_NONEXISTING_PROC_GRANT)

  Message: There is no such grant defined for user '%s' on host '%s' on routine '%s'

- Error: 1404 SQLSTATE: HY000 (ER_PROC_AUTO_GRANT_FAIL)

  Message: Failed to grant EXECUTE and ALTER ROUTINE privileges

- Error: 1405 SQLSTATE: HY000 (ER_PROC_AUTO_REVOKE_FAIL)

  Message: Failed to revoke all privileges to dropped routine

- Error: `1406` SQLSTATE: `22001` (`ER_DATA_TOO_LONG`)

  Message: Data too long for column '%s' at row %ld

Client error information comes from the following files:

- The Error values and the symbols in parentheses correspond to definitions in the `include/errmsg.h` MySQL source file.
- The Message values correspond to the error messages that are listed in the `libmysql/errmsg.c` file. %d or %s represent numbers or strings that are substituted into the messages %when they are displayed.

Because updates are frequent, it is possible that these files contain additional error information not listed here.

- Error: `2000` (`CR_UNKNOWN_ERROR`)

  Message: Unknown MySQL error
- Error: `2001` (`CR_SOCKET_CREATE_ERROR`)

  Message: Can't create UNIX socket (%d)
- Error: `2002` (`CR_CONNECTION_ERROR`)

  Message: Can't connect to local MySQL server through socket '%s' (%d)
- Error: `2003` (`CR_CONN_HOST_ERROR`)

  Message: Can't connect to MySQL server on '%s' (%d)
- Error: `2004` (`CR_IPSOCK_ERROR`)

  Message: Can't create TCP/IP socket (%d)
- Error: `2005` (`CR_UNKNOWN_HOST`)

  Message: Unknown MySQL server host '%s' (%d)
- Error: `2006` (`CR_SERVER_GONE_ERROR`)

  Message: MySQL server has gone away
- Error: `2007` (`CR_VERSION_ERROR`)

  Message: Protocol mismatch; server version = %d, client version = %d
- Error: `2008` (`CR_OUT_OF_MEMORY`)

  Message: MySQL client ran out of memory
- Error: `2009` (`CR_WRONG_HOST_INFO`)

  Message: Wrong host info

- Error: `2010` (`CR_LOCALHOST_CONNECTION`)

  Message: Localhost via UNIX socket
- Error: `2011` (`CR_TCP_CONNECTION`)

  Message: %s via TCP/IP
- Error: `2012` (`CR_SERVER_HANDSHAKE_ERR`)

  Message: Error in server handshake
- Error: `2013` (`CR_SERVER_LOST`)

  Message: Lost connection to MySQL server during query
- Error: `2014` (`CR_COMMANDS_OUT_OF_SYNC`)

  Message: Commands out of sync; you can't run this command now
- Error: `2015` (`CR_NAMEDPIPE_CONNECTION`)

  Message: Named pipe: %s
- Error: `2016` (`CR_NAMEDPIPEWAIT_ERROR`)

  Message: Can't wait for named pipe to host: %s pipe: %s (%lu)
- Error: `2017` (`CR_NAMEDPIPEOPEN_ERROR`)

  Message: Can't open named pipe to host: %s pipe: %s (%lu)
- Error: `2018` (`CR_NAMEDPIPESETSTATE_ERROR`)

  Message: Can't set state of named pipe to host: %s pipe: %s (%lu)
- Error: `2019` (`CR_CANT_READ_CHARSET`)

  Message: Can't initialize character set %s (path: %s)
- Error: `2020` (`CR_NET_PACKET_TOO_LARGE`)

  Message: Got packet bigger than 'max_allowed_packet' bytes
- Error: `2021` (`CR_EMBEDDED_CONNECTION`)

  Message: Embedded server
- Error: `2022` (`CR_PROBE_SLAVE_STATUS`)

  Message: Error on SHOW SLAVE STATUS:
- Error: `2023` (`CR_PROBE_SLAVE_HOSTS`)

  Message: Error on SHOW SLAVE HOSTS:
- Error: `2024` (`CR_PROBE_SLAVE_CONNECT`)

  Message: Error connecting to slave:

- Error: `2025` (`CR_PROBE_MASTER_CONNECT`)

  Message: Error connecting to master:
- Error: `2026` (`CR_SSL_CONNECTION_ERROR`)

  Message: SSL connection error
- Error: `2027` (`CR_MALFORMED_PACKET`)

  Message: Malformed packet
- Error: `2028` (`CR_WRONG_LICENSE`)

  Message: This client library is licensed only for use with MySQL servers having '%s' license
- Error: `2029` (`CR_NULL_POINTER`)

  Message: Invalid use of null pointer
- Error: `2030` (`CR_NO_PREPARE_STMT`)

  Message: Statement not prepared
- Error: `2031` (`CR_PARAMS_NOT_BOUND`)

  Message: No data supplied for parameters in prepared statement
- Error: `2032` (`CR_DATA_TRUNCATED`)

  Message: Data truncated
- Error: `2033` (`CR_NO_PARAMETERS_EXISTS`)

  Message: No parameters exist in the statement
- Error: `2034` (`CR_INVALID_PARAMETER_NO`)

  Message: Invalid parameter number
- Error: `2035` (`CR_INVALID_BUFFER_USE`)

  Message: Can't send long data for non-string/non-binary data types (parameter: %d)
- Error: `2036` (`CR_UNSUPPORTED_PARAM_TYPE`)

  Message: Using unsupported buffer type: %d (parameter: %d)
- Error: `2037` (`CR_SHARED_MEMORY_CONNECTION`)

  Message: Shared memory: %s
- Error: `2038` (`CR_SHARED_MEMORY_CONNECT_REQUEST_ERROR`)

  Message: Can't open shared memory; client could not create request event (%lu)
- Error: `2039` (`CR_SHARED_MEMORY_CONNECT_ANSWER_ERROR`)

  Message: Can't open shared memory; no answer event received from server (%lu)

- Error: `2040` (`CR_SHARED_MEMORY_CONNECT_FILE_MAP_ERROR`)

  Message: Can't open shared memory; server could not allocate file mapping (%lu)

- Error: `2041` (`CR_SHARED_MEMORY_CONNECT_MAP_ERROR`)

  Message: Can't open shared memory; server could not get pointer to file mapping (%lu)

- Error: `2042` (`CR_SHARED_MEMORY_FILE_MAP_ERROR`)

  Message: Can't open shared memory; client could not allocate file mapping (%lu)

- Error: `2043` (`CR_SHARED_MEMORY_MAP_ERROR`)

  Message: Can't open shared memory; client could not get pointer to file mapping (%lu)

- Error: `2044` (`CR_SHARED_MEMORY_EVENT_ERROR`)

  Message: Can't open shared memory; client could not create %s event (%lu)

- Error: `2045` (`CR_SHARED_MEMORY_CONNECT_ABANDONED_ERROR`)

  Message: Can't open shared memory; no answer from server (%lu)

- Error: `2046` (`CR_SHARED_MEMORY_CONNECT_SET_ERROR`)

  Message: Can't open shared memory; cannot send request event to server (%lu)

- Error: `2047` (`CR_CONN_UNKNOW_PROTOCOL`)

  Message: Wrong or unknown protocol

- Error: `2048` (`CR_INVALID_CONN_HANDLE`)

  Message: Invalid connection handle

- Error: `2049` (`CR_SECURE_AUTH`)

  Message: Connection using old (pre-4.1.1) authentication protocol refused (client option 'secure_auth' enabled)

- Error: `2050` (`CR_FETCH_CANCELED`)

  Message: Row retrieval was canceled by mysql_stmt_close() call

- Error: `2051` (`CR_NO_DATA`)

  Message: Attempt to read column without prior row fetch

- Error: `2052` (`CR_NO_STMT_METADATA`)

  Message: Prepared statement contains no metadata

Chapter 25. Extending MySQL

25.1. MySQL Internals

This chapter describes a lot of things that you need to know when working on the MySQL code. If you plan to contribute to MySQL development, want to have access to the bleeding-edge in-between versions code, or just want to keep track of development, follow the instructions in Section 2.8.3, "Installing from the Development Source Tree". If you are interested in MySQL internals, you should also subscribe to our `internals` mailing list. This list is relatively low traffic. For details on how to subscribe, please see Section 1.4.1.1, "The MySQL Mailing Lists". All developers at MySQL AB are on the `internals` list and we help other people who are working on the MySQL code. Feel free to use this list both to ask questions about the code and to send patches that you would like to contribute to the MySQL project!

25.1.1. MySQL Threads

The MySQL server creates the following threads:

- The TCP/IP connection thread handles all connection requests and creates a new dedicated thread to handle the authentication and SQL query processing for each connection.
- On Windows NT there is a named pipe handler thread that does the same work as the TCP/IP connection thread on named pipe connect requests.
- The signal thread handles all signals. This thread also normally handles alarms and calls `process_alarm()` to force timeouts on connections that have been idle too long.
- If **mysqld** is compiled with `-DUSE_ALARM_THREAD`, a dedicated thread that handles alarms is created. This is only used on some systems where there are problems with `sigwait()` or if you want to use the `thr_alarm()` code in your application without a dedicated signal handling thread.
- If one uses the `--flush_time=#` option, a dedicated thread is created to flush all tables at the given interval.
- Every connection has its own thread.
- Every different table on which one uses `INSERT DELAYED` gets its own thread.
- If you use `--master-host`, a slave replication thread is started to read and apply updates from the master.

mysqladmin processlist only shows the connection, `INSERT DELAYED`, and replication threads.

25.1.2. MySQL Test Suite

Until recently, our main full-coverage test suite was based on proprietary customer data and for that reason has not been publicly available. The only publicly available part of our testing process consisted of the `crash-me` test, a Perl DBI/DBD benchmark found in the `sql-bench` directory, and miscellaneous tests located in `tests` directory. The lack of a standardized publicly available test suite has made it difficult for our users, as well developers, to do regression tests on the MySQL code. To address this problem, we have created a new test system that is included in Unix source distributions and binary distributions starting with Version 3.23.29. The tests can be run under Unix, or on Windows in the Cygwin environment if the server has been compiled under Cygwin. They cannot be run in a native Win-

dows environment currently.

The current set of test cases doesn't test everything in MySQL, but it should catch most obvious bugs in the SQL processing code, OS/library issues, and is quite thorough in testing replication. Our eventual goal is to have the tests cover 100% of the code. We welcome contributions to our test suite. You may especially want to contribute tests that examine the functionality critical to your system, because this ensures that all future MySQL releases work well with your applications.

25.1.2.1. Running the MySQL Test Suite

The test system consist of a test language interpreter (**mysqltest**), a shell script to run all tests(**mysql-test-run**), the actual test cases written in a special test language, and their expected results. To run the test suite on your system after a build, type **make test** or `mysql-test/mysql-test-run` from the source root. If you have installed a binary distribution, `cd` to the install root (eg. `/usr/local/mysql`), and do `scripts/mysql-test-run`. All tests should succeed. If not, you should try to find out why and report the problem if this is a bug in MySQL. See Section 25.1.2.3, “Reporting Bugs in the MySQL Test Suite”.

From MySQL 4.1 on, if you have a copy of **mysqld** running on the machine where you want to run the test suite you do not have to stop it, as long as it is not using ports `9306` and `9307`. If one of those ports is taken, you should edit **mysql-test-run** and change the values of the master and/or slave port to one that is available.

Before MySQL 4.1, **mysql-test-run** does not try to run its own server by default but tries to use your currently running server. To override this and cause **mysql-test-run** to start its own server, run it with the `--local` option.

You can run one individual test case with `mysql-test/mysql-test-run test_name`.

If one test fails, you should test running **mysql-test-run** with the `--force` option to check whether any other tests fail.

25.1.2.2. Extending the MySQL Test Suite

You can use the **mysqltest** language to write your own test cases. Unfortunately, we have not yet written full documentation for it. You can, however, look at our current test cases and use them as an example. The following points should help you get started:

- The tests are located in `mysql-test/t/*.test`
- A test case consists of `;` terminated statements and is similar to the input of **mysql** command-line client. A statement by default is a query to be sent to MySQL server, unless it is recognized as internal command (eg. **sleep**).
- All queries that produce results---for example, `SELECT`, `SHOW`, `EXPLAIN`, etc., must be preceded with *`@/path/to/result/file`*. The file must contain the expected results. An easy way to generate the result file is to run **mysqltest -r < t/test-case-name.test** from the `mysql-test` directory, and then edit the generated result files, if needed, to adjust them to the expected output. In that case, be very careful about not adding or deleting any invisible characters --- make sure to only change the text and/or delete lines. If you have to insert a line, make sure that the fields are separated by a hard tab, and that there is a hard tab at the end. You may want to use **od -c** to make sure that your text editor has not messed anything up during edit. We hope that you never have to edit the output of **mysqltest -r** as you only have to do it when you find a bug.
- To be consistent with our setup, you should put your result files in the `mysql-test/r` directory and name them `test_name.result`. If the test produces more than one result, you should use `test_name.a.result`, `test_name.b.result`, etc.

- If a statement returns an error, you should specify it with `--error error-number` on the line before the statement. The error number can be a list of possible error numbers separated by `','`.
- If you are writing a replication test case, you should on the first line of the test file, put `source include/master-slave.inc;`. To switch between master and slave, use `connection master;` and `connection slave;`. If you need to do something on an alternate connection, you can do `connection master1;` for the master, and `connection slave1;` for the slave.
- If you need to do something in a loop, you can use something like this:

  ```
  let $1=1000;
  while ($1)
  {
   # do your queries here
   dec $1;
  }
  ```

- To sleep between queries, use the **sleep** command. It supports fractions of a second, so you can use **sleep 1.3;**, for example, to sleep 1.3 seconds.
- To run the slave with additional options for your test case, put them in the command-line format in `mysql-test/t/test_name-slave.opt`. For the master, put them in `mysql-test/t/test_name-master.opt`.
- If you have a question about the test suite, or have a test case to contribute, send an email message to the MySQL `internals` mailing list. See Section 1.4.1.1, “The MySQL Mailing Lists”. As this list does not accept attachments, you should ftp all the relevant files to: ftp://ftp.mysql.com/pub/mysql/upload/

25.1.2.3. Reporting Bugs in the MySQL Test Suite

If your MySQL version doesn't pass the test suite you should do the following:

- Don't send a bug report before you have found out as much as possible of what when wrong! When you do it, please use the **mysqlbug** script so that we can get information about your system and MySQL version. See Section 1.4.1.3, “How to Report Bugs or Problems”.
- Make sure to include the output of **mysql-test-run**, as well as contents of all `.reject` files in `mysql-test/r` directory.
- If a test in the test suite fails, check whether the test fails also when run by its own:

  ```
  cd mysql-test
  mysql-test-run --local test-name
  ```

  If this fails, then you should configure MySQL with `--with-debug` and run **mysql-test-run** with the `--debug` option. If this also fails send the trace file `var/tmp/master.trace` to ftp://ftp.mysql.com/pub/mysql/upload/ so that we can examine it. Please remember to also include a full description of your system, the version of the **mysqld** binary and how you compiled it.
- Try also to run **mysql-test-run** with the `--force` option to see whether there is any other test that fails.
- If you have compiled MySQL yourself, check our manual for how to compile MySQL on your platform or, preferable, use one of the binaries we have compiled for you at ht-

tp://dev.mysql.com/downloads/. All our standard binaries should pass the test suite!

- If you get an error such as `Result length mismatch` or `Result content mismatch` it means that the output of the test didn't match exactly the expected output. This could be a bug in MySQL or that your version of **mysqld** produces slightly different results under some circumstances.

  Failed test results are put in a file with the same base name as the result file with the `.reject` extension. If your test case is failing, you should do a diff on the two files. If you cannot see how they are different, examine both with `od -c` and also check their lengths.

- If a test fails totally, you should check the logs file in the `mysql-test/var/log` directory for hints of what went wrong.

- If you have compiled MySQL with debugging you can try to debug this by running **mysql-test-run** with the `--gdb` and/or `--debug` options. See Section E.1.2, “Creating Trace Files”.

  If you have not compiled MySQL for debugging you should probably do that. Just specify the `--with-debug` options to **configure**. See Section 2.8, “MySQL Installation Using a Source Distribution”.

25.2. Adding New Functions to MySQL

There are two ways to add new functions to MySQL:

- You can add functions through the user-defined function (UDF) interface. User-defined functions are compiled as object files and then added to and removed from the server dynamically using the `CREATE FUNCTION` and `DROP FUNCTION` statements. See Section 25.2.2, “`CREATE FUNCTION/DROP FUNCTION` Syntax”.

- You can add functions as native (built-in) MySQL functions. Native functions are compiled into the **mysqld** server and become available on a permanent basis.

Each method has advantages and disadvantages:

- If you write user-defined functions, you must install object files in addition to the server itself. If you compile your function into the server, you don't need to do that.

- You can add UDFs to a binary MySQL distribution. Native functions require you to modify a source distribution.

- If you upgrade your MySQL distribution, you can continue to use your previously installed UDFs, unless you upgrade to a newer version for which the UDF interface changes. (An incompatible change occurred in MySQL 4.1.1 for aggregate functions. A function named `xxx_clear()` must be defined rather than `xxx_reset()`.) For native functions, you must repeat your modifications each time you upgrade.

Whichever method you use to add new functions, they can be invoked in SQL statements just like native functions such as `ABS()` or `SOUNDEX()`.

Another way to add functions is by creating stored functions. These are written using SQL statements rather than by compiling object code. The syntax for writing stored functions is described in `Stored Procedures`.

The following sections describe features of the UDF interface, provide instructions for writing UDFs, and discuss security precautions that MySQL takes to prevent UDF misuse.

For example source code that illustrates how to write UDFs, take a look at the `sql/udf_example.cc` file that is provided in MySQL source distributions.

25.2.1. Features of the User-Defined Function Interface

The MySQL interface for user-defined functions provides the following features and capabilties:

- Functions can return string, integer, or real values.
- You can define simple functions that operate on a single row at a time, or aggregate functions that operate on groups of rows.
- Information is provided to functions that enables them to check the number and types of the arguments passed to them.
- You can tell MySQL to coerce arguments to a given type before passing them to a function.
- You can indicate that a function returns `NULL` or that an error occurred.

25.2.2. `CREATE FUNCTION/DROP FUNCTION` Syntax

```
CREATE [AGGREGATE] FUNCTION function_name RETURNS {STRING|INTEGER|REAL}
       SONAME shared_library_name

DROP FUNCTION function_name
```

A user-defined function (UDF) is a way to extend MySQL with a new function that works like a native (built-in) MySQL function such as `ABS()` or `CONCAT()`.

`function_name` is the name that should be used in SQL statements to invoke the function. The `RETURNS` clause indicates the type of the function's return value. *`shared_library_name`* is the basename of the shared object file that contains the code that implements the function. The file must be located in a directory that is searched by your system's dynamic linker.

To create a function, you must have the `INSERT` and privilege for the `mysql` database. To drop a function, you must have the `DELETE` privilege for the `mysql` database. This is because `CREATE FUNCTION` adds a row to the `mysql.func` system table that records the function's name, type, and shared library name, and `DROP FUNCTION` deletes the function's row from that table. If you do not have this table, you should run the **mysql_fix_privilege_tables** script to create it. See Section 2.10.7, “Upgrading the Grant Tables”.

An active function is one that has been loaded with `CREATE FUNCTION` and not removed with `DROP FUNCTION`. All active functions are reloaded each time the server starts, unless you start **mysqld** with the `--skip-grant-tables` option. In this case, UDF initialization is skipped and UDFs are unavailable.

For instructions on writing user-defined functions, see Section 25.2.3, “Adding a New User-defined Function”. For the UDF mechanism to work, functions must be written in C or C++, your operating system must support dynamic loading and you must have compiled **mysqld** dynamically (not statically).

`AGGREGATE` is a new option for MySQL 3.23. An `AGGREGATE` function works exactly like a native MySQL aggregate (summary) function such as `SUM` or `COUNT()`. For `AGGREGATE` to work, your

`mysql.func` table must contain a `type` column. If your `mysql.func` table does not have this column, you should run the **mysql_fix_privilege_tables** script to create it.

25.2.3. Adding a New User-defined Function

For the UDF mechanism to work, functions must be written in C or C++ and your operating system must support dynamic loading. The MySQL source distribution includes a file `sql/udf_example.cc` that defines 5 new functions. Consult this file to see how UDF calling conventions work.

To be able to use UDFs, you need to link **mysqld** dynamically. Don't configure MySQL using `--with-mysqld-ldflags=-all-static`. If you want to use a UDF that needs to access symbols from **mysqld** (for example, the `metaphone` function in `sql/udf_example.cc` that uses `default_charset_info`), you must link the program with `-rdynamic` (see `man dlopen`). If you plan to use UDFs, the rule of thumb is to configure configure MySQL with `--with-mysqld-ldflags=-rdynamic` unless you have a very good reason not to.

If you to use a precompiled distribution of MySQL, use MySQL-Max, which contains a dynamically linked server that supports dynamic loading.

For each function that you want to use in SQL statements, you should define corresponding C (or C++) functions. In the following discussion, the name ``xxx'' is used for an example function name. To distinguish between SQL and C/C++ usage, `XXX()` (uppercase) indicates an SQL function call, and `xxx()` (lowercase) indicates a C/C++ function call.

The C/C++ functions that you write to implement the interface for `XXX()` are:

- `xxx()` (required)

  The main function. This is where the function result is computed. The correspondence between the SQL function data type and the return type of your C/C++ function is shown here:

SQL Type	C/C++ Type
`STRING`	`char *`
`INTEGER`	`long long`
`REAL`	`double`

- `xxx_init()` (optional)

  The initialization function for `xxx()`. It can be used to:

  - Check the number of arguments to `XXX()`.
  - Check that the arguments are of a required type or, alternatively, tell MySQL to coerce arguments to the types you want when the main function is called.
  - Allocate any memory required by the main function.
  - Specify the maximum length of the result.
  - Specify (for `REAL` functions) the maximum number of decimals.
  - Specify whether the result can be `NULL`.

- `xxx_deinit()` (optional)

The deinitialization function for `xxx()`. It should deallocate any memory allocated by the initialization function.

When an SQL statement invokes `XXX()`, MySQL calls the initialization function `xxx_init()` to let it perform any required setup, such as argument checking or memory allocation. If `xxx_init()` returns an error, the SQL statement is aborted with an error message and the main and deinitialization functions are not called. Otherwise, the main function `xxx()` is called once for each row. After all rows have been processed, the deinitialization function `xxx_deinit()` is called so it can perform any required cleanup.

For aggregate functions that work like `SUM()`, you must also provide the following functions:

- `xxx_reset()` (required before 4.1.1)

  Reset the current aggregate value and insert the argument as the initial aggregate value for a new group.

- `xxx_clear()` (required starting from 4.1.1)

  Reset the current aggregate value but do not insert the argument as the initial aggregate value for a new group.

- `xxx_add()` (required)

  Add the argument to the current aggregate value.

MySQL handles aggregate UDFs as follows:

1. Call `xxx_init()` to let the aggregate function allocate any memory it needs for storing results.
2. Sort the table according to the `GROUP BY` expression.
3. Call `xxx_clear()` for the first row in each new group.
4. Call `xxx_add()` for each new row that belongs in the same group.
5. Call `xxx()` to get the result for the aggregate when the group changes or after the last row has been processed.
6. Repeat 3-5 until all rows has been processed
7. Call `xxx_deinit()` to let the UDF free any memory it has allocated.

All functions must be thread-safe. This includes not just the main function, but the initialization and deinitialization functions as well, and also the additional functions required by aggregate functions. A consequence of this requirement is that you are not allowed to allocate any global or static variables that change! If you need memory, you should allocate it in `xxx_init()` and free it in `xxx_deinit()`.

25.2.3.1. UDF Calling Sequences for simple functions

This section describes the different functions that you need to define when you create a simple UDF. Section 25.2.3, “Adding a New User-defined Function” describes the order in which MySQL calls these functions.

The main `xxx()` function should be declared as shown in this section. Note that the return type and parameters differ, depending on whether you declare the SQL function `XXX()` to return `STRING`, `INTEGER`, or `REAL` in the `CREATE FUNCTION` statement:

For `STRING` functions:

```
char *xxx(UDF_INIT *initid, UDF_ARGS *args,
          char *result, unsigned long *length,
          char *is_null, char *error);
```

For `INTEGER` functions:

```
long long xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

For `REAL` functions:

```
double xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

The initialization and deinitialization functions are declared like this:

```
my_bool xxx_init(UDF_INIT *initid, UDF_ARGS *args, char *message);

void xxx_deinit(UDF_INIT *initid);
```

The `initid` parameter is passed to all three functions. It points to a `UDF_INIT` structure that is used to communicate information between functions. The `UDF_INIT` structure members follow. The initialization function should fill in any members that it wishes to change. (To use the default for a member, leave it unchanged.)

- `my_bool maybe_null`

  `xxx_init()` should set `maybe_null` to `1` if `xxx()` can return `NULL`. The default value is `1` if any of the arguments are declared `maybe_null`.

- `unsigned int decimals`

  The number of decimals. The default value is the maximum number of decimals in the arguments passed to the main function. (For example, if the function is passed `1.34`, `1.345`, and `1.3`, the default would be 3, because `1.345` has 3 decimals.

- `unsigned int max_length`

  The maximum length of the result. The default `max_length` value differs depending on the result type of the function. For string functions, the default is the length of the longest argument. For integer functions, the default is 21 digits. For real functions, the default is 13 plus the number of decimals indicated by `initid->decimals`. (For numeric functions, the length includes any sign or decimal point characters.)

  If you want to return a blob value, you can set `max_length` to 65KB or 16MB. This memory is not allocated, but the value is used to decide which column type to use if there is a need to temporarily store the data.

- `char *ptr`

A pointer that the function can use for its own purposes. For example, functions can use `initid->ptr` to communicate allocated memory among themselves. `xxx_init()` should allocate the memory and assign it to this pointer:

```
initid->ptr = allocated_memory;
```

In `xxx()` and `xxx_deinit()`, refer to `initid->ptr` to use or deallocate the memory.

25.2.3.2. UDF Calling Sequences for aggregate functions

This section describes the different functions that you need to define when you create an aggregate UDF. Section 25.2.3, "Adding a New User-defined Function" describes the order in which MySQL calls these functions.

- `xxx_reset()`

  This function is called when MySQL finds the first row in a new group. It should reset any internal summary variables and then use the given `UDF_ARGS` argument as the first value in your internal summary value for the group. Declare `xxx_reset()` as follows:

  ```
  char *xxx_reset(UDF_INIT *initid, UDF_ARGS *args,
                  char *is_null, char *error);
  ```

  `xxx_reset()` is needed only before MySQL 4.1.1. It is *not* needed or used as of MySQL 4.1.1, when the UDF interface changed to use `xxx_clear()` instead. However, you can define both `xxx_reset()` and `xxx_clear()` if you want to have your UDF work both before and after the interface change. (If you do include both functions, the `xxx_reset()` function in many cases can be implemented internally by calling `xxx_clear()` to reset all variables, and then calling `xxx_add()` to add the `UDF_ARGS` argument as the first value in the group.)

- `xxx_clear()`

  This function is called when MySQL needs to reset the summary results. It is called at the beginning for each new group but can also be called to reset the values for a query where there were no matching rows. Declare `xxx_clear()` as follows:

  ```
  char *xxx_clear(UDF_INIT *initid, char *is_null, char *error);
  ```

  `is_null` is set to point to `CHAR(0)` before calling `xxx_clear()`.

  If something went wrong, you can store a value in the variable to which the `error` argument points. `error` points to a single-byte variable, not to a string buffer.

  `xxx_clear()` is required only by MySQL 4.1.1 and above. Before MySQL 4.1.1, use `xxx_reset()` instead.

- `xxx_add()`

  This function is called for all rows that belong to the same group, except for the first row. You should use it to add the value in the `UDF_ARGS` argument to your internal summary variable.

```
char *xxx_add(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

The `xxx()` function for an aggregate UDF should be declared the same way as for a non-aggregate UDF. See Section 25.2.3.1, “UDF Calling Sequences for simple functions”.

For an aggregate UDF, MySQL calls the `xxx()` function after all rows in the group have been processed. You should normally never access its `UDF_ARGS` argument here but instead return a value based on your internal summary variables.

Return value handling in `xxx()` should be done the same way as for a non-aggregate UDF. See Section 25.2.3.4, “Return Values and Error Handling”.

The `xxx_reset()` and `xxx_add()` functions handle their `UDF_ARGS` argument the same way as functions for non-aggregate UDFs. See Section 25.2.3.3, “Argument Processing”.

The pointer arguments to `is_null` and `error` are the same for all calls to `xxx_reset()`, `xxx_clear()`, `xxx_add()` and `xxx()`. You can use this to remember that you got an error or whether the `xxx()` function should return `NULL`. You should not store a string into `*error`! `error` points to a single-byte variable, not to a string buffer.

`*is_null` is reset for each group (before calling `xxx_clear()`). `*error` is never reset.

If `*is_null` or `*error` are set when `xxx()` returns, MySQL returns `NULL` as the result for the group function.

25.2.3.3. Argument Processing

The `args` parameter points to a `UDF_ARGS` structure that has the members listed here:

- `unsigned int arg_count`

  The number of arguments. Check this value in the initialization function if you require your function to be called with a particular number of arguments. For example:

  ```
  if (args->arg_count != 2)
  {
      strcpy(message,"XXX() requires two arguments");
      return 1;
  }
  ```

- `enum Item_result *arg_type`

  A pointer to an array containing the types for each argument. The possible type values are `STRING_RESULT`, `INT_RESULT`, and `REAL_RESULT`.

  To make sure that arguments are of a given type and return an error if they are not, check the `arg_type` array in the initialization function. For example:

  ```
  if (args->arg_type[0] != STRING_RESULT ||
      args->arg_type[1] != INT_RESULT)
  {
  ```

```
    strcpy(message,"XXX() requires a string and an integer");
    return 1;
}
```

As an alternative to requiring your function's arguments to be of particular types, you can use the initialization function to set the `arg_type` elements to the types you want. This causes MySQL to coerce arguments to those types for each call to `xxx()`. For example, to specify that the first two arguments should be coerced to string and integer, respectively, do this in `xxx_init()`:

```
args->arg_type[0] = STRING_RESULT;
args->arg_type[1] = INT_RESULT;
```

- `char **args`

  `args->args` communicates information to the initialization function about the general nature of the arguments passed to your function. For a constant argument `i`, `args->args[i]` points to the argument value. (See below for instructions on how to access the value properly.) For a non-constant argument, `args->args[i]` is `0`. A constant argument is an expression that uses only constants, such as `3` or `4*7-2` or `SIN(3.14)`. A non-constant argument is an expression that refers to values that may change from row to row, such as column names or functions that are called with non-constant arguments.

  For each invocation of the main function, `args->args` contains the actual arguments that are passed for the row currently being processed.

  Functions can refer to an argument `i` as follows:

  - An argument of type `STRING_RESULT` is given as a string pointer plus a length, to allow handling of binary data or data of arbitrary length. The string contents are available as `args->args[i]` and the string length is `args->lengths[i]`. You should not assume that strings are null-terminated.

  - For an argument of type `INT_RESULT`, you must cast `args->args[i]` to a `long long` value:

    ```
    long long int_val;
    int_val = *((long long*) args->args[i]);
    ```

  - For an argument of type `REAL_RESULT`, you must cast `args->args[i]` to a `double` value:

    ```
    double    real_val;
    real_val = *((double*) args->args[i]);
    ```

- `unsigned long *lengths`

  For the initialization function, the `lengths` array indicates the maximum string length for each argument. You should not change these. For each invocation of the main function, `lengths` contains the actual lengths of any string arguments that are passed for the row currently being processed. For arguments of types `INT_RESULT` or `REAL_RESULT`, `lengths` still contains the maximum length of the argument (as for the initialization function).

25.2.3.4. Return Values and Error Handling

The initialization function should return `0` if no error occurred and `1` otherwise. If an error occurs, `xxx_init()` should store a null-terminated error message in the `message` parameter. The message is returned to the client. The message buffer is `MYSQL_ERRMSG_SIZE` characters long, but you should try to keep the message to less than 80 characters so that it fits the width of a standard terminal screen.

The return value of the main function `xxx()` is the function value, for `long long` and `double` functions. A string function should return a pointer to the result and set `*result` and `*length` to the contents and length of the return value. For example:

```
memcpy(result, "result string", 13);
*length = 13;
```

The `result` buffer that is passed to the `xxx()` function is 255 bytes long. If your result fits in this, you don't have to worry about memory allocation for results.

If your string function needs to return a string longer than 255 bytes, you must allocate the space for it with `malloc()` in your `xxx_init()` function or your `xxx()` function and free it in your `xxx_deinit()` function. You can store the allocated memory in the `ptr` slot in the `UDF_INIT` structure for reuse by future `xxx()` calls. See Section 25.2.3.1, “UDF Calling Sequences for simple functions”.

To indicate a return value of `NULL` in the main function, set `*is_null` to `1`:

```
*is_null = 1;
```

To indicate an error return in the main function, set `*error` to `1`:

```
*error = 1;
```

If `xxx()` sets `*error` to `1` for any row, the function value is `NULL` for the current row and for any subsequent rows processed by the statement in which `XXX()` was invoked. (`xxx()` is not even called for subsequent rows.) **Note**: Before MySQL 3.22.10, you should set both `*error` and `*is_null`:

```
*error = 1;
*is_null = 1;
```

25.2.3.5. Compiling and Installing User-defined Functions

Files implementing UDFs must be compiled and installed on the host where the server runs. This process is described below for the example UDF file `sql/udf_example.cc` that is included in the MySQL source distribution.

The immediately following instructions are for Unix. Instructions for Windows are given later in this section.

The `udf_example.cc` file contains the following functions:

- `metaphon()` returns a metaphon string of the string argument. This is something like a soundex string, but it's more tuned for English.
- `myfunc_double()` returns the sum of the ASCII values of the characters in its arguments, divided by the sum of the length of its arguments.
- `myfunc_int()` returns the sum of the length of its arguments.

- `sequence([const int])` returns a sequence starting from the given number or 1 if no number has been given.
- `lookup()` returns the IP number for a hostname.
- `reverse_lookup()` returns the hostname for an IP number. The function may be called either with a single string argument of the form `'xxx.xxx.xxx.xxx'` or with four numbers.

A dynamically loadable file should be compiled as a sharable object file, using a command something like this:

```
shell> gcc -shared -o udf_example.so udf_example.cc
```

If you are using **gcc**, you should be able to create `udf_example.so` with a simpler command:

```
shell> make udf_example.so
```

You can easily determine the correct compiler options for your system by running this command in the `sql` directory of your MySQL source tree:

```
shell> make udf_example.o
```

You should run a compile command similar to the one that **make** displays, except that you should remove the `-c` option near the end of the line and add `-o udf_example.so` to the end of the line. (On some systems, you may need to leave the `-c` on the command.)

After you compile a shared object containing UDFs, you must install it and tell MySQL about it. Compiling a shared object from `udf_example.cc` produces a file named something like `udf_example.so` (the exact name may vary from platform to platform). Copy this file to some directory such as `/usr/lib` that searched by your system's dynamic (runtime) linker, or add the directory in which you placed the shared object to the linker configuration file (for example, `/etc/ld.so.conf`).

The dynamic linker name is system-specific (for example, **ld-elf.so.1** on FreeBSD, **ld.so** on Linux, or **dyld** on Mac OS X). Consult your system documentation for information about the linker name and how to configure it.

On many systems, you can also set the `LD_LIBRARY` or `LD_LIBRARY_PATH` environment variable to point at the directory where you have the files for your UDF. The `dlopen` manual page tells you which variable you should use on your system. You should set this in **mysql.server** or **mysqld_safe** startup scripts and restart **mysqld**.

On some systems, the **ldconfig** program that configures the dynamic linker does not recognize a shared object unless its name begins with `lib`. In this case you should rename a file such as `udf_example.so` to `libudf_example.so`.

On Windows, you can compile user-defined functions by using the following procedure:

1. You need to obtain the BitKeeper source repository for MySQL 4.0 or higher. See Section 2.8.3, “Installing from the Development Source Tree”.
2. In the source repository, look in the `VC++Files/examples/udf_example` directory. There are files named `udf_example.def`, `udf_example.dsp`, and `udf_example.dsw` there.
3. In the source repository, look in the `sql` directory. Copy the `udf_example.cc` from this direct-

ory to the `VC++Files/examples/udf_example` directory and rename the file to `udf_example.cpp`.

4. Open the `udf_example.dsw` file with Visual Studio VC++ and use it to compile the UDFs as a normal project.

After the shared object file has been installed, notify **mysqld** about the new functions with these statements:

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE FUNCTION myfunc_double RETURNS REAL SONAME 'udf_example.so';
mysql> CREATE FUNCTION myfunc_int RETURNS INTEGER SONAME 'udf_example.so';
mysql> CREATE FUNCTION lookup RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE FUNCTION reverse_lookup
    ->        RETURNS STRING SONAME 'udf_example.so';
mysql> CREATE AGGREGATE FUNCTION avgcost
    ->        RETURNS REAL SONAME 'udf_example.so';
```

Functions can be deleted using `DROP FUNCTION`:

```
mysql> DROP FUNCTION metaphon;
mysql> DROP FUNCTION myfunc_double;
mysql> DROP FUNCTION myfunc_int;
mysql> DROP FUNCTION lookup;
mysql> DROP FUNCTION reverse_lookup;
mysql> DROP FUNCTION avgcost;
```

The `CREATE FUNCTION` and `DROP FUNCTION` statements update the `func` system table in the `mysql` database. The function's name, type and shared library name are saved in the table. You must have the `INSERT` and `DELETE` privileges for the `mysql` database to create and drop functions.

You should not use `CREATE FUNCTION` to add a function that has previously been created. If you need to reinstall a function, you should remove it with `DROP FUNCTION` and then reinstall it with `CREATE FUNCTION`. You would need to do this, for example, if you recompile a new version of your function, so that **mysqld** gets the new version. Otherwise, the server continues to use the old version.

An active function is one that has been loaded with `CREATE FUNCTION` and not removed with `DROP FUNCTION`. All active functions are reloaded each time the server starts, unless you start **mysqld** with the `--skip-grant-tables` option. In this case, UDF initialization is skipped and UDFs are unavailable.

25.2.3.6. User-defined Function Security Precautions

MySQL takes the following measures to prevent misuse of user-defined functions.

You must have the `INSERT` privilege to be able to use `CREATE FUNCTION` and the `DELETE` privilege to be able to use `DROP FUNCTION`. This is necessary because these statements add and delete rows from the `mysql.func` table.

UDFs should have at least one symbol defined in addition to the `xxx` symbol that corresponds to the main `xxx()` function. These auxiliary symbols correspond to the `xxx_init()`, `xxx_deinit()`, `xxx_reset()`, `xxx_clear()`, and `xxx_add()` functions. As of MySQL 4.0.24, 4.1.10a, and 5.0.3, **mysqld** supports an `--allow-suspicious-udfs` option that controls whether UDFs that have only an `xxx` symbol can be loaded. By default, the option is off, to prevent attempts at loading functions from shared object files other than those containing legitimate UDFs. If you have older UDFs that contain only the `xxx` symbol and that cannot be recompiled to include an auxiliary symbol, it may be necessary to specify the `--allow-suspicious-udfs` option. Otherwise, you should avoid en-

abling this capability.

UDF object files cannot be placed in arbitrary directories. They must be located in some system directory that the dynamic linker is configured to search. To enforce this restriction and prevent attempts at specifying pathnames outside of directories searched by the dynamic linker, MySQL checks the shared object file name specified in `CREATE FUNCTION` statements for pathname delimiter characters. As of MySQL 4.0.24, 4.1.10a, and 5.0.3, MySQL also checks for pathname delimiters in filenames stored in the `mysql.func` table when it loads functions. This prevents attempts at specifying illegitimate pathnames through direct manipulation of the `mysql.func` table. For information about UDFs and the runtime linker, see Section 25.2.3.5, “Compiling and Installing User-defined Functions”.

25.2.4. Adding a New Native Function

The procedure for adding a new native function is described here. Note that you cannot add native functions to a binary distribution because the procedure involves modifying MySQL source code. You must compile MySQL yourself from a source distribution. Also note that if you migrate to another version of MySQL (for example, when a new version is released), you need to repeat the procedure with the new version.

To add a new native MySQL function, follow these steps:

1. Add one line to `lex.h` that defines the function name in the `sql_functions[]` array.

2. If the function prototype is simple (just takes zero, one, two or three arguments), you should in `lex.h` specify `SYM(FUNC_ARG`*N*`)` (where *N* is the number of arguments) as the second argument in the `sql_functions[]` array and add a function that creates a function object in `item_create.cc`. Take a look at `"ABS"` and `create_funcs_abs()` for an example of this.

   If the function prototype is complicated (for example, if it takes a variable number of arguments), you should add two lines to `sql_yacc.yy`. One indicates the preprocessor symbol that **yacc** should define (this should be added at the beginning of the file). Then define the function parameters and add an ``item" with these parameters to the `simple_expr` parsing rule. For an example, check all occurrences of `ATAN` in `sql_yacc.yy` to see how this is done.

3. In `item_func.h`, declare a class inheriting from `Item_num_func` or `Item_str_func`, depending on whether your function returns a number or a string.

4. In `item_func.cc`, add one of the following declarations, depending on whether you are defining a numeric or string function:

   ```
   double   Item_func_newname::val()
   longlong Item_func_newname::val_int()
   String  *Item_func_newname::Str(String *str)
   ```

   If you inherit your object from any of the standard items (like `Item_num_func`), you probably only have to define one of these functions and let the parent object take care of the other functions. For example, the `Item_str_func` class defines a `val()` function that executes `atof()` on the value returned by `::str()`.

5. You should probably also define the following object function:

   ```
   void Item_func_newname::fix_length_and_dec()
   ```

   This function should at least calculate `max_length` based on the given arguments. `max_length` is the maximum number of characters the function may return. This function should also set `maybe_null = 0` if the main function can't return a `NULL` value. The function can check whether any of the function arguments can return `NULL` by checking the arguments'

maybe_null variable. You can take a look at Item_func_mod::fix_length_and_dec for a typical example of how to do this.

All functions must be thread-safe. In other words, don't use any global or static variables in the functions without protecting them with mutexes)

If you want to return NULL, from ::val(), ::val_int() or ::str() you should set null_value to 1 and return 0.

For ::str() object functions, there are some additional considerations to be aware of:

- The String *str argument provides a string buffer that may be used to hold the result. (For more information about the String type, take a look at the sql_string.h file.)
- The ::str() function should return the string that holds the result or (char*) 0 if the result is NULL.
- All current string functions try to avoid allocating any memory unless absolutely necessary!

25.3. Adding New Procedures to MySQL

In MySQL, you can define a procedure in C++ that can access and modify the data in a query before it is sent to the client. The modification can be done on a row-by-row or GROUP BY level.

We have created an example procedure in MySQL 3.23 to show you what can be done.

Additionally, we recommend that you take a look at mylua. With this you can use the LUA language to load a procedure at runtime into **mysqld**.

25.3.1. Procedure Analyse

analyse([*max_elements*,[*max_memory*]])

This procedure is defined in the sql/sql_analyse.cc. This examines the result from your query and returns an analysis of the results:

- *max_elements* (default 256) is the maximum number of distinct values analyse does notice per column. This is used by analyse to check whether the optimal column type should be of type ENUM.
- *max_memory* (default 8192) is the maximum amount of memory that analyse should allocate per column while trying to find all distinct values.

SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([*max_elements*,[*max_memory*]])

25.3.2. Writing a Procedure

For the moment, the only documentation for this is the source.

You can find all information about procedures by examining the following files:

- `sql/sql_analyse.cc`
- `sql/procedure.h`
- `sql/procedure.cc`
- `sql/sql_select.cc`

Appendix A. Problems and Common Errors

This appendix lists some common problems and error messages that you may encounter. It describes how to determine the causes of the problems and what to do to solve them.

A.1. How to Determine What Is Causing a Problem

When you run into a problem, the first thing you should do is to find out which program or piece of equipment is causing it:

- If you have one of the following symptoms, then it is probably a hardware problems (such as memory, motherboard, CPU, or hard disk) or kernel problem:
  - The keyboard doesn't work. This can normally be checked by pressing the Caps Lock key. If the Caps Lock light doesn't change, you have to replace your keyboard. (Before doing this, you should try to restart your computer and check all cables to the keyboard.)
  - The mouse pointer doesn't move.
  - The machine doesn't answer to a remote machine's pings.
  - Other programs that are not related to MySQL don't behave correctly.
  - Your system restarted unexpectedly. (A faulty user-level program should never be able to take down your system.)

  In this case, you should start by checking all your cables and run some diagnostic tool to check your hardware! You should also check whether there are any patches, updates, or service packs for your operating system that could likely solve your problem. Check also that all your libraries (such as `glibc`) are up to date.

  It's always good to use a machine with ECC memory to discover memory problems early.
- If your keyboard is locked up, you may be able to recover by logging in to your machine from another machine and executing `kbd_mode -a`.
- Please examine your system log file (`/var/log/messages` or similar) for reasons for your problem. If you think the problem is in MySQL, you should also examine MySQL's log files. See Section 5.9, “The MySQL Log Files”.
- If you don't think you have hardware problems, you should try to find out which program is causing problems. Try using **top**, **ps**, Task Manager, or some similar program, to check which program is taking all CPU or is locking the machine.
- Use **top**, **df**, or a similar program to check whether you are out of memory, disk space, file descriptors, or some other critical resource.
- If the problem is some runaway process, you can always try to kill it. If it doesn't want to die, there is probably a bug in the operating system.

If after you have examined all other possibilities and you have concluded that the MySQL server or a MySQL client is causing the problem, it's time to create a bug report for our mailing list or our support team. In the bug report, try to give a very detailed description of how the system is behaving and what you think is happening. You should also state why you think that MySQL is causing the problem. Take into consideration all the situations in this chapter. State any problems exactly how they appear when you examine your system. Use the ``copy and paste'' method for any output and error messages from programs and log files.

Try to describe in detail which program is not working and all symptoms you see. We have in the past received many bug reports that state only ``the system doesn't work.'' This doesn't provide us with any information about what could be the problem.

If a program fails, it's always useful to know the following information:

- Has the program in question made a segmentation fault (did it dump core)?
- Is the program taking up all available CPU time? Check with **top**. Let the program run for a while, it may simply be evaluating something computationally intensive.
- If the **mysqld** server is causing problems, can you get any response from it with **mysqladmin -u root ping** or **mysqladmin -u root processlist**?
- What does a client program say when you try to connect to the MySQL server? (Try with **mysql**, for example.) Does the client jam? Do you get any output from the program?

When sending a bug report, you should follow the outline described in Section 1.4.1.2, “Asking Questions or Reporting Bugs”.

A.2. Common Errors When Using MySQL Programs

This section lists some errors that users frequently encounter when running MySQL programs. Although the problems show up when you try to run client programs, the solutions to many of the problems involves changing the configuration of the MySQL server.

A.2.1. `Access denied`

An `Access denied` error can have many causes. Often the problem is related to the MySQL accounts that the server allows client programs to use when connecting. See Section 5.5.8, “Causes of `Access denied` Errors”. See Section 5.5.2, “How the Privilege System Works”.

A.2.2. `Can't connect to [local] MySQL server`

A MySQL client on Unix can connect to the **mysqld** server in two different ways: By using a Unix socket file to connect through a file in the filesystem (default `/tmp/mysql.sock`), or by using TCP/IP, which connects through a port number. A Unix socket file connection is faster than TCP/IP, but can be used only when connecting to a server on the same computer. A Unix socket file is used if you don't specify a hostname or if you specify the special hostname `localhost`.

If the MySQL server is running on Windows 9x or Me, you can connect only via TCP/IP. If the server is running on Windows NT, 2000, XP, or 2003 and is started with the `--enable-named-pipe` option, you can also connect with named pipes if you run the client on the host where the server is running. The name of the named pipe is `MySQL` by default. If you don't give a hostname when connecting to **mysqld**,

a MySQL client first tries to connect to the named pipe. If that doesn't work, it connects to the TCP/IP port. You can force the use of named pipes on Windows by using `.` as the hostname.

The error (2002) `Can't connect to ...` normally means that there is no MySQL server running on the system or that you are using an incorrect Unix socket filename or TCP/IP port number when trying to connect to the server.

Start by checking whether there is a process named **mysqld** running on your server host. (Use **ps xa | grep mysqld** on Unix or the Task Manager on Windows.) If there is no such process, you should start the server. See Section 2.9.2.3, “Starting and Troubleshooting the MySQL Server”.

If a **mysqld** process is running, you can check it by trying the following commands. The port number or Unix socket filename might be different in your setup. `host_ip` represents the IP number of the machine where the server is running.

```
shell> mysqladmin version
shell> mysqladmin variables
shell> mysqladmin -h `hostname` version variables
shell> mysqladmin -h `hostname` --port=3306 version
shell> mysqladmin -h host_ip version
shell> mysqladmin --protocol=socket --socket=/tmp/mysql.sock version
```

Note the use of backticks rather than forward quotes with the `hostname` command; these cause the output of `hostname` (that is, the current hostname) to be substituted into the **mysqladmin** command. If you have no `hostname` command or are running on Windows, you can manually type the hostname of your machine (without backticks) following the `-h` option. You can also try `-h 127.0.0.1` to connect with TCP/IP to the local host.

Here are some reasons the `Can't connect to local MySQL server` error might occur:

- **mysqld** is not running. Check your operating system's process list to ensure the **mysqld** process is present.
- You are running on a system that uses MIT-pthreads. If you are running on a system that doesn't have native threads, **mysqld** uses the MIT-pthreads package. See Section 2.1.1, “Operating Systems Supported by MySQL”. However, not all MIT-pthreads versions support Unix socket files. On a system without socket file support, you must always specify the hostname explicitly when connecting to the server. Try using this command to check the connection to the server:

  ```
  shell> mysqladmin -h `hostname` version
  ```

- Someone has removed the Unix socket file that **mysqld** uses (`/tmp/mysql.sock` by default). For example, you might have a **cron** job that removes old files from the `/tmp` directory. You can always run **mysqladmin version** to check whether the Unix socket file that **mysqladmin** is trying to use really exists. The fix in this case is to change the **cron** job to not remove `mysql.sock` or to place the socket file somewhere else. See Section A.4.5, “How to Protect or Change the MySQL Socket File `/tmp/mysql.sock`”.
- You have started the **mysqld** server with the `--socket=/path/to/socket` option, but forgotten to tell client programs the new name of the socket file. If you change the socket pathname for the server, you must also notify the MySQL clients. You can do this by providing the same `--socket` option when you run client programs. You also need to ensure that clients have permission to access the `mysql.sock` file. To find out where the socket file is, you can do:

  ```
  shell> netstat -ln | grep mysql
  ```

  See Section A.4.5, “How to Protect or Change the MySQL Socket File `/tmp/mysql.sock`”.

- You are using Linux and one server thread has died (dumped core). In this case, you must kill the other **mysqld** threads (for example, with `kill` or with the `mysql_zap` script) before you can restart the MySQL server. See Section A.4.2, “What to Do If MySQL Keeps Crashing”.
- The server or client program might not have the proper access privileges for the directory that holds the Unix socket file or the socket file itself. In this case, you must either change the access privileges for the directory or socket file so that the server and clients can access them, or restart **mysqld** with a `--socket` option that specifies a socket filename in a directory where the server can create it and where client programs can access it.

If you get the error message `Can't connect to MySQL server on some_host`, you can try the following things to find out what the problem is:

- Check whether the server is running on that host by executing `telnet some_host 3306` and pressing the Enter key a couple of times. (3306 is the default MySQL port number. Change the value if your server is listening to a different port.) If there is a MySQL server running and listening to the port, you should get a response that includes the server's version number. If you get an error such as `telnet: Unable to connect to remote host: Connection refused`, then there is no server running on the given port.
- If the server is running on the local host, try using **mysqladmin -h localhost variables** to connect using the Unix socket file. Verify the TCP/IP port number that the server is configured to listen to (it is the value of the `port` variable.)
- Make sure that your **mysqld** server was not started with the `--skip-networking` option. If it was, you cannot connect to it using TCP/IP.
- Check to make sure that there is no firewall blocking access to MySQL. Applications such as ZoneAlarm and the Windows XP personal firewall may need to be configured to allow external access to a MySQL server.

A.2.3. Client does not support authentication protocol

MySQL 4.1 and up uses an authentication protocol based on a password hashing algorithm that is incompatible with that used by older clients. If you upgrade the server to 4.1, attempts to connect to it with an older client may fail with the following message:

```
shell> mysql
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

To solve this problem, you should use one of the following approaches:

- Upgrade all client programs to use a 4.1.1 or newer client library.
- When connecting to the server with a pre-4.1 client program, use an account that still has a pre-4.1-style password.
- Reset the password to pre-4.1 style for each user that needs to use a pre-4.1 client program. This can be done using the `SET PASSWORD` statement and the `OLD_PASSWORD()` function:

  ```
  mysql> SET PASSWORD FOR
  ```

```
    -> 'some_user'@'some_host' = OLD_PASSWORD('newpwd');
```

Alternatively, use `UPDATE` and `FLUSH PRIVILEGES`:

```
mysql> UPDATE mysql.user SET Password = OLD_PASSWORD('newpwd')
    -> WHERE Host = 'some_host' AND User = 'some_user';
mysql> FLUSH PRIVILEGES;
```

Substitute the password you want to use for ``*newpwd*'' in the preceding examples. MySQL cannot tell you what the original password was, so you'll need to pick a new one.

- Tell the server to use the older password hashing algorithm:

  1. Start **mysqld** with the `--old-passwords` option.

  2. Assign an old-format password to each account that has had its password updated to the longer 4.1 format. You can identify these accounts with the following query:

     ```
     mysql> SELECT Host, User, Password FROM mysql.user
         -> WHERE LENGTH(Password) > 16;
     ```

     For each account record displayed by the query, use the `Host` and `User` values and assign a password using the `OLD_PASSWORD()` function and either `SET PASSWORD` or `UPDATE`, as described earlier.

Note: In PHP, the `mysql` extension does not support the new authentication protocol in MySQL 4.1.1 and higher. This is true regardless of the PHP version being used. If you wish to use the `mysql` extension with MySQL 4.1 or newer, you will need to follow one of the options discussed above for configuring MySQL to work with old clients. The `mysqli` extension (stands for "MySQL, Improved"; new in PHP 5) **is** compatible with the improved password hashing employed in MySQL 4.1 and higher, and no special configuration of MySQL need be done in order to use this newer MySQL client library for PHP. For more information about the `mysqli` extension, see http://php.net/mysqli.

For additional background on password hashing and authentication, see Section 5.5.9, “Password Hashing in MySQL 4.1”.

A.2.4. Password Fails When Entered Interactively

MySQL client programs prompt for a password when invoked with a `--password` or `-p` option that has no following password value:

```
shell> mysql -u user_name -p
Enter password:
```

On some systems, you may find that your password works when specified in an option file or on the command line, but not when you enter it interactively at the `Enter password:` prompt. This occurs when the library provided by the system to read passwords limits password values to a small number of characters (typically eight). That is a problem with the system library, not with MySQL. To work around it, change your MySQL password to a value that is eight or fewer characters long, or put your password in an option file.

A.2.5. Host '*host_name*' is blocked

If you get the following error, it means that **mysqld** has received many connect requests from the host '*host_name*' that have been interrupted in the middle:

```
Host 'host_name' is blocked because of many connection errors.
Unblock with 'mysqladmin flush-hosts'
```

The number of interrupted connect requests allowed is determined by the value of the `max_connect_errors` system variable. After `max_connect_errors` failed requests, **mysqld** assumes that something is wrong (for example, that someone is trying to break in), and blocks the host from further connections until you execute a **mysqladmin flush-hosts** command or issue a `FLUSH HOSTS` statement. See Section 5.2.3, "Server System Variables".

By default, **mysqld** blocks a host after 10 connection errors. You can adjust the value by starting the server like this:

```
shell> mysqld_safe --max_connect_errors=10000 &
```

If you get this error message for a given host, you should first verify that there isn't anything wrong with TCP/IP connections from that host. If you are having network problems, it does you no good to increase the value of the `max_connect_errors` variable.

A.2.6. Too many connections

If you get a `Too many connections` error when you try to connect to the **mysqld** server, this means that all available connections are in use by other clients.

The number of connections allowed is controlled by the `max_connections` system variable. Its default value is 100. If you need to support more connections, you should restart **mysqld** with a larger value for this variable.

mysqld actually allows `max_connections+1` clients to connect. The extra connection is reserved for use by accounts that have the `SUPER` privilege. By granting the `SUPER` privilege to administrators and not to normal users (who should not need it), an administrator can connect to the server and use `SHOW PROCESSLIST` to diagnose problems even if the maximum number of unprivileged clients are connected. See Section 13.5.4.15, "`SHOW PROCESSLIST` Syntax".

The maximum number of connections MySQL can support depends on the quality of the thread library on a given platform. Linux or Solaris should be able to support 500-1000 simultaneous connections, depending on how much RAM you have and what your clients are doing. Static Linux binaries provided by MySQL AB can support up to 4000 connections.

A.2.7. Out of memory

If you issue a query using the **mysql** client program and receive an error like the following one, it means that **mysql** does not have enough memory to store the entire query result:

```
mysql: Out of memory at line 42, 'malloc.c'
mysql: needed 8136 byte (8k), memory in use: 12481367 bytes (12189k)
ERROR 2008: MySQL client ran out of memory
```

To remedy the problem, first check whether your query is correct. Is it reasonable that it should return so many rows? If not, correct the query and try again. Otherwise, you can invoke **mysql** with the `--quick` option. This causes it to use the `mysql_use_result()` C API function to retrieve the result set, which places less of a load on the client (but more on the server).

A.2.8. MySQL server has gone away

This section also covers the related `Lost connection to server during query` error.

The most common reason for the `MySQL server has gone away` error is that the server timed out and closed the connection. In this case, you normally get one of the following error codes (which one you get is operating system-dependent):

Error Code	Description
`CR_SERVER_GONE_ERROR`	The client couldn't send a question to the server.
`CR_SERVER_LOST`	The client didn't get an error when writing to the server, but it didn't get a full answer (or any answer) to the question.

By default, the server closes the connection after eight hours if nothing has happened. You can change the time limit by setting the `wait_timeout` variable when you start **mysqld**. See Section 5.2.3, “Server System Variables”.

If you have a script, you just have to issue the query again for the client to do an automatic reconnection. This assumes that you have automatic reconnection in the client enabled (which is the default for the `mysql` command-line client).

Some other common reasons for the `MySQL server has gone away` error are:

- You (or the db administrator) has killed the running thread with a `KILL` statement or a **mysqladmin kill** command.
- You tried to run a query after closing the connection to the server. This indicates a logic error in the application that should be corrected.
- You got a timeout from the TCP/IP connection on the client side. This may happens if you have been using the commands: `mysql_options(..., MYSQL_OPT_READ_TIMEOUT,...)` or `mysql_options(..., MYSQL_OPT_WRITE_TIMEOUT,...)`. In this case increasing the timeout may help solve the problem.
- You have encountered a timeout on the server side and the automatic reconnection in the client is disabled (the `reconnect` flag in the `MYSQL` structure is equal to 0).
- You are using a windows client and the server had dropped the connection (probably because `wait_timeout` expired) before the command was issued.

  The problem on windows is that in some cases MySQL doesn't get an error from the OS when writing to the TCP/IP connection to the server, but instead gets the error when trying to read the answer from connection.

  In this case, even if the `reconnect` flag in the `MYSQL` structure is equal to 1, MySQL does not automatically reconnect and re-issue the query as it doesn't know if the server did get the original query or not.

  The solution to this is to either do a `mysql_ping` on the connection if there has been a long time since the last query (this is what `MyODBC` does) or set `wait_timeout` on the **mysqld** server so high that it in practice never times out.
- You can also get these errors if you send a query to the server that is incorrect or too large. If **mysqld** receives a packet that is too large or out of order, it assumes that something has gone wrong with the client and closes the connection. If you need big queries (for example, if you are working with big `BLOB` columns), you can increase the query limit by setting the server's

max_allowed_packet variable, which has a default value of 1MB. You may also need to increase the maximum packet size on the client end. More information on setting the packet size is given in Section A.2.9, "Packet too large".

- You also get a lost connection if you are sending a packet 16MB or larger if your client is older than 4.0.8 and your server is 4.0.8 and above, or the other way around.
- You may also see the MySQL server has gone away error if MySQL is started with the --skip-networking option.
- You have encountered a bug where the server died while executing the query.

You can check whether the MySQL server died and restarted by executing **mysqladmin version** and examining the server's uptime. If the client connection was broken because **mysqld** crashed and restarted, you should concentrate on finding the reason for the crash. Start by checking whether issuing the query again kills the server again. See Section A.4.2, "What to Do If MySQL Keeps Crashing".

You can get more information about the lost connections by starting mysqld with the --log-warnings=2 option. This logs some of the disconnected errors in the hostname.err file. See Section 5.9.1, "The Error Log".

If you want to create a bug report regarding this problem, be sure that you include the following information:

- Indicate whether or not the MySQL server died. You can find information about this in the server error log. See Section A.4.2, "What to Do If MySQL Keeps Crashing".
- If a specific query kills **mysqld** and the tables involved were checked with CHECK TABLE before you ran the query, can you provide a reproducible test case? See Section E.1.6, "Making a Test Case If You Experience Table Corruption".
- What is the value of the wait_timeout system variable in the MySQL server? (**mysqladmin variables** gives you the value of this variable.)
- Have you tried to run **mysqld** with the --log option to determine whether the problem query appears in the log?

See also See Section A.2.10, "Communication Errors and Aborted Connections".

See Section 1.4.1.2, "Asking Questions or Reporting Bugs".

A.2.9. Packet too large

A communication packet is a single SQL statement sent to the MySQL server or a single row that is sent to the client.

In MySQL 3.23, the largest possible packet is 16MB, due to limits in the client/server protocol. In MySQL 4.0.1 and up, the limit is 1GB.

When a MySQL client or the **mysqld** server receives a packet bigger than max_allowed_packet bytes, it issues a Packet too large error and closes the connection. With some clients, you may also get a Lost connection to MySQL server during query error if the communication packet is too large.

Both the client and the server have their own max_allowed_packet variable, so if you want to

handle big packets, you must increase this variable both in the client and in the server.

If you are using the **mysql** client program, its default `max_allowed_packet` variable is 16MB. That is also the maximum value before MySQL 4.0. To set a larger value from 4.0 on, start **mysql** like this:

```
mysql> mysql --max_allowed_packet=32M
```

That sets the packet size to 32MB.

The server's default `max_allowed_packet` value is 1MB. You can increase this if the server needs to handle big queries (for example, if you are working with big `BLOB` columns). For example, to set the variable to 16MB, start the server like this:

```
mysql> mysqld --max_allowed_packet=16M
```

Before MySQL 4.0, use this syntax instead:

```
mysql> mysqld --set-variable=max_allowed_packet=16M
```

You can also use an option file to set `max_allowed_packet`. For example, to set the size for the server to 16MB, add the following lines in an option file:

```
[mysqld]
max_allowed_packet=16M
```

Before MySQL 4.0, use this syntax instead:

```
[mysqld]
set-variable = max_allowed_packet=16M
```

It's safe to increase the value of this variable because the extra memory is allocated only when needed. For example, **mysqld** allocates more memory only when you issue a long query or when **mysqld** must return a large result row. The small default value of the variable is a precaution to catch incorrect packets between the client and server and also to ensure that you don't run out of memory by using large packets accidentally.

You can also get strange problems with large packets if you are using large `BLOB` values but have not given **mysqld** access to enough memory to handle the query. If you suspect this is the case, try adding **ulimit -d 256000** to the beginning of the **mysqld_safe** script and restarting **mysqld**.

A.2.10. Communication Errors and Aborted Connections

The server error log can be a useful source of information about connection problems. See Section 5.9.1, “The Error Log”. Starting with MySQL 3.23.40, if you start the server with the `--warnings` option (or `--log-warnings` from MySQL 4.0.3 on), you might find messages like this in your error log:

```
010301 14:38:23  Aborted connection 854 to db: 'users' user: 'josh'
```

If `Aborted connections` messages appear in the error log, the cause can be any of the following:

- The client program did not call `mysql_close()` before exiting.
- The client had been sleeping more than `wait_timeout` or `interactive_timeout` seconds without issuing any requests to the server. See Section 5.2.3, “Server System Variables”.

- The client program ended abruptly in the middle of a data transfer.

When any of these things happen, the server increments the `Aborted_clients` status variable.

The server increments the `Aborted_connects` status variable when the following things happen:

- A client doesn't have privileges to connect to a database.
- A client uses an incorrect password.
- A connection packet doesn't contain the right information.
- It takes more than `connect_timeout` seconds to get a connect packet. See Section 5.2.3, “Server System Variables”.

If these kinds of things happen, it might indicate that someone is trying to break into your server!

Other reasons for problems with aborted clients or aborted connections:

- Use of Ethernet protocol with Linux, both half and full duplex. Many Linux Ethernet drivers have this bug. You should test for this bug by transferring a huge file via FTP between the client and server machines. If a transfer goes in burst-pause-burst-pause mode, you are experiencing a Linux duplex syndrome. The only solution is switching the duplex mode for both your network card and hub/switch to either full duplex or to half duplex and testing the results to determine the best setting.
- Some problem with the thread library that causes interrupts on reads.
- Badly configured TCP/IP.
- Faulty Ethernets, hubs, switches, cables, and so forth. This can be diagnosed properly only by replacing hardware.
- The `max_allowed_packet` variable value is too small or queries require more memory than you have allocated for **mysqld**. See Section A.2.9, “`Packet too large`”.

See also See Section A.2.8, “`MySQL server has gone away`”.

A.2.11. The table is full

There are several ways a full-table error can occur:

- You are using a MySQL server older than 3.23 and an in-memory temporary table becomes larger than `tmp_table_size` bytes. To avoid this problem, you can use the `-O tmp_table_size=#` option to make **mysqld** increase the temporary table size or use the SQL option `SQL_BIG_TABLES` before you issue the problematic query. See Section 13.5.3, “`SET` Syntax”.

  You can also start **mysqld** with the `--big-tables` option. This is exactly the same as using `SQL_BIG_TABLES` for all queries.

  As of MySQL 3.23, this problem should not occur. If an in-memory temporary table becomes larger than `tmp_table_size`, the server automatically converts it to a disk-based `MyISAM` table.

- You are using `InnoDB` tables and run out of room in the `InnoDB` tablespace. In this case, the solu-

tion is to extend the InnoDB tablespace. See Section 15.8, “Adding and Removing InnoDB Data and Log Files”.

- You are using ISAM or MyISAM tables on an operating system that supports files only up to 2GB in size and you have hit this limit for the data file or index file.
- You are using a MyISAM table and the space required for the table exceeds what is allowed by the internal pointer size. (If you don't specify the MAX_ROWS table option when you create a table, MySQL uses the myisam_data_pointer_size system variable. Its default value of 4 bytes is enough to allow only 4GB of data.) See Section 5.2.3, “Server System Variables”.

  You can check the maximum data/index sizes by using this statement:

  ```
  SHOW TABLE STATUS FROM database LIKE 'tbl_name';
  ```

  You also can use **myisamchk -dv /path/to/table-index-file**.

  If the pointer size is too small, you can fix the problem by using ALTER TABLE:

  ```
  ALTER TABLE tbl_name MAX_ROWS=1000000000 AVG_ROW_LENGTH=nnn;
  ```

  You have to specify AVG_ROW_LENGTH only for tables with BLOB or TEXT columns; in this case, MySQL can't optimize the space required based only on the number of rows.

A.2.12. Can't create/write to file

If you get an error of the following type for some queries, it means that MySQL cannot create a temporary file for the result set in the temporary directory:

```
Can't create/write to file '\\sqla3fe_0.ism'.
```

The preceding error is a typical message for Windows; the Unix message is similar.

One fix is to start **mysqld** with the --tmpdir option or to add the option to the [mysqld] section of your option file. For example, to specify a directory of C:\temp, use these lines:

```
[mysqld]
tmpdir=C:/temp
```

The C:\temp directory must exist and have sufficient space for the MySQL server to write to. See Section 4.3.2, “Using Option Files”.

Another cause of this error can be permissions issues. Make sure that the MySQL server can write to the tmpdir directory.

Check also the error code that you get with **perror**. One reason the server cannot write to a table is that the filesystem is full:

```
shell> perror 28
Error code  28:  No space left on device
```

A.2.13. Commands out of sync

If you get `Commands out of sync; you can't run this command now` in your client code, you are calling client functions in the wrong order.

This can happen, for example, if you are using `mysql_use_result()` and try to execute a new query before you have called `mysql_free_result()`. It can also happen if you try to execute two queries that return data without calling `mysql_use_result()` or `mysql_store_result()` in between.

A.2.14. Ignoring user

If you get the following error, it means that when **mysqld** was started or when it reloaded the grant tables, it found an account in the `user` table that had an invalid password.

```
Found wrong password for user 'some_user'@'some_host'; ignoring user
```

As a result, the account is simply ignored by the permission system.

The following list indicates possible causes of and fixes for this problem:

- You may be running a new version of **mysqld** with an old `user` table. You can check this by executing **mysqlshow mysql user** to see whether the `Password` column is shorter than 16 characters. If so, you can correct this condition by running the `scripts/add_long_password` script.
- The account has an old password (eight characters long) and you didn't start **mysqld** with the `--old-protocol` option. Update the account in the `user` table to have a new password or restart **mysqld** with the `--old-protocol` option.
- You have specified a password in the `user` table without using the `PASSWORD()` function. Use **mysql** to update the account in the `user` table with a new password, making sure to use the `PASSWORD()` function:

  ```
  mysql> UPDATE user SET Password=PASSWORD('newpwd')
      -> WHERE User='some_user' AND Host='some_host';
  ```

A.2.15. Table '*tbl_name*' doesn't exist

If you get either of the following errors, it usually means that no table exists in the current database with the given name:

```
Table 'tbl_name' doesn't exist
Can't find file: 'tbl_name' (errno: 2)
```

In some cases, it may be that the table does exist but that you are referring to it incorrectly:

- Because MySQL uses directories and files to store databases and tables, database and table names are case sensitive if they are located on a filesystem that has case-sensitive filenames.
- Even for filesystems that are not case sensitive, such as on Windows, all references to a given table within a query must use the same lettercase.

You can check which tables are in the current database with `SHOW TABLES`. See Section 13.5.4,

`SHOW` Syntax".

A.2.16. Can't initialize character set

You might see an error like this if you have character set problems:

```
MySQL Connection Failed: Can't initialize character set charset_name
```

This error can have any of the following causes:

- The character set is a multi-byte character set and you have no support for the character set in the client. In this case, you need to recompile the client by running **configure** with the `--with-charset=charset_name` or `--with-extra-charsets=charset_name` option. See Section 2.8.2, "Typical **configure** Options".

  All standard MySQL binaries are compiled with `--with-extra-character-sets=complex`, which enables support for all multi-byte character sets. See Section 5.8.1, "The Character Set Used for Data and Sorting".

- The character set is a simple character set that is not compiled into **mysqld**, and the character set definition files are not in the place where the client expects to find them.

  In this case, you need to use one of the following methods to solve the problem:

  - Recompile the client with support for the character set. See Section 2.8.2, "Typical **configure** Options".
  - Specify to the client the directory where the character set definition files are located. For many clients, you can do this with the `--character-sets-dir` option.
  - Copy the character definition files to the path where the client expects them to be.

A.2.17. File Not Found

If you get `ERROR '...' not found (errno: 23)`, `Can't open file: ... (errno: 24)`, or any other error with `errno 23` or `errno 24` from MySQL, it means that you haven't allocated enough file descriptors for the MySQL server. You can use the **perror** utility to get a description of what the error number means:

```
shell> perror 23
Error code  23:  File table overflow
shell> perror 24
Error code  24:  Too many open files
shell> perror 11
Error code  11:  Resource temporarily unavailable
```

The problem here is that **mysqld** is trying to keep open too many files simultaneously. You can either tell **mysqld** not to open so many files at once or increase the number of file descriptors available to **mysqld**.

To tell **mysqld** to keep open fewer files at a time, you can make the table cache smaller by reducing the value of the `table_cache` system variable (the default value is 64). Reducing the value of `max_connections` also reduces the number of open files (the default value is 100).

To change the number of file descriptors available to **mysqld**, you can use the `-`

`-open-files-limit` option to **mysqld_safe** or (as of MySQL 3.23.30) set the `open_files_limit` system variable. See Section 5.2.3, “Server System Variables”. The easiest way to set these values is to add an option to your option file. See Section 4.3.2, “Using Option Files”. If you have an old version of **mysqld** that doesn't support setting the open files limit, you can edit the **mysqld_safe** script. There is a commented-out line **ulimit -n 256** in the script. You can remove the '#' character to uncomment this line, and change the number `256` to set the number of file descriptors to be made available to **mysqld**.

`--open-files-limit` and **ulimit** can increase the number of file descriptors, but only up to the limit imposed by the operating system. There is also a ``hard'' limit that can be overridden only if you start **mysqld_safe** or **mysqld** as `root` (just remember that you also need to start the server with the `--user` option in this case so that it does not continue to run as `root` after it starts up). If you need to increase the operating system limit on the number of file descriptors available to each process, consult the documentation for your system.

Note: If you run the **tcsh** shell, **ulimit** does not work! **tcsh** also reports incorrect values when you ask for the current limits. In this case, you should start **mysqld_safe** using **sh**.

A.3. Installation-Related Issues

A.3.1. Problems Linking to the MySQL Client Library

When you are linking an application program to use the MySQL client library, you might get undefined reference errors for symbols that start with `mysql_`, such as those shown here:

```
/tmp/ccFKsdPa.o: In function `main':
/tmp/ccFKsdPa.o(.text+0xb): undefined reference to `mysql_init'
/tmp/ccFKsdPa.o(.text+0x31): undefined reference to `mysql_real_connect'
/tmp/ccFKsdPa.o(.text+0x57): undefined reference to `mysql_real_connect'
/tmp/ccFKsdPa.o(.text+0x69): undefined reference to `mysql_error'
/tmp/ccFKsdPa.o(.text+0x9a): undefined reference to `mysql_close'
```

You should be able to solve this problem by adding `-Ldir_path -lmysqlclient` at the end of your link command, where `dir_path` represents the pathname of the directory where the client library is located. To determine the correct directory, try this command:

```
shell> mysql_config --libs
```

The output from **mysql_config** might indicate other libraries that should be specified on the link command as well.

If you get `undefined reference` errors for the `uncompress` or `compress` function, add `-lz` to the end of your link command and try again.

If you get `undefined reference` errors for a function that should exist on your system, such as `connect`, check the manual page for the function in question to determine which libraries you should add to the link command.

You might get `undefined reference` errors such as the following for functions that don't exist on your system:

```
mf_format.o(.text+0x201): undefined reference to `__lxstat'
```

This usually means that your MySQL client library was compiled on a system that is not 100% compatible with yours. In this case, you should download the latest MySQL source distribution and compile MySQL yourself. See Section 2.8, “MySQL Installation Using a Source Distribution”.

You might get undefined reference errors at runtime when you try to execute a MySQL program. If these errors specify symbols that start with `mysql_` or indicate that the `mysqlclient` library can't be found, it means that your system can't find the shared `libmysqlclient.so` library. The fix for this is to tell your system to search for shared libraries where the library is located. Use whichever of the following methods is appropriate for your system:

- Add the path to the directory where `libmysqlclient.so` is located to the `LD_LIBRARY_PATH` environment variable.
- Add the path to the directory where `libmysqlclient.so` is located to the `LD_LIBRARY` environment variable.
- Copy `libmysqlclient.so` to some directory that is searched by your system, such as `/lib`, and update the shared library information by executing `ldconfig`.

Another way to solve this problem is by linking your program statically with the `-static` option, or by removing the dynamic MySQL libraries before linking your code. Before trying the second method, you should be sure that no other programs are using the dynamic libraries.

A.3.2. How to Run MySQL as a Normal User

On Windows, you can run the server as a Windows service using normal user accounts beginning with MySQL 4.0.17 and 4.1.2. (Older MySQL versions required you to have administrator rights. This was a bug introduced in MySQL 3.23.54.)

On Unix, the MySQL server **mysqld** can be started and run by any user. However, you should avoid running the server as the Unix `root` user for security reasons. In order to change **mysqld** to run as a normal unprivileged Unix user *`user_name`*, you must do the following:

1. Stop the server if it's running (use **mysqladmin shutdown**).
2. Change the database directories and files so that *`user_name`* has privileges to read and write files in them (you might need to do this as the Unix `root` user):

   ```
   shell> chown -R user_name /path/to/mysql/datadir
   ```

   If you do not do this, the server is not able to access databases or tables when it runs as *`user_name`*.

   If directories or files within the MySQL data directory are symbolic links, you'll also need to follow those links and change the directories and files they point to. `chown -R` might not follow symbolic links for you.
3. Start the server as user *`user_name`*. If you are using MySQL 3.22 or later, another alternative is to start **mysqld** as the Unix `root` user and use the `--user=`*`user_name`* option. **mysqld** starts up, then switches to run as the Unix user *`user_name`* before accepting any connections.
4. To start the server as the given user automatically at system startup time, specify the username by adding a `user` option to the `[mysqld]` group of the `/etc/my.cnf` option file or the `my.cnf` option file in the server's data directory. For example:

   ```
   [mysqld]
   user=user_name
   ```

If your Unix machine itself isn't secured, you should assign passwords to the MySQL `root` accounts in the grant tables. Otherwise, any user with a login account on that machine can run the **mysql** client with a `--user=root` option and perform any operation. (It is a good idea to assign passwords to MySQL accounts in any case, but especially so when other login accounts exist on the server host.) See Section 2.9, “Post-Installation Setup and Testing”.

A.3.3. Problems with File Permissions

If you have problems with file permissions, the `UMASK` environment variable might be set incorrectly when **mysqld** starts. For example, MySQL might issue the following error message when you create a table:

```
ERROR: Can't find file: 'path/with/filename.frm' (Errcode: 13)
```

The default `UMASK` value is `0660`. You can change this behavior by starting **mysqld_safe** as follows:

```
shell> UMASK=384  # = 600 in octal
shell> export UMASK
shell> mysqld_safe &
```

By default, MySQL creates database and `RAID` directories with an access permission value of `0700`. You can modify this behavior by setting the `UMASK_DIR` variable. If you set its value, new directories are created with the combined `UMASK` and `UMASK_DIR` values. For example, if you want to give group access to all new directories, you can do this:

```
shell> UMASK_DIR=504  # = 770 in octal
shell> export UMASK_DIR
shell> mysqld_safe &
```

In MySQL 3.23.25 and above, MySQL assumes that the value for `UMASK` and `UMASK_DIR` is in octal if it starts with a zero.

See Appendix F, *Environment Variables*.

A.4. Administration-Related Issues

A.4.1. How to Reset the Root Password

If you have never set a `root` password for MySQL, the server does not require a password at all for connecting as `root`. However, it is recommended to set a password for each account. See Section 5.4.1, “General Security Guidelines”.

If you set a `root` password previously, but have forgotten what it was, you can set a new password. The following procedure is for Windows systems. The procedure for Unix systems is given later in this section.

The procedure under Windows:

1. Log on to your system as Administrator.
2. Stop the MySQL server if it is running. For a server that is running as a Windows service, go to the Services manager:

```
Start Menu -> Control Panel -> Administrative Tools -> Services
```

Then find the MySQL service in the list, and stop it.

If your server is not running as a service, you may need to use the Task Manager to force it to stop.

3. Open a console window to get to the DOS command prompt:

```
Start Menu -> Run -> cmd
```

4. We are assuming that you installed MySQL to `C:\mysql`. If you installed MySQL to another location, adjust the following commands accordingly.

   At the DOS command prompt, execute this command:

```
C:\> C:\mysql\bin\mysqld-nt --skip-grant-tables
```

   This starts the server in a special mode that does not check the grant tables to control access.

5. Keeping the first console window open, open a second console window and execute the following commands (type each on a single line):

```
C:\> C:\mysql\bin\mysqladmin -u root
         flush-privileges password "newpwd"
C:\> C:\mysql\bin\mysqladmin -u root -p shutdown
```

   Replace ``*newpwd*" with the actual `root` password that you want to use. The second command prompts you to enter the new password for access. Enter the password that you assigned in the first command.

6. Stop the MySQL server, then restart it in normal mode again. If you run the server as a service, start it from the Windows Services window. If you start the server manually, use whatever command you normally use.

7. You should be able to connect using the new password.

In a Unix environment, the procedure for resetting the `root` password is as follows:

1. Log on to your system as either the Unix `root` user or as the same user that the **mysqld** server runs as.

2. Locate the `.pid` file that contains the server's process ID. The exact location and name of this file depend on your distribution, hostname, and configuration. Common locations are `/var/lib/mysql/`, `/var/run/mysqld/`, and `/usr/local/mysql/data/`. Generally, the filename has the extension of `.pid` and begins with either `mysqld` or your system's hostname.

   You can stop the MySQL server by sending a normal `kill` (not `kill -9`) to the **mysqld** process, using the pathname of the `.pid` file in the following command:

```
shell> kill `cat /mysql-data-directory/host_name.pid`
```

   Note the use of backticks rather than forward quotes with the `cat` command; these cause the output of `cat` to be substituted into the `kill` command.

3. Restart the MySQL server with the special `--skip-grant-tables` option:

```
shell> mysqld_safe --skip-grant-tables &
```

4. Set a new password for the `root@localhost` MySQL account:

   ```
   shell> mysqladmin -u root flush-privileges password "newpwd"
   ```

   Replace ``*newpwd*'' with the actual `root` password that you want to use.

5. You should be able to connect using the new password.

Alternatively, on any platform, you can set the new password using the **mysql** client:

1. Stop **mysqld** and restart it with the `--skip-grant-tables` option as described earlier.
2. Connect to the **mysqld** server with this command:

   ```
   shell> mysql -u root
   ```

3. Issue the following statements in the **mysql** client:

   ```
   mysql> UPDATE mysql.user SET Password=PASSWORD('newpwd')
       ->                   WHERE User='root';
   mysql> FLUSH PRIVILEGES;
   ```

   Replace ``*newpwd*'' with the actual `root` password that you want to use.

4. You should be able to connect using the new password.

A.4.2. What to Do If MySQL Keeps Crashing

Each MySQL version is tested on many platforms before it is released. This doesn't mean that there are no bugs in MySQL, but if there are bugs, they should be very few and can be hard to find. If you have a problem, it always helps if you try to find out exactly what crashes your system, because you have a much better chance of getting the problem fixed quickly.

First, you should try to find out whether the problem is that the **mysqld** server dies or whether your problem has to do with your client. You can check how long your **mysqld** server has been up by executing **mysqladmin version**. If **mysqld** has died and restarted, you may find the reason by looking in the server's error log. See Section 5.9.1, “The Error Log”.

On some systems, you can find in the error log a stack trace of where **mysqld** died that you can resolve with the `resolve_stack_dump` program. See Section E.1.4, “Using a Stack Trace”. Note that the variable values written in the error log may not always be 100% correct.

Many server crashes are caused by corrupted data files or index files. MySQL updates the files on disk with the `write()` system call after every SQL statement and before the client is notified about the result. (This is not true if you are running with `--delay-key-write`, in which case data files are written but not index files.) This means that data file contents are safe even if **mysqld** crashes, because the operating system ensures that the unflushed data is written to disk. You can force MySQL to flush everything to disk after every SQL statement by starting **mysqld** with the `--flush` option.

The preceding means that normally you should not get corrupted tables unless one of the following happens:

- The MySQL server or the server host was killed in the middle of an update.
- You have found a bug in **mysqld** that caused it to die in the middle of an update.
- Some external program is manipulating data files or index files at the same time as **mysqld** without locking the table properly.
- You are running many **mysqld** servers using the same data directory on a system that doesn't support good filesystem locks (normally handled by the `lockd` lock manager), or you are running multiple servers with the `--skip-external-locking` option.
- You have a crashed data file or index file that contains very corrupt data that confused **mysqld**.
- You have found a bug in the data storage code. This isn't likely, but it's at least possible. In this case, you can try to change the table type to another storage engine by using `ALTER TABLE` on a repaired copy of the table.

Because it is very difficult to know why something is crashing, first try to check whether things that work for others crash for you. Please try the following things:

- Stop the **mysqld** server with **mysqladmin shutdown**, run **myisamchk --silent --force */*.MYI** from the data directory to check all `MyISAM` tables, and restart **mysqld**. This ensures that you are running from a clean state. See Chapter 5, *Database Administration*.
- Start **mysqld** with the `--log` option and try to determine from the information written to the log whether some specific query kills the server. About 95% of all bugs are related to a particular query. Normally, this is one of the last queries in the log file just before the server restarts. See Section 5.9.2, “The General Query Log”. If you can repeatedly kill MySQL with a specific query, even when you have checked all tables just before issuing it, then you have been able to locate the bug and should submit a bug report for it. See Section 1.4.1.3, “How to Report Bugs or Problems”.
- Try to make a test case that we can use to repeat the problem. See Section E.1.6, “Making a Test Case If You Experience Table Corruption”.
- Try running the tests in the `mysql-test` directory and the MySQL benchmarks. See Section 25.1.2, “MySQL Test Suite”. They should test MySQL rather well. You can also add code to the benchmarks that simulates your application. The benchmarks can be found in the `sql-bench` directory in a source distribution or, for a binary distribution, in the `sql-bench` directory under your MySQL installation directory.
- Try the `fork_big.pl` script. (It is located in the `tests` directory of source distributions.)
- If you configure MySQL for debugging, it is much easier to gather information about possible errors if something goes wrong. Configuring MySQL for debugging causes a safe memory allocator to be included that can find some errors. It also provides a lot of output about what is happening. Reconfigure MySQL with the `--with-debug` or `--with-debug=full` option to **configure** and then recompile. See Section E.1, “Debugging a MySQL Server”.
- Make sure that you have applied the latest patches for your operating system.
- Use the `--skip-external-locking` option to **mysqld**. On some systems, the `lockd` lock manager does not work properly; the `--skip-external-locking` option tells **mysqld** not to use external locking. (This means that you cannot run two **mysqld** servers on the same data directory and that you must be careful if you use **myisamchk**. Nevertheless, it may be instructive to try the option as a test.)
- Have you tried **mysqladmin -u root processlist** when **mysqld** appears to be running but not re-

sponding? Sometimes **mysqld** is not comatose even though you might think so. The problem may be that all connections are in use, or there may be some internal lock problem. **mysqladmin -u root processlist** usually is able to make a connection even in these cases, and can provide useful information about the current number of connections and their status.

- Run the command **mysqladmin -i 5 status** or **mysqladmin -i 5 -r status** in a separate window to produce statistics while you run your other queries.
- Try the following:

  1. Start **mysqld** from **gdb** (or another debugger). See Section E.1.3, “Debugging **mysqld** under **gdb**”.
  2. Run your test scripts.
  3. Print the backtrace and the local variables at the three lowest levels. In **gdb**, you can do this with the following commands when **mysqld** has crashed inside **gdb**:

     ```
     backtrace
     info local
     up
     info local
     up
     info local
     ```

     With **gdb**, you can also examine which threads exist with `info threads` and switch to a specific thread with `thread #`, where # is the thread ID.
- Try to simulate your application with a Perl script to force MySQL to crash or misbehave.
- Send a normal bug report. See Section 1.4.1.3, “How to Report Bugs or Problems”. Be even more detailed than usual. Because MySQL works for many people, it may be that the crash results from something that exists only on your computer (for example, an error that is related to your particular system libraries).
- If you have a problem with tables containing dynamic-length rows and you are using only `VARCHAR` columns (not `BLOB` or `TEXT` columns), you can try to change all `VARCHAR` to `CHAR` with `ALTER TABLE`. This forces MySQL to use fixed-size rows. Fixed-size rows take a little extra space, but are much more tolerant to corruption.

  The current dynamic row code has been in use at MySQL AB for several years with very few problems, but dynamic-length rows are by nature more prone to errors, so it may be a good idea to try this strategy to see whether it helps.
- Do not rule out your server hardware when diagnosing problems. Defective hardware can be the cause of data corruption. Particular attention should be paid to both RAMS and hard-drives when troubleshooting hardware.

A.4.3. How MySQL Handles a Full Disk

This section describes how MySQL responds to disk-full errors (such as ``no space left on device''), and, as of MySQL 4.0.22, to quota-exceeded errors (such as ``write failed'' or ``user block limit reached'').

This section is relevant for writes to `MyISAM` tables. As of MySQL 4.1.9, it also applies for writes to binary log files and binary log index file, except that references to ``row'' and ``record'' should be understood to mean ``event.''

When a disk-full condition occurs, MySQL does the following:

- It checks once every minute to see whether there is enough space to write the current row. If there is enough space, it continues as if nothing had happened.
- Every 10 minutes it writes an entry to the log file, warning about the disk-full condition.

To alleviate the problem, you can take the following actions:

- To continue, you only have to free enough disk space to insert all records.
- To abort the thread, you must use **mysqladmin kill**. The thread is aborted the next time it checks the disk (in one minute).
- Other threads might be waiting for the table that caused the disk-full condition. If you have several ``locked'' threads, killing the one thread that is waiting on the disk-full condition allows the other threads to continue.

Exceptions to the preceding behavior are when you use `REPAIR TABLE` or `OPTIMIZE TABLE` or when the indexes are created in a batch after `LOAD DATA INFILE` or after an `ALTER TABLE` statement. All of these statements may create large temporary files that, if left to themselves, would cause big problems for the rest of the system. If the disk becomes full while MySQL is doing any of these operations, it removes the big temporary files and mark the table as crashed. The exception is that for `ALTER TABLE`, the old table is left unchanged.

A.4.4. Where MySQL Stores Temporary Files

MySQL uses the value of the `TMPDIR` environment variable as the pathname of the directory in which to store temporary files. If you don't have `TMPDIR` set, MySQL uses the system default, which is normally `/tmp`, `/var/tmp`, or `/usr/tmp`. If the filesystem containing your temporary file directory is too small, you can use the `--tmpdir` option to **mysqld** to specify a directory in a filesystem where you have enough space.

Starting from MySQL 4.1, the `--tmpdir` option can be set to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters (`':'`) on Unix and semicolon characters (`';'`) on Windows, NetWare, and OS/2. **Note**: To spread the load effectively, these paths should be located on different *physical* disks, not different partitions of the same disk.

If the MySQL server is acting as a replication slave, you should not set `--tmpdir` to point to a directory on a memory-based filesystem or to a directory that is cleared when the server host restarts. A replication slave needs some of its temporary files to survive a machine restart so that it can replicate temporary tables or `LOAD DATA INFILE` operations. If files in the temporary file directory are lost when the server restarts, replication fails.

MySQL creates all temporary files as hidden files. This ensures that the temporary files are removed if **mysqld** is terminated. The disadvantage of using hidden files is that you do not see a big temporary file that fills up the filesystem in which the temporary file directory is located.

When sorting (`ORDER BY` or `GROUP BY`), MySQL normally uses one or two temporary files. The maximum disk space required is determined by the following expression:

```
(length of what is sorted + sizeof(row pointer))
* number of matched rows
* 2
```

The row pointer size is usually four bytes, but may grow in the future for really big tables.

For some `SELECT` queries, MySQL also creates temporary SQL tables. These are not hidden and have names of the form `SQL_*`.

`ALTER TABLE` creates a temporary table in the same directory as the original table.

A.4.5. How to Protect or Change the MySQL Socket File `/tmp/mysql.sock`

The default location for the Unix socket file that the server uses for communication with local clients is `/tmp/mysql.sock`. This might cause problems, because on some versions of Unix, anyone can delete files in the `/tmp` directory.

On most versions of Unix, you can protect your `/tmp` directory so that files can be deleted only by their owners or the superuser (`root`). To do this, set the `sticky` bit on the `/tmp` directory by logging in as `root` and using the following command:

```
shell> chmod +t /tmp
```

You can check whether the `sticky` bit is set by executing `ls -ld /tmp`. If the last permission character is `t`, the bit is set.

Another approach is to change the place where the server creates the Unix socket file. If you do this, you should also let client programs know the new location of the file. You can specify the file location in several ways:

- Specify the path in a global or local option file. For example, put the following lines in `/etc/my.cnf`:

  ```
  [mysqld]
  socket=/path/to/socket

  [client]
  socket=/path/to/socket
  ```

  See Section 4.3.2, “Using Option Files”.

- Specify a `--socket` option on the command line to **mysqld_safe** and when you run client programs.

- Set the `MYSQL_UNIX_PORT` environment variable to the path of the Unix socket file.

- Recompile MySQL from source to use a different default Unix socket file location. Define the path to the file with the `--with-unix-socket-path` option when you run **configure**. See Section 2.8.2, “Typical **configure** Options”.

You can test whether the new socket location works by attempting to connect to the server with this command:

```
shell> mysqladmin --socket=/path/to/socket version
```

A.4.6. Time Zone Problems

If you have a problem with `SELECT NOW()` returning values in GMT and not your local time, you have to tell the server your current time zone. The same applies if `UNIX_TIMESTAMP()` returns the wrong value. This should be done for the environment in which the server runs; for example, in **mysqld_safe** or **mysql.server**. See Appendix F, *Environment Variables*.

You can set the time zone for the server with the `--timezone=timezone_name` option to **mysqld_safe**. You can also set it by setting the `TZ` environment variable before you start **mysqld**.

The allowable values for `--timezone` or `TZ` are system-dependent. Consult your operating system documentation to see what values are acceptable.

A.5. Query-Related Issues

A.5.1. Case Sensitivity in Searches

By default, MySQL searches are not case sensitive (although there are some character sets that are never case insensitive, such as `czech`). This means that if you search with *`col_name`* `LIKE 'a%'`, you get all column values that start with `A` or `a`. If you want to make this search case sensitive, make sure that one of the operands has a case sensitive or binary collation. For example, if you are comparing a column and a string that both have the `latin1` character set, you can use the `COLLATE` operator to cause either operand to have the `latin1_general_cs` or `latin1_bin` collation. For example:

```
col_name COLLATE latin1_general_cs LIKE 'a%'
col_name LIKE 'a%' COLLATE latin1_general_cs
col_name COLLATE latin1_bin LIKE 'a%'
col_name LIKE 'a%' COLLATE latin1_bin
```

If you want a column always to be treated in case-sensitive fashion, declare it with a case sensitive or binary collation. See Section 13.2.6, “`CREATE TABLE` Syntax”.

Before MySQL 4.1, `COLLATE` is unavailable. Use the `BINARY` operator in expressions to treat a string as a binary string: `BINARY` *`col_name`* `LIKE 'a%'` or *`col_name`* `LIKE BINARY 'a%'`. In column declarations, use the `BINARY` attribute.

Simple comparison operations (`>=, >, =, <, <=`, sorting, and grouping) are based on each character's ``sort value.'' Characters with the same sort value (such as 'E', 'e', and 'é') are treated as the same character.

A.5.2. Problems Using `DATE` Columns

The format of a `DATE` value is `'YYYY-MM-DD'`. According to standard SQL, no other format is allowed. You should use this format in `UPDATE` expressions and in the `WHERE` clause of `SELECT` statements. For example:

```
mysql> SELECT * FROM tbl_name WHERE date >= '2003-05-05';
```

As a convenience, MySQL automatically converts a date to a number if the date is used in a numeric context (and vice versa). It is also smart enough to allow a ``relaxed'' string form when updating and in a `WHERE` clause that compares a date to a `TIMESTAMP`, `DATE`, or `DATETIME` column. (``Relaxed form'' means that any punctuation character may be used as the separator between parts. For example, `'2004-08-15'` and `'2004#08#15'` are equivalent.) MySQL can also convert a string containing no separators (such as `'20040815'`), provided it makes sense as a date.

The special date `'0000-00-00'` can be stored and retrieved as `'0000-00-00'`. When using a `'0000-00-00'` date through MyODBC, it is automatically converted to `NULL` in MyODBC 2.50.12

and above, because ODBC can't handle this kind of date.

Because MySQL performs the conversions described above, the following statements work:

```
mysql> INSERT INTO tbl_name (idate) VALUES (19970505);
mysql> INSERT INTO tbl_name (idate) VALUES ('19970505');
mysql> INSERT INTO tbl_name (idate) VALUES ('97-05-05');
mysql> INSERT INTO tbl_name (idate) VALUES ('1997.05.05');
mysql> INSERT INTO tbl_name (idate) VALUES ('1997 05 05');
mysql> INSERT INTO tbl_name (idate) VALUES ('0000-00-00');

mysql> SELECT idate FROM tbl_name WHERE idate >= '1997-05-05';
mysql> SELECT idate FROM tbl_name WHERE idate >= 19970505;
mysql> SELECT MOD(idate,100) FROM tbl_name WHERE idate >= 19970505;
mysql> SELECT idate FROM tbl_name WHERE idate >= '19970505';
```

However, the following does not work:

```
mysql> SELECT idate FROM tbl_name WHERE STRCMP(idate,'20030505')=0;
```

`STRCMP()` is a string function, so it converts `idate` to a string in `'YYYY-MM-DD'` format and performs a string comparison. It does not convert `'20030505'` to the date `'2003-05-05'` and perform a date comparison.

If you are using the `ALLOW_INVALID_DATES` SQL mode, MySQL allows you to store dates that are given only limited checking: MySQL ensures only that the day is in the range from 1 to 31 and the month is in the range from 1 to 12.

This makes MySQL very convenient for Web applications where you obtain year, month, and day in three different fields and you want to store exactly what the user inserted (without date validation).

If you are not using the `NO_ZERO_IN_DATE` SQL mode, the day or month part can be zero. This is convenient if you want to store a birthdate in a `DATE` column and you know only part of the date.

If you are not using the `NO_ZERO_DATE` SQL mode, MySQL also allows you to store `'0000-00-00'` as a ``dummy date." This is in some cases more convenient than using `NULL` values.

If the date cannot be converted to any reasonable value, a `0` is stored in the `DATE` column, which is retrieved as `'0000-00-00'`. This is both a speed and a convenience issue. We believe that the database server's responsibility is to retrieve the same date you stored (even if the data was not logically correct in all cases). We think it is up to the application and not the server to check the dates.

If you want MySQL to check all dates and accept only legal dates (unless overriden by IGNORE), you should set `sql_mode` to `"NO_ZERO_IN_DATE,NO_ZERO_DATE"`.

Date handling in MySQL 5.0.1 and earlier works like MySQL 5.0.2 with the `ALLOW_INVALID_DATES` SQL mode enabled.

A.5.3. Problems with NULL Values

The concept of the `NULL` value is a common source of confusion for newcomers to SQL, who often think that `NULL` is the same thing as an empty string `''`. This is not the case. For example, the following statements are completely different:

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);
mysql> INSERT INTO my_table (phone) VALUES ('');
```

Both statements insert a value into the `phone` column, but the first inserts a `NULL` value and the second

inserts an empty string. The meaning of the first can be regarded as ``phone number is not known'' and the meaning of the second can be regarded as ``the person is known to have no phone, and thus no phone number.''

To help with NULL handling, you can use the IS NULL and IS NOT NULL operators and the IFNULL() function.

In SQL, the NULL value is never true in comparison to any other value, even NULL. An expression that contains NULL always produces a NULL value unless otherwise indicated in the documentation for the operators and functions involved in the expression. All columns in the following example return NULL:

```
mysql> SELECT NULL, 1+NULL, CONCAT('Invisible',NULL);
```

If you want to search for column values that are NULL, you cannot use an expr = NULL test. The following statement returns no rows, because expr = NULL is never true for any expression:

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

To look for NULL values, you must use the IS NULL test. The following statements show how to find the NULL phone number and the empty phone number:

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;
mysql> SELECT * FROM my_table WHERE phone = '';
```

You can add an index on a column that can have NULL values if you are using MySQL 3.23.2 or newer and are using the MyISAM, InnoDB, or BDB storage engine. As of MySQL 4.0.2, the MEMORY storage engine also supports NULL values in indexes. Otherwise, you must declare an indexed column NOT NULL and you cannot insert NULL into the column.

When reading data with LOAD DATA INFILE, empty or missing columns are updated with ''. If you want a NULL value in a column, you should use \N in the data file. The literal word ``NULL'' may also be used under some circumstances. See Section 13.1.5, “LOAD DATA INFILE Syntax”.

When using DISTINCT, GROUP BY, or ORDER BY, all NULL values are regarded as equal.

When using ORDER BY, NULL values are presented first, or last if you specify DESC to sort in descending order. Exception: In MySQL 4.0.2 through 4.0.10, NULL values sort first regardless of sort order.

Aggregate (summary) functions such as COUNT(), MIN(), and SUM() ignore NULL values. The exception to this is COUNT(*), which counts rows and not individual column values. For example, the following statement produces two counts. The first is a count of the number of rows in the table, and the second is a count of the number of non-NULL values in the age column:

```
mysql> SELECT COUNT(*), COUNT(age) FROM person;
```

For some column types, MySQL handles NULL values specially. If you insert NULL into a TIMESTAMP column, the current date and time is inserted. If you insert NULL into an integer column that has the AUTO_INCREMENT attribute, the next number in the sequence is inserted.

A.5.4. Problems with Column Aliases

You can use an alias to refer to a column in GROUP BY, ORDER BY, or HAVING clauses. Aliases can also be used to give columns better names:

```
SELECT SQRT(a*b) AS root FROM tbl_name GROUP BY root HAVING root > 0;
SELECT id, COUNT(*) AS cnt FROM tbl_name GROUP BY id HAVING cnt > 0;
```

```
SELECT id AS 'Customer identity' FROM tbl_name;
```

Standard SQL doesn't allow you to refer to a column alias in a WHERE clause. This is because when the WHERE code is executed, the column value may not yet be determined. For example, the following query is illegal:

```
SELECT id, COUNT(*) AS cnt FROM tbl_name WHERE cnt > 0 GROUP BY id;
```

The WHERE statement is executed to determine which rows should be included in the GROUP BY part, whereas HAVING is used to decide which rows from the result set should be used.

A.5.5. Rollback Failure for Non-Transactional Tables

If you receive the following message when trying to perform a ROLLBACK, it means that one or more of the tables you used in the transaction do not support transactions:

```
Warning: Some non-transactional changed tables couldn't be rolled back
```

These non-transactional tables are not affected by the ROLLBACK statement.

If you were not deliberately mixing transactional and non-transactional tables within the transaction, the most likely cause for this message is that a table you thought was transactional actually is not. This can happen if you try to create a table using a transactional storage engine that is not supported by your **mysqld** server (or that was disabled with a startup option). If **mysqld** doesn't support a storage engine, it instead creates the table as a MyISAM table, which is non-transactional.

You can check the table type for a table by using either of these statements:

```
SHOW TABLE STATUS LIKE 'tbl_name';
SHOW CREATE TABLE tbl_name;
```

See Section 13.5.4.17, “SHOW TABLE STATUS Syntax” and Section 13.5.4.5, “SHOW CREATE TABLE Syntax”.

You can check which storage engines your **mysqld** server supports by using this statement:

```
SHOW ENGINES;
```

Before MySQL 4.1.2, SHOW ENGINES is unavailable. Use the following statement instead and check the value of the variable that is associated with the storage engine in which you are interested:

```
SHOW VARIABLES LIKE 'have_%';
```

For example, to determine whether the InnoDB storage engine is available, check the value of the have_innodb variable.

See Section 13.5.4.8, “SHOW ENGINES Syntax” and Section 13.5.4.19, “SHOW VARIABLES Syntax”.

A.5.6. Deleting Rows from Related Tables

MySQL does not support subqueries prior to Version 4.1, or the use of more than one table in the DELETE statement prior to Version 4.0. If your version of MySQL does not support subqueries or multiple-table DELETE statements, you can use the following approach to delete rows from two related tables:

1. SELECT the rows based on some WHERE condition in the main table.
2. DELETE the rows in the main table based on the same condition.
3. DELETE FROM related_table WHERE related_column IN (selected_rows).

If the total length of the DELETE statement for related_table is more than 1MB (the default value of the max_allowed_packet system variable), you should split it into smaller parts and execute multiple DELETE statements. You probably get the fastest DELETE by specifying only 100 to 1,000 related_column values per statement if the related_column is indexed. If the related_column isn't indexed, the speed is independent of the number of arguments in the IN clause.

A.5.7. Solving Problems with No Matching Rows

If you have a complicated query that uses many tables but that doesn't return any rows, you should use the following procedure to find out what is wrong:

1. Test the query with EXPLAIN to check whether you can find something that is obviously wrong. See Section 7.2.1, “EXPLAIN Syntax (Get Information About a SELECT)”.
2. Select only those columns that are used in the WHERE clause.
3. Remove one table at a time from the query until it returns some rows. If the tables are large, it's a good idea to use LIMIT 10 with the query.
4. Issue a SELECT for the column that should have matched a row against the table that was last removed from the query.
5. If you are comparing FLOAT or DOUBLE columns with numbers that have decimals, you can't use equality (=) comparisons. This problem is common in most computer languages because not all floating-point values can be stored with exact precision. In some cases, changing the FLOAT to a DOUBLE fixes this. See Section A.5.8, “Problems with Floating-Point Comparisons”.
6. If you still can't figure out what's wrong, create a minimal test that can be run with mysql test < query.sql that shows your problems. You can create a test file by dumping the tables with **mysqldump --quick db_name *tbl_name_1* ... *tbl_name_n* > query.sql**. Open the file in an editor, remove some insert lines (if there are more than needed to demonstrate the problem), and add your SELECT statement at the end of the file.

   Verify that the test file demonstrates the problem by executing these commands:

   ```
   shell> mysqladmin create test2
   shell> mysql test2 < query.sql
   ```

   Post the test file using **mysqlbug** to the general MySQL mailing list. See Section 1.4.1.1, “The MySQL Mailing Lists”.

A.5.8. Problems with Floating-Point Comparisons

Floating-point numbers sometimes cause confusion because they are not stored as exact values inside computer architecture. What you can see on the screen usually is not the exact value of the number. The column types FLOAT, DOUBLE, and DECIMAL are such. DECIMAL columns store values with exact precision because they are represented as strings, but calculations on DECIMAL values may be done using floating-point operations.

The following example demonstrate the problem. It shows that even for the `DECIMAL` column type, calculations that are done using floating-point operations are subject to floating-point error.

```
mysql> CREATE TABLE t1 (i INT, d1 DECIMAL(9,2), d2 DECIMAL(9,2));
mysql> INSERT INTO t1 VALUES (1, 101.40, 21.40), (1, -80.00, 0.00),
    -> (2, 0.00, 0.00), (2, -13.20, 0.00), (2, 59.60, 46.40),
    -> (2, 30.40, 30.40), (3, 37.00, 7.40), (3, -29.60, 0.00),
    -> (4, 60.00, 15.40), (4, -10.60, 0.00), (4, -34.00, 0.00),
    -> (5, 33.00, 0.00), (5, -25.80, 0.00), (5, 0.00, 7.20),
    -> (6, 0.00, 0.00), (6, -51.40, 0.00);

mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b
    -> FROM t1 GROUP BY i HAVING a <> b;
+------+--------+-------+
| i    | a      | b     |
+------+--------+-------+
|    1 |  21.40 | 21.40 |
|    2 |  76.80 | 76.80 |
|    3 |   7.40 |  7.40 |
|    4 |  15.40 | 15.40 |
|    5 |   7.20 |  7.20 |
|    6 | -51.40 |  0.00 |
+------+--------+-------+
```

The result is correct. Although the first five records look like they shouldn't pass the comparison test (the values of `a` and `b` do not appear to be different), they may do so because the difference between the numbers shows up around the tenth decimal or so, depending on computer architecture.

The problem cannot be solved by using `ROUND()` or similar functions, because the result is still a floating-point number:

```
mysql> SELECT i, ROUND(SUM(d1), 2) AS a, ROUND(SUM(d2), 2) AS b
    -> FROM t1 GROUP BY i HAVING a <> b;
+------+--------+-------+
| i    | a      | b     |
+------+--------+-------+
|    1 |  21.40 | 21.40 |
|    2 |  76.80 | 76.80 |
|    3 |   7.40 |  7.40 |
|    4 |  15.40 | 15.40 |
|    5 |   7.20 |  7.20 |
|    6 | -51.40 |  0.00 |
+------+--------+-------+
```

This is what the numbers in column `a` look like when displayed with more decimal places:

```
mysql> SELECT i, ROUND(SUM(d1), 2)*1.0000000000000000 AS a,
    -> ROUND(SUM(d2), 2) AS b FROM t1 GROUP BY i HAVING a <> b;
+------+----------------------+-------+
| i    | a                    | b     |
+------+----------------------+-------+
|    1 |  21.3999999999999986 | 21.40 |
|    2 |  76.7999999999999972 | 76.80 |
|    3 |   7.4000000000000004 |  7.40 |
|    4 |  15.4000000000000004 | 15.40 |
|    5 |   7.2000000000000002 |  7.20 |
|    6 | -51.3999999999999986 |  0.00 |
+------+----------------------+-------+
```

Depending on your computer architecture, you may or may not see similar results. Different CPUs may

evaluate floating-point numbers differently. For example, on some machines you may get the ``correct'' results by multiplying both arguments by 1, as the following example shows.

Warning: Never use this method in your applications. It is not an example of a trustworthy method!

```
mysql> SELECT i, ROUND(SUM(d1), 2)*1 AS a, ROUND(SUM(d2), 2)*1 AS b
    -> FROM t1 GROUP BY i HAVING a <> b;
+------+--------+------+
| i    | a      | b    |
+------+--------+------+
|    6 | -51.40 | 0.00 |
+------+--------+------+
```

The reason that the preceding example seems to work is that on the particular machine where the test was done, CPU floating-point arithmetic happens to round the numbers to the same value. However, there is no rule that any CPU should do so, so this method cannot be trusted.

The correct way to do floating-point number comparison is to first decide on an acceptable tolerance for differences between the numbers and then do the comparison against the tolerance value. For example, if we agree that floating-point numbers should be regarded the same if they are same within a precision of one in ten thousand (0.0001), the comparison should be written to find differences larger than the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
    -> GROUP BY i HAVING ABS(a - b) > 0.0001;
+------+--------+------+
| i    | a      | b    |
+------+--------+------+
|    6 | -51.40 | 0.00 |
+------+--------+------+
1 row in set (0.00 sec)
```

Conversely, to get rows where the numbers are the same, the test should find differences within the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
    -> GROUP BY i HAVING ABS(a - b) <= 0.0001;
+------+-------+-------+
| i    | a     | b     |
+------+-------+-------+
|    1 | 21.40 | 21.40 |
|    2 | 76.80 | 76.80 |
|    3 |  7.40 |  7.40 |
|    4 | 15.40 | 15.40 |
|    5 |  7.20 |  7.20 |
+------+-------+-------+
```

A.6. Optimizer-Related Issues

MySQL uses a cost-based optimizer to determine the best way to resolve a query. In many cases, MySQL can calculate the best possible query plan, but sometimes MySQL doesn't have enough information about the data at hand and has to make ``educated'' guesses about the data.

For the cases when MySQL does not do the "right" thing, tools that you have available to help MySQL are:

- Use the EXPLAIN statement to get information about how MySQL processes a query. To use it, just add the keyword EXPLAIN to the front of your SELECT statement:

  ```
  mysql> EXPLAIN SELECT * FROM t1, t2 WHERE t1.i = t2.i;
  ```

  EXPLAIN is discussed in more detail in Section 7.2.1, "EXPLAIN Syntax (Get Information About a SELECT)".

- Use ANALYZE TABLE *tbl_name* to update the key distributions for the scanned table. See Section 13.5.2.1, "ANALYZE TABLE Syntax".

- Use FORCE INDEX for the scanned table to tell MySQL that table scans are very expensive compared to using the given index. See Section 13.1.7, "SELECT Syntax".

  ```
  SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
  WHERE t1.col_name=t2.col_name;
  ```

  USE INDEX and IGNORE INDEX may also be useful.

- Global and table-level STRAIGHT_JOIN. See Section 13.1.7, "SELECT Syntax".

- You can tune global or thread-specific system variables. For example, Start **mysqld** with the --max-seeks-for-key=1000 option or use SET max_seeks_for_key=1000 to tell the optimizer to assume that no key scan causes more than 1,000 key seeks. See Section 5.2.3, "Server System Variables".

A.7. Table Definition-Related Issues

A.7.1. Problems with ALTER TABLE

ALTER TABLE changes a table to the current character set. If you get a duplicate-key error during ALTER TABLE, the cause is either that the new character sets maps two keys to the same value or that the table is corrupted. In the latter case, you should run REPAIR TABLE on the table.

If ALTER TABLE dies with the following error, the problem may be that MySQL crashed during an earlier ALTER TABLE operation and there is an old table named A-*xxx* or B-*xxx* lying around:

```
Error on rename of './database/name.frm'
to './database/B-xxx.frm' (Errcode: 17)
```

In this case, go to the MySQL data directory and delete all files that have names starting with A- or B-. (You may want to move them elsewhere instead of deleting them.)

ALTER TABLE works in the following way:

- Create a new table named A-*xxx* with the requested structural changes.
- Copy all rows from the original table to A-*xxx*.
- Rename the original table to B-*xxx*.
- Rename A-*xxx* to your original table name.

- Delete B-*xxx*.

If something goes wrong with the renaming operation, MySQL tries to undo the changes. If something goes seriously wrong (although this shouldn't happen), MySQL may leave the old table as B-*xxx*. A simple rename of the table files at the system level should get your data back.

If you use ALTER TABLE on a transactional table or if you are using Windows or OS/2, ALTER TABLE unlocks the table if you had done a LOCK TABLE on it. This is because InnoDB and these operating systems cannot drop a table that is in use.

A.7.2. How to Change the Order of Columns in a Table

First, consider whether you really need to change the column order in a table. The whole point of SQL is to abstract the application from the data storage format. You should always specify the order in which you wish to retrieve your data. The first of the following statements returns columns in the order *col_name1*, *col_name2*, *col_name3*, whereas the second returns them in the order *col_name1*, *col_name3*, *col_name2*:

```
mysql> SELECT col_name1, col_name2, col_name3 FROM tbl_name;
mysql> SELECT col_name1, col_name3, col_name2 FROM tbl_name;
```

If you decide to change the order of table columns anyway, you can do so as follows:

1. Create a new table with the columns in the new order.
2. Execute this statement:

   ```
   mysql> INSERT INTO new_table
       -> SELECT columns-in-new-order FROM old_table;
   ```

3. Drop or rename old_table.
4. Rename the new table to the original name:

   ```
   mysql> ALTER TABLE new_table RENAME old_table;
   ```

SELECT * is quite suitable for testing queries. However, in an application, you should *never* rely on using SELECT * and retrieving the columns based on their position. The order and position in which columns are returned does not remain the same if you add, move, or delete columns. A simple change to your table structure could cause your application to fail.

A.7.3. TEMPORARY TABLE Problems

The following list indicates limitations on the use of TEMPORARY tables:

- A TEMPORARY table can only be of type HEAP, ISAM, MyISAM, MERGE, or InnoDB.
- You cannot refer to a TEMPORARY table more than once in the same query. For example, the following does not work:

  ```
  mysql> SELECT * FROM temp_table, temp_table AS t2;
  ERROR 1137: Can't reopen table: 'temp_table'
  ```

- The `SHOW TABLES` statement does not list `TEMPORARY` tables.
- You cannot use `RENAME` to rename a `TEMPORARY` table. However, you can use `ALTER TABLE` instead:

  ```
  mysql> ALTER TABLE orig_name RENAME new_name;
  ```

Appendix B. Credits

This appendix lists the developers, contributors, and supporters that have helped to make MySQL what it is today.

B.1. Developers at MySQL AB

These are the developers that are or have been employed by MySQL AB to work on the `MySQL` database software, roughly in the order they started to work with us. Following each developer is a small list of the tasks that the developer is responsible for, or the accomplishments they have made. All developers are involved in support.

- Michael (Monty) Widenius
  - Lead developer and main author of the MySQL server (**mysqld**).
  - New functions for the string library.
  - Most of the `mysys` library.
  - The `ISAM` and `MyISAM` libraries (B-tree index file handlers with index compression and different record formats).
  - The `HEAP` library. A memory table system with our superior full dynamic hashing. In use since 1981 and published around 1984.
  - The **replace** program (take a look at it, it's **COOL**!).
  - Connector/ODBC (MyODBC), the ODBC driver for Windows.
  - Fixing bugs in MIT-pthreads to get it to work for MySQL Server. And also Unireg, a curses-based application tool with many utilities.
  - Porting of `mSQL` tools like `msqlperl`, `DBD/DBI`, and `DB2mysql`.
  - Most of `crash-me` and the foundation for the MySQL benchmarks.
- David Axmark
  - Initial main writer of the **Reference Manual**, including enhancements to **texi2html**.
  - Automatic Web site updating from the manual.
  - Initial Autoconf, Automake, and Libtool support.
  - Licensing.
  - Parts of all the text files. (Nowadays only the `README` is left. The rest ended up in the manual.)
  - Lots of testing of new features.
  - Our in-house Free Software legal expert.
  - Mailing list maintainer (who never has the time to do it right...).
  - Our original portability code (now more than 10 years old). Nowadays only some parts of

  `mysys` are left.
  - Someone for Monty to call in the middle of the night when he just got that new feature to work.
  - Chief "Open Sourcerer" (MySQL community relations).
- Jani Tolonen
  - **mysqlimport**
  - A lot of extensions to the command-line clients.
  - `PROCEDURE ANALYSE()`
- Sinisa Milivojevic
  - Compression (with `zlib`) in the client/server protocol.
  - Perfect hashing for the lexical analyzer phase.
  - Multi-row `INSERT`
  - **mysqldump** -e option
  - `LOAD DATA LOCAL INFILE`
  - `SQL_CALC_FOUND_ROWS` `SELECT` option
  - `--max-user-connections=...` option
  - `net_read` and `net_write_timeout`
  - `GRANT`/`REVOKE` and `SHOW GRANTS FOR`
  - New client/server protocol for 4.0
  - `UNION` in 4.0
  - Multiple-table `DELETE`/`UPDATE`
  - Derived tables in 4.1
  - User resources management
  - Initial developer of the `MySQL++` C++ API and the `MySQLGUI` client.
- Tonu Samuel (past developer)
  - VIO interface (the foundation for the encrypted client/server protocol).
  - MySQL Filesystem (a way to use MySQL databases as files and directories).
  - The `CASE` expression.
  - The `MD5()` and `COALESCE()` functions.
  - `RAID` support for `MyISAM` tables.
- Sasha Pachev

  - Initial implementation of replication (up to version 4.0).
  - `SHOW CREATE TABLE`.
  - `mysql-bench`
- Matt Wagner
  - MySQL test suite.
  - Webmaster (until 2002).
  - Coordinator of development.
- Miguel Solorzano
  - Win32 development and release builds.
  - Windows NT server code.
  - WinMySQLAdmin
- Timothy Smith (past developer)
  - Dynamic character sets support.
  - configure, RPMs and other parts of the build system.
  - Initial developer of `libmysqld`, the embedded server.
- Sergei Golubchik
  - Full-text search.
  - Added keys to the `MERGE` library.
- Jeremy Cole
  - Proofreading and editing this fine manual.
  - `ALTER TABLE ... ORDER BY ....`
  - `UPDATE ... ORDER BY ....`
  - `DELETE ... ORDER BY ....`
- Indrek Siitan
  - Designing/programming of our Web interface.
  - Author of our newsletter management system.
- Jorge del Conde
  - **MySQLCC** (**MySQL Control Center**)
  - Win32 development
  - Initial implementation of the Web site portals.

- Venu Anuganti
  - MyODBC 3.51
  - New client/server protocol for 4.1 (for prepared statements).
- Arjen Lentz
  - Maintainer of the MySQL Reference Manual.
  - Preparing the O'Reilly printed edition of the manual.
- Alexander (Bar) Barkov, Alexey (Holyfoot) Botchkov, and Ramil Kalimullin
  - Spatial data (GIS) and R-Trees implementation for 4.1
  - Unicode and character sets for 4.1; documentation for same
- Oleksandr (Sanja) Byelkin
  - Query cache in 4.0
  - Implementation of subqueries (4.1).
- Aleksey (Walrus) Kishkin and Alexey (Ranger) Stroganov
  - Benchmarks design and analysis.
  - Maintenance of the MySQL test suite.
- Zak Greant
  - Open Source advocate, MySQL community relations.
- Carsten Pedersen
  - The MySQL Certification program.
- Lenz Grimmer
  - Production (build and release) engineering.
- Peter Zaitsev
  - `SHA1()`, `AES_ENCRYPT()` and `AES_DECRYPT()` functions.
  - Debugging, cleaning up various features.
- Alexander (Salle) Keremidarski
  - Support.
  - Debugging.
- Per-Erik Martin
  - Lead developer for stored procedures (5.0) and triggers.
- Jim Winstead

  - Lead Web developer.
- Mark Matthews
  - Connector/J driver (Java).
- Peter Gulutzan
  - SQL standards compliance.
  - Documentation of existing MySQL code/algorithms.
  - Character set documentation.
- Guilhem Bichot
  - Replication, from `MySQL` version 4.0.
  - Fixed handling of exponents for `DECIMAL`.
  - Author of `mysql_tableinfo`.
- Antony T. Curtis
  - Porting of the MySQL Database software to OS/2.
- Mikael Ronstrom
  - Much of the initial work on NDB Cluster until 2000. Roughly half the code base at that time. Transaction protocol, node recovery, system restart and restart code and parts of the API functionality.
  - Lead Architect, developer, debugger of NDB Cluster 1994-2004
  - Lots of optimizations
- Jonas Oreland
  - On-line Backup
  - The automatic test environment of MySQL Cluster
  - Portability Library for NDB Cluster
  - Lots of other things
- Pekka Nouisiainen
  - Ordered index implementation of MySQL Cluster
  - BLOB support in MySQL Cluster
  - Charset support in MySQL Cluster
- Martin Skold
  - Unique index implementation of MySQL Cluster
  - Integration of NDB Cluster into MySQL

- Magnus Svensson
  - The test framework for MySQL Cluster
  - Integration of NDB Cluster into MySQL
- Tomas Ulin
  - Lots of work on configuration changes for simple installation and use of MySQL Cluster

B.2. Contributors to MySQL

While MySQL AB owns all copyrights in the `MySQL server` and the `MySQL manual`, we wish to recognize those who have made contributions of one kind or another to the `MySQL distribution`. Contributors are listed here, in somewhat random order:

- Gianmassimo Vigazzola `<qwerg@mbox.vol.it>` or `<qwerg@tin.it>`

  The initial port to Win32/NT.

- Per Eric Olsson

  For more or less constructive criticism and real testing of the dynamic record format.

- Irena Pancirov `<irena@mail.yacc.it>`

  Win32 port with Borland compiler. `mysqlshutdown.exe` and `mysqlwatch.exe`

- David J. Hughes

  For the effort to make a shareware SQL database. At TcX, the predecessor of MySQL AB, we started with `mSQL`, but found that it couldn't satisfy our purposes so instead we wrote an SQL interface to our application builder Unireg. **mysqladmin** and **mysql** client are programs that were largely influenced by their `mSQL` counterparts. We have put a lot of effort into making the MySQL syntax a superset of `mSQL`. Many of the API's ideas are borrowed from `mSQL` to make it easy to port free `mSQL` programs to the MySQL API. The MySQL software doesn't contain any code from `mSQL`. Two files in the distribution (`client/insert_test.c` and `client/select_test.c`) are based on the corresponding (non-copyrighted) files in the `mSQL` distribution, but are modified as examples showing the changes necessary to convert code from `mSQL` to MySQL Server. (`mSQL` is copyrighted David J. Hughes.)

- Patrick Lynch

  For helping us acquire http://www.mysql.com/.

- Fred Lindberg

  For setting up qmail to handle the MySQL mailing list and for the incredible help we got in managing the MySQL mailing lists.

- Igor Romanenko `<igor@frog.kiev.ua>`

  mysqldump (previously `msqldump`, but ported and enhanced by Monty).

- Yuri Dario

For keeping up and extending the MySQL OS/2 port.

- Tim Bunce

  Author of **mysqlhotcopy**.

- Zarko Mocnik `<zarko.mocnik@dem.si>`

  Sorting for Slovenian language.

- "TAMITO" `<tommy@valley.ne.jp>`

  The `_MB` character set macros and the ujis and sjis character sets.

- Joshua Chamas `<joshua@chamas.com>`

  Base for concurrent insert, extended date syntax, debugging on NT, and answering on the MySQL mailing list.

- Yves Carlier `<Yves.Carlier@rug.ac.be>`

  mysqlaccess, a program to show the access rights for a user.

- Rhys Jones `<rhys@wales.com>` (And GWE Technologies Limited)

  For one of the early JDBC drivers.

- Dr Xiaokun Kelvin ZHU `<X.Zhu@brad.ac.uk>`

  Further development of one of the early JDBC drivers and other MySQL-related Java tools.

- James Cooper `<pixel@organic.com>`

  For setting up a searchable mailing list archive at his site.

- Rick Mehalick `<Rick_Mehalick@i-o.com>`

  For `xmysql`, a graphical X client for MySQL Server.

- Doug Sisk `<sisk@wix.com>`

  For providing RPM packages of MySQL for Red Hat Linux.

- Diemand Alexander V. `<axeld@vial.ethz.ch>`

  For providing RPM packages of MySQL for Red Hat Linux-Alpha.

- Antoni Pamies Olive `<toni@readysoft.es>`

  For providing RPM versions of a lot of MySQL clients for Intel and SPARC.

- Jay Bloodworth `<jay@pathways.sde.state.sc.us>`

  For providing RPM versions for MySQL 3.21.

- David Sacerdote `<davids@secnet.com>`

  Ideas for secure checking of DNS hostnames.

- Wei-Jou Chen <jou@nematic.ieo.nctu.edu.tw>

  Some support for Chinese(BIG5) characters.

- Wei He <hewei@mail.ied.ac.cn>

  A lot of functionality for the Chinese(GBK) character set.

- Jan Pazdziora <adelton@fi.muni.cz>

  Czech sorting order.

- Zeev Suraski <bourbon@netvision.net.il>

  FROM_UNIXTIME() time formatting, ENCRYPT() functions, and **bison** advisor. Active mailing list member.

- Luuk de Boer <luuk@wxs.nl>

  Ported (and extended) the benchmark suite to DBI/DBD. Have been of great help with crash-me and running benchmarks. Some new date functions. The **mysql_setpermission** script.

- Alexis Mikhailov <root@medinf.chuvashia.su>

  User-defined functions (UDFs); CREATE FUNCTION and DROP FUNCTION.

- Andreas F. Bobak <bobak@relog.ch>

  The AGGREGATE extension to user-defined functions.

- Ross Wakelin <R.Wakelin@march.co.uk>

  Help to set up InstallShield for MySQL-Win32.

- Jethro Wright III <jetman@li.net>

  The libmysql.dll library.

- James Pereria <jpereira@iafrica.com>

  Mysqlmanager, a Win32 GUI tool for administering MySQL Servers.

- Curt Sampson <cjs@portal.ca>

  Porting of MIT-pthreads to NetBSD/Alpha and NetBSD 1.3/i386.

- Martin Ramsch <m.ramsch@computer.org>

  Examples in the MySQL Tutorial.

- Steve Harvey

  For making **mysqlaccess** more secure.

- Konark IA-64 Centre of Persistent Systems Private Limited

  http://www.pspl.co.in/konark/. Help with the Win64 port of the MySQL server.

- Albert Chin-A-Young.

Configure updates for Tru64, large file support and better TCP wrappers support.

- John Birrell

  Emulation of `pthread_mutex()` for OS/2.

- Benjamin Pflugmann

  Extended `MERGE` tables to handle `INSERTS`. Active member on the MySQL mailing lists.

- Jocelyn Fournier

  Excellent spotting and reporting innumerable bugs (especially in the MySQL 4.1 subquery code).

- Marc Liyanage

  Maintaining the Mac OS X packages and providing invaluable feedback on how to create Mac OS X PKGs.

- Robert Rutherford

  Providing invaluable information and feedback about the QNX port.

- Previous developers of NDB Cluster

  Lots of people were involved in various ways summer students, master thesis students, employees. In total more than 100 people so too many to mention here. Notable name is Ataullah Dabaghi who up until 1999 contributed around a third of the code base. A special thanks also to developers of the AXE system which provided much of the architecturial foundations for NDB Cluster with blocks, signals and crash tracing functionality. Also credit should be given to those who believed in the ideas enough to allocate of their budgets for its development from 1992 to present time.

Other contributors, bugfinders, and testers: James H. Thompson, Maurizio Menghini, Wojciech Tryc, Luca Berra, Zarko Mocnik, Wim Bonis, Elmar Haneke, `<jehamby@lightside>`, `<psmith@BayNetworks.com>`, `<duane@connect.com.au>`, Ted Deppner `<ted@psyber.com>`, Mike Simons, Jaakko Hyvatti.

And lots of bug report/patches from the folks on the mailing list.

A big tribute goes to those that help us answer questions on the MySQL mailing lists:

- Daniel Koch `<dkoch@amcity.com>`

  Irix setup.

- Luuk de Boer `<luuk@wxs.nl>`

  Benchmark questions.

- Tim Sailer `<tps@users.buoy.com>`

  `DBD::mysql` questions.

- Boyd Lynn Gerber `<gerberb@zenez.com>`

  SCO-related questions.

- Richard Mehalick <RM186061@shellus.com>

  xmysql-related questions and basic installation questions.

- Zeev Suraski <bourbon@netvision.net.il>

  Apache module configuration questions (log & auth), PHP-related questions, SQL syntax-related questions and other general questions.

- Francesc Guasch <frankie@citel.upc.es>

  General questions.

- Jonathan J Smith <jsmith@wtp.net>

  Questions pertaining to OS-specifics with Linux, SQL syntax, and other things that might need some work.

- David Sklar <sklar@student.net>

  Using MySQL from PHP and Perl.

- Alistair MacDonald <A.MacDonald@uel.ac.uk>

  Not yet specified, but is flexible and can handle Linux and maybe HP-UX. tries to get user to use **mysqlbug**.

- John Lyon <jlyon@imag.net>

  Questions about installing MySQL on Linux systems, using either .rpm files or compiling from source.

- Lorvid Ltd. <lorvid@WOLFENET.com>

  Simple billing/license/support/copyright issues.

- Patrick Sherrill <patrick@coconet.com>

  ODBC and VisualC++ interface questions.

- Randy Harmon <rjharmon@uptimecomputers.com>

  DBD, Linux, some SQL syntax questions.

B.3. Documenters and translators

The following people has helped us with writing the MySQL documentation and translating the documentation or error messages in MySQL.

- Paul DuBois

  Ongoing help with making this manual correct and understandable. That includes rewriting Monty's and David's attempts at English into English as other people know it.

- Kim Aldale

  Helped to rewrite Monty's and David's early attempts at English into English.

- Michael J. Miller Jr. <mke@terrapin.turbolift.com>

  For the first MySQL manual. And a lot of spelling/language fixes for the FAQ (that turned into the MySQL manual a long time ago).

- Yan Cailin

  First translator of the MySQL Reference Manual into simplified Chinese in early 2000 on which the Big5 and HK coded (http://mysql.hitstar.com/) versions were based. Personal home page at linuxdb.yeah.net [http://linuxdb.yeah.net].

- Jay Flaherty <fty@mediapulse.com>

  Big parts of the Perl DBI/DBD section in the manual.

- Paul Southworth <pauls@etext.org>, Ray Loyzaga <yar@cs.su.oz.au>

  Proof-reading of the Reference Manual.

- Therrien Gilbert <gilbert@ican.net>, Jean-Marc Pouyot <jmp@scalaire.fr>

  French error messages.

- Petr Snajdr, <snajdr@pvt.net>

  Czech error messages.

- Jaroslaw Lewandowski <jotel@itnet.com.pl>

  Polish error messages.

- Miguel Angel Fernandez Roiz

  Spanish error messages.

- Roy-Magne Mo <rmo@www.hivolda.no>

  Norwegian error messages and testing of MySQL 3.21.xx.

- Timur I. Bakeyev <root@timur.tatarstan.ru>

  Russian error messages.

- <brenno@dewinter.com> & Filippo Grassilli <phil@hyppo.com>

  Italian error messages.

- Dirk Munzinger <dirk@trinity.saar.de>

  German error messages.

- Billik Stefan <billik@sun.uniag.sk>

  Slovak error messages.

- Stefan Saroiu <tzoompy@cs.washington.edu>

  Romanian error messages.

- Peter Feher

Hungarian error messages.

- Roberto M. Serqueira

  Portuguese error messages.

- Carsten H. Pedersen

  Danish error messages.

- Arjen G. Lentz

  Dutch error messages, completing earlier partial translation (also work on consistency and spelling).

B.4. Libraries used by and included with MySQL

The following is a list of the creators of the libraries we have included with the MySQL server source to make it easy to compile and install MySQL. We are very thankfully to all individuals that have created these and it has made our life much easier.

- Fred Fish

  For his excellent C debugging and trace library. Monty has made a number of smaller improvements to the library (speed and additional options).

- Richard A. O'Keefe

  For his public domain string library.

- Henry Spencer

  For his regex library, used in `WHERE column REGEXP regexp`.

- Chris Provenzano

  Portable user level pthreads. From the copyright: This product includes software developed by Chris Provenzano, the University of California, Berkeley, and contributors. We are currently using version 1_60_beta6 patched by Monty (see `mit-pthreads/Changes-mysql`).

- Jean-loup Gailly and Mark Adler

  For the zlib library (used on MySQL on Windows).

- Bjorn Benson

  For his safe_malloc (memory checker) package which is used in when you configure MySQL with `--debug`.

- Free Software Foundation

  The `readline` library (used by the **mysql** command-line client).

- The NetBSD foundation

The `libedit` package (optionally used by the **mysql** command-line client).

B.5. Packages that support MySQL

The following is a list of creators/maintainers of some of the most important API/packages/applications that a lot of people use with MySQL.

We can't list every possible package here because the list would then be way to hard to maintain. For other packages, please refer to the software portal at http://solutions.mysql.com/software/.

- Tim Bunce, Alligator Descartes

  For the `DBD` (Perl) interface.

- Andreas Koenig `<a.koenig@mind.de>`

  For the Perl interface for MySQL Server.

- Jochen Wiedmann `<wiedmann@neckar-alb.de>`

  For maintaining the Perl `DBD::mysql` module.

- Eugene Chan `<eugene@acenet.com.sg>`

  For porting PHP for MySQL Server.

- Georg Richter

  MySQL 4.1 testing and bug hunting. New PHP 5.0 `mysqli` extension (API) for use with MySQL 4.1 and up.

- Giovanni Maruzzelli `<maruzz@matrice.it>`

  For porting iODBC (Unix ODBC).

- Xavier Leroy `<Xavier.Leroy@inria.fr>`

  The author of LinuxThreads (used by the MySQL Server on Linux).

B.6. Tools that were used to create MySQL

The following is a list of some of the tools we have used to create MySQL. We use this to express our thanks to those that has created them as without these we could not have made MySQL what it is today.

- Free Software Foundation

  From whom we got an excellent compiler (**gcc**), an excellent debugger (**gdb** and the `libc` library (from which we have borrowed `strto.c` to get some code working in Linux).

- Free Software Foundation & The XEmacs development team

  For a really great editor/environment used by almost everybody at MySQL AB.

- Julian Seward

  Author of `valgrind`, an excellent memory checker tool that has helped us find a lot of otherwise hard to find bugs in MySQL.

- Dorothea Lütkehaus and Andreas Zeller

  For `DDD` (The Data Display Debugger) which is an excellent graphical frontend to **gdb**).

B.7. Supporters of MySQL

While MySQL AB owns all copyrights in the `MySQL server` and the `MySQL manual`, we wish to recognize the following companies, which helped us finance the development of the `MySQL server`, such as by paying us for developing a new feature or giving us hardware for development of the `MySQL server`.

- VA Linux / Andover.net

  Funded replication.

- NuSphere

  Editing of the MySQL manual.

- Stork Design studio

  The MySQL Web site in use between 1998-2000.

- Intel

  Contributed to development on Windows and Linux platforms.

- Compaq

  Contributed to Development on Linux/Alpha.

- SWSoft

  Development on the embedded **mysqld** version.

- FutureQuest

  `--skip-show-database`

Appendix C. MySQL and the Future (the TODO)

This section summarizes the features that we plan to implement in MySQL Server. The items are ordered by release series. Within a list, items are shown in approximately the order they will be done.

Note: If you are an enterprise-level user with an urgent need for a particular feature, please contact `<sales@mysql.com>` to discuss sponsoring options. Targeted financing by sponsor companies allows us to allocate additional resources for specific purposes. One example of a feature sponsored in the past is replication.

C.1. New Features Planned for 5.0

The following features are planned for inclusion into MySQL 5.0. Some of the features such as stored procedures and true `VARCHAR` are complete and are included in MySQL 5.0 alpha. Others such as cursors are only partially available. Expect these and other features to mature and be fully supported in upcoming releases.

Note that because we have many developers that are working on different projects, there will also be many additional features. There is also a small chance that some of these features will be added to MySQL 4.1. For a list what is done in MySQL 4.1, see Section 1.3.2.1, “Features Available in MySQL 4.1”.

For those wishing to take a look at the bleeding edge of MySQL development, we make our BitKeeper repository for MySQL version 5.0 publicly available. See Section 2.8.3, “Installing from the Development Source Tree”.

- Views
  - Views, implemented in stepwise fashion up to full functionality. See Section 1.5.5.6, “Views”. See Section 13.2.7, “`CREATE VIEW` Syntax”.
- Stored Procedures
  - Stored procedures currently are implemented, based on the SQL:2003 standard. See Chapter 19, *Stored Procedures and Functions*.
- New functionality
  - Elementary cursor support. See Section 19.2.11, “Cursors”.
  - The ability to specify explicitly for `MyISAM` tables that an index should be created as an `RTREE` index. (In MySQL 4.1, `RTREE` indexes are used internally for geometrical data that use GIS data types, but cannot be created on request.)
  - Dynamic length rows for `MEMORY` tables.
  - `BIT` data type for holding bit-field values.
- Standards compliance, portability and migration
  - Support for Data Dictionary / `INFORMATION_SCHEMA`. See Chapter 21, *The `INFORMATION_SCHEMA` Information Database*.

  - Add true `VARCHAR` support (column lengths longer than 255, and no stripping of trailing whitespace).

- Speed enhancements
  - `SHOW COLUMNS FROM` *`tbl_name`* (used by the **mysql** client to allow expansions of column names) should not open the table, only the definition file. This requires less memory and will be much faster.
  - Allow `DELETE` on `MyISAM` tables to use the record cache. To do this, we need to update the threads record cache when we update the `.MYD` file.
  - Better support for `MEMORY` tables:
    - Dynamic length rows.
    - Faster row handling (less copying).

- Usability enhancements
  - Resolving the issue of `RENAME TABLE` on a table used in an active `MERGE` table possibly corrupting the table.

The news section of this manual includes a more in-depth list of features. See Section D.1, “Changes in release 5.0.x (Development)”.

C.2. New Features Planned for 5.1

- New functionality
  - `FOREIGN KEY` support for all table types, not just `InnoDB`.
  - Column-level constraints. See Section 1.5.6, “How MySQL Deals with Constraints”.
  - Online backup with very low performance penalty. The online backup will make it easy to add a new replication slave without taking down the master.

- Speed enhancements
  - New text based table definition file format (`.frm` files) and a table cache for table definitions. This enables us to do faster queries of table structures and do more efficient foreign key support.

- Usability enhancements
  - Add options to the client/server protocol to get progress notes for long running commands.
  - Implement `RENAME DATABASE`. To make this safe for all storage engines, it should work as follows:
    1. Create the new database.

2. For every table, do a rename of the table to another database, as we do with the `RENAME` command.
3. Drop the old database.

- New internal file interface change. This makes all file handling much more general and makes it easier to add extensions.

C.3. New Features Planned for the Near Future

- New functionality
  - Oracle-like `CONNECT BY PRIOR` to search tree-like (hierarchical) structures.
  - Add all missing standard SQL and ODBC 3.0 types.
  - `INSERT SQL_CONCURRENT` and **mysqld --concurrent-insert** to do a concurrent insert at the end of a table if the table is read-locked.
  - Allow variables to be updated in `UPDATE` statements. For example: `UPDATE foo SET @a:=a+b,a=@a, b=@a+c`.
  - Change when user variables are updated so that you can use them with `GROUP BY`, as in the following statement: `SELECT id, @a:=COUNT(*), SUM(`*`sum_col`*`)/@a FROM `*`tbl_name`*` GROUP BY id`.
  - Add an `IMAGE` option to `LOAD DATA INFILE` to not update `TIMESTAMP` and `AUTO_INCREMENT` columns.
  - Add `LOAD DATA INFILE ... UPDATE` syntax that works like this:
    - For tables with primary keys, if an input record contains a primary key value, existing rows matching that primary key value are updated from the remainder of the input columns. However, columns corresponding to columns that are *missing* from the input record are not touched.
    - For tables with primary keys, if an input record does not contain the primary key value or is missing some part of the key, the record is treated as `LOAD DATA INFILE ... REPLACE INTO`.
  - Make `LOAD DATA INFILE` understand syntax like this:

    ```
    LOAD DATA INFILE 'file_name.txt' INTO TABLE tbl_name
         TEXT_FIELDS (text_col1, text_col2, text_col3)
         SET table_col1=CONCAT(text_col1, text_col2),
             table_col3=23
         IGNORE text_col3
    ```

    This can be used to skip over extra columns in the text file, or update columns based on expressions of the read data.
  - New functions for working with `SET` type columns:
    - `ADD_TO_SET(`*`value`*`,`*`set`*`)`

    - `REMOVE_FROM_SET(value,set)`

  - If you abort **mysql** in the middle of a query, you should open another connection and kill the old running query. Alternatively, an attempt should be made to detect this in the server.

  - Add a storage engine interface for table information so that you can use it as a system table. This would be a bit slow if you requested information about all tables, but very flexible. `SHOW INFO FROM tbl_name` for basic table information should be implemented.

  - Allow `SELECT a FROM tbl_name1 LEFT JOIN tbl_name2 USING (a)`; in this case `a` is assumed to come from `tbl_name1`.

  - `DELETE` and `REPLACE` options to the `UPDATE` statement (this deletes rows when a duplicate-key error occurs while updating).

  - Change the format of `DATETIME` to store fractions of seconds.

  - Make it possible to use the new GNU `regexp` library instead of the current one (the new library should be much faster than the current one).

- Standards compliance, portability and migration

  - Add `ANY()`, `EVERY()`, and `SOME()` group functions. In standard SQL, these work only on boolean columns, but we can extend these to work on any columns or expressions by treating a value of zero as FALSE and non-zero values as TRUE.

  - Fix the type of `MAX(column)` to be the same as the column type:

    ```
    mysql> CREATE TABLE t1 (a DATE);
    mysql> INSERT INTO t1 VALUES (NOW());
    mysql> CREATE TABLE t2 SELECT MAX(a) FROM t1;
    mysql> SHOW COLUMNS FROM t2;
    ```

- Speed enhancements

  - Don't allow more than a defined number of threads to run `MyISAM` recovery at the same time.

  - Change `INSERT INTO ... SELECT` to optionally use concurrent inserts.

  - Add an option to periodically flush key pages for tables with delayed keys if they haven't been used in a while.

  - Allow join on key parts (optimization issue).

  - Add a log file analyzer that can extract information about which tables are hit most often, how often multiple-table joins are executed, and so on. This should help users identify areas of table design that could be optimized to execute much more efficient queries.

- Usability enhancements

  - Return the original column types when doing `SELECT MIN(column) ... GROUP BY`.

  - Make it possible to specify `long_query_time` with a granularity in microseconds.
  - Link the **myisampack** code into the server so that it can perform `PACK` or `COMPRESS` operations.
  - Add a temporary key buffer cache during `INSERT/DELETE/UPDATE` so that we can gracefully recover if the index file gets full.
  - If you perform an `ALTER TABLE` on a table that is symlinked to another disk, create temporary tables on that disk.
  - Implement a `DATE/DATETIME` type that handles time zone information properly, to make dealing with dates in different time zones easier.
  - Fix **configure** so that all libraries (like `MyISAM`) can be compiled without threads.
  - Allow user variables as `LIMIT` arguments; for example, `LIMIT @a,@b`.
  - Automatic output from **mysql** to a Web browser.
  - `LOCK DATABASES` (with various options).
  - Many more variables for `SHOW STATUS`. Record reads and updates. Selects on a single table and selects with joins. Mean number of tables in selects. Number of `ORDER BY` and `GROUP BY` queries.
  - **mysqladmin copy database new-database**; this requires a `COPY` operation to be added to **mysqld**.
  - Processlist output should indicate the number of queries/threads.
  - `SHOW HOSTS` for printing information about the hostname cache.
  - Change table names from empty strings to `NULL` for calculated columns.
  - Don't use `Item_copy_string` on numerical values to avoid number-to-string-to-number conversion in case of `SELECT COUNT(*)*(id+0) FROM` *`tbl_name`* `GROUP BY id`.
  - Change so that `ALTER TABLE` doesn't abort clients that execute `INSERT DELAYED`.
  - Fix so that when columns are referenced in an `UPDATE` clause, they contain the old values from before the update started.
- New operating systems
  - Port the MySQL clients to LynxOS.

C.4. New Features Planned for the Mid-Term Future

- Implement function: `get_changed_tables(timeout,table1,table2,...)`.
- Change reading through tables to use `mmap()` when possible. Only compressed tables use `mmap()`.

- Make the automatic timestamp code nicer. Add timestamps to the update log with `SET TIMESTAMP=val;`.
- Use read/write mutex in some places to get more speed.
- Automatically close some tables if a table, temporary table, or temporary file gets error 23 (too many open files).
- Better constant propagation. When an occurrence of *`col_name=n`* is found in an expression, for some constant `n`, replace other occurrences of *`col_name`* within the expression with `n`. Currently, this is done only for some simple cases.
- Change all const expressions with calculated expressions if possible.
- Optimize *`key`* = *`expr`* comparisons. At the moment, only *`key`* = *`column`* or *`key`* = *`constant`* comparisons are optimized.
- Join some of the copy functions for nicer code.
- Change `sql_yacc.yy` to an inline parser to reduce its size and get better error messages.
- Change the parser to use only one rule per different number of arguments in function.
- Use of full calculation names in the order part (for Access97).
- `MINUS`, `INTERSECT`, and `FULL OUTER JOIN`. (Currently `UNION` and `LEFT|RIGHT OUTER JOIN` are supported.)
- Allow `SQL_OPTION MAX_SELECT_TIME=val`, for placing a time limit on a query.
- Allow updates to be logged to a database.
- Enhance `LIMIT` to allow retrieval of data from the end of a result set.
- Alarm around client connect/read/write functions.
- Please note the changes to **mysqld_safe**: According to FSSTND (which Debian tries to follow), PID files should go into `/var/run/<progname>.pid` and log files into `/var/log`. It would be nice if you could put the "DATADIR" in the first declaration of "pidfile" and "log" so that the placement of these files can be changed with a single statement.
- Allow a client to request logging.
- Allow the `LOAD DATA INFILE` statement to read files that have been compressed with **gzip**.
- Fix sorting and grouping of `BLOB` columns (partly solved).
- Change to use semaphores when counting threads. One should first implement a semaphore library for MIT-pthreads.
- Add full support for `JOIN` with parentheses.
- As an alternative to the one-thread-per-connection model, manage a pool of threads to handle queries.
- Allow `GET_LOCK()` to obtain more than one lock. When doing this, it is also necessary to handle the possible deadlocks this change will introduce.

C.5. New Features We Don't Plan to Implement

We aim toward full compliance with ANSI/ISO SQL. There are no features we plan not to implement.

Appendix D. MySQL Change History

This appendix lists the changes from version to version in the MySQL source code.

We are working actively on MySQL 4.1 and 5.0, and provide only critical bugfixes for MySQL 4.0 and MySQL 3.23. We update this section as we add new features, so that everybody can follow the development.

Our TODO section contains what further plans we have for MySQL 4.1 and 5.0. See Appendix C, *MySQL and the Future (the TODO)*.

Note that we tend to update the manual at the same time we make changes to MySQL. If you find a recent version of MySQL listed here that you can't find on our download page (http://dev.mysql.com/downloads/), it means that the version has not yet been released.

The date mentioned with a release version is the date of the last BitKeeper ChangeSet on which the release was based, not the date when the packages were made available. The binaries are usually made available a few days after the date of the tagged ChangeSet, because building and testing all packages takes some time.

D.1. Changes in release 5.0.x (Development)

The following changelog shows what has been done in the 5.0 tree:

- Basic support for read-only server side cursors.
- Basic support for (updatable) views. See, for example, Section 13.2.7, "`CREATE VIEW` Syntax".
- Basic support for stored procedures (SQL:2003 style). See Chapter 19, *Stored Procedures and Functions*.
- Initial support for rudimentary triggers.
- Added `SELECT INTO` *`list_of_vars`*, which can be of mixed (that is, global and local) types. See Section 19.2.9.3, "`SELECT ... INTO` Statement".
- Removed the update log. It is fully replaced by the binary log. If the MySQL server is started with `--log-update`, it is translated to `--log-bin` (or ignored if the server is explicitly started with `--log-bin`), and a warning message is written to the error log. Setting `SQL_LOG_UPDATE` silently sets `SQL_LOG_BIN` instead (or do nothing if the server is explicitly started with `--log-bin`).
- Support for the `ISAM` storage engine has been removed. If you have `ISAM` tables, you should convert them before upgrading. See Section 2.10.1, "Upgrading from Version 4.1 to 5.0".
- Support for `RAID` options in `MyISAM` tables has been removed. If you have tables that use these options, you should convert them before upgrading. See Section 2.10.1, "Upgrading from Version 4.1 to 5.0".
- User variable names are now case insensitive: If you do `SET @a=10;` then `SELECT @A;` now returns `10`. Case sensitivity of a variable's value depends on the collation of the value.
- Strict mode, which in essence means that you get an error instead of a warning when inserting an incorrect value into a column. See Section 5.2.2, "The Server SQL Mode".
- `VARCHAR` and `VARBINARY` columns remember end space. A `VARCHAR()` or `VARBINARY`

column can contain up to 65,535 characters or bytes, respectively.

- MEMORY (HEAP) tables can have VARCHAR() columns.
- When using a constant string or a function that generate a string result in CREATE ... SELECT, MySQL creates the result field based on the max_length of the string/expression:

max_length	Column type
= 0	CHAR(0)
< 512	VARCHAR(max_length)
>= 512	TEXT

For a full list of changes, please refer to the changelog sections for each individual 5.0.x release.

D.1.1. Changes in release 5.0.4 (not released yet)

Functionality added or changed:

-

Bugs fixed:

- Fixed server crash resulting from queries that combined SELECT DISTINCT, SUM(), and ROLLUP. (Bug #8615 [http://bugs.mysql.com/8615])
- Incorrect results were returned from queries that combined SELECT DISTINCT, GROUP BY , and ROLLUP. (Bug #8616 [http://bugs.mysql.com/8616])
- Too many rows were returned from queries that combined ROLLUP and LIMIT if SQL_CALC_FOUND_ROWS was given. (Bug #8617 [http://bugs.mysql.com/8617])

D.1.2. Changes in release 5.0.3 (release imminent)

Functionality added or changed:

- Security improvement: The server creates .frm, .MYD, .MYI, .MRG, .ISD, and .ISM table files only if a file with the same name does not already exist. (Thanks to Stefano Di Paola <stefano.dipaola@wisec.it> for finding and informing us about this issue.)
- Security improvement: User-defined functions should have at least one symbol defined in addition to the xxx symbol that corresponds to the main xxx() function. These auxiliary symbols correspond to the xxx_init(), xxx_deinit(), xxx_reset(), xxx_clear(), and xxx_add() functions. **mysqld** by default no longer loads UDFs unless they have at least one auxiliary symbol defined in addition to the main symbol. The --allow-suspicious-udfs option controls whether UDFs that have only an xxx symbol can be loaded. By default, the option is off. mysqld also checks UDF filenames when it reads them from the mysql.func table and rejects those that contain directory pathname separator characters. (It already checked names as given in CREATE FUNCTION statements.) See Section 25.2.3.1, “UDF Calling Sequences for simple functions”, Section 25.2.3.2, “UDF Calling Sequences for aggregate functions”, and Section 25.2.3.6, “User-defined Function Security Precautions”. (Thanks to Stefano Di Paola <stefano.dipaola@wisec.it>

for finding and informing us about this issue.)

- Support for the ISAM storage engine has been removed. If you have ISAM tables, you should convert them before upgrading. See Section 2.10.1, “Upgrading from Version 4.1 to 5.0”.
- Support for RAID options in MyISAM tables has been removed. If you have tables that use these options, you should convert them before upgrading. See Section 2.10.1, “Upgrading from Version 4.1 to 5.0”.
- Added support for AVG(DISTINCT).
- ONLY_FULL_GROUP_BY no longer is included in the ANSI composite SQL mode. (Bug #8510 [http://bugs.mysql.com/8510])
- **mysqld_safe** will create the directory where the UNIX socket file is to be located if the directory does not exist. This applies only to the last component of the directory pathname. (Bug #8513 [http://bugs.mysql.com/8513])
- The coercibility for the return value of functions such as USER() or VERSION() now is ``system constant'' rather than ``implicit.'' This makes these functions more coercible than column values so that comparisons of the two do not result in Illegal mix of collations errors. COERCIBILITY() was modified to accommodate this new coercibility value. See Section 12.8.3, “Information Functions”.
- User variable coercibility has been changed from ``coercible'' to ``implicit.'' That is, user variables have the same coercibility as column values.
- Boolean full-text phrase searching now requires only that matches contain exactly the same words as the phrase and in the same order. Non-word characters no longer need match exactly.
- CHECKSUM TABLE returns a warning for non-existing tables. The checksum value remains NULL as before. (Bug #8256 [http://bugs.mysql.com/8256])
- The server now includes a timestamp in the Ready for connections message that is written to the error log at startup. (Bug #8444 [http://bugs.mysql.com/8444])
- Added SQL_NOTES session variable to cause Note-level warnings not to be recorded. (Bug #6662 [http://bugs.mysql.com/6662])
- Allowed the service-installation command for Windows servers to specify a single option other than --defaults-file following the service name. This is for compatibility with MySQL 4.1. (Bug #7856 [http://bugs.mysql.com/7856])
- InnoDB: Commit after every 10,000 copied rows when executing ALTER TABLE, CREATE INDEX, DROP INDEX or OPTIMIZE TABLE. This makes it much faster to recover from an aborted operation.
- Added VAR_POP() and STDDEV_POP() as standard SQL aliases for the VARIANCE() and STDDEV() functions that compute population variance and standard deviation. Added new VAR_SAMP() and STDDEV_SAMP() functions to compute sample variance and standard deviation. (Bug #3190 [http://bugs.mysql.com/3190])
- Fixed a problem with out-of-order packets being sent (ERROR after OK or EOF) following a KILL QUERY statement. (Bug #6804 [http://bugs.mysql.com/6804])
- Retrieving from a view defined as a SELECT that mixed UNION ALL and UNION DISTINCT resulted in a different result than retrieving from the original SELECT. (Bug #6565 [http://bugs.mysql.com/6565])

- Fixed a problem with non-optimal `index_merge` query execution plans being chosen on IRIX. (Bug #8578 [http://bugs.mysql.com/8578])
- `BIT` in column definitions now is a distinct data type; it no longer is treated as a synonym for `TINYINT(1)`.
- Bit-field values can be written using `b'value'` notation. *`value`* is a binary value written using 0s and 1s.
- From the Windows distribution, predefined accounts without passwords for remote users ("root@%", "@%") were removed (other distributions never had them).
- Added `mysql_library_init()` and `mysql_library_end()` as synonyms for the `mysql_server_init()` and `mysql_server_end()` C API functions. `mysql_library_init()` and `mysql_library_end()` are `#define` symbols, but the names more clearly indicate that they should be called when beginning and ending use of a MySQL C API library no matter whether the application uses `libmysqlclient` or `libmysqld`. (Bug #6149 [http://bugs.mysql.com/6149])
- `SHOW COLUMNS` now displays `NO` rather than blank in the `Null` output column if the corresponding table column cannot be `NULL`.
- Changed XML format for **mysql** from `<col_name>col_value</col_name>` to `<field name="col_name">col_value</field>` to allow for proper encoding of column names that are not legal as element names. (Bug #7811 [http://bugs.mysql.com/7811])
- Added `--innodb-checksums` and `--innodb-doublewrite` options for **mysqld**.
- Added `--large-pages` option for **mysqld**.
- Added `multi_read_range` system variable.
- `SHOW DATABASES`, `SHOW TABLES`, `SHOW COLUMNS`, and so forth display information about the `INFORMATION_SCHEMA` database. Also, several `SHOW` statements now accept a `WHERE` clause specifying which output rows to display. See Chapter 21, *The `INFORMATION_SCHEMA` Information Database*.
- Added the `CREATE ROUTINE` and `ALTER ROUTINE` privileges, and made the `EXECUTE` privilege operational.
- InnoDB: Corrected a bug in the crash recovery of `ROW_FORMAT=COMPACT` tables that caused corruption. (Bug #7973 [http://bugs.mysql.com/7973]) There may still be bugs in the crash recovery, especially in `COMPACT` tables.
- When the `MyISAM` storage engine detects corruption of a `MyISAM` table, a message describing the problem now is written to the error log.
- InnoDB: When MySQL/InnoDB is compiled on Mac OS X 10.2 or earlier, detect the operating system version at run time and use the `fcntl()` file flush method on Mac OS X versions 10.3 and later. Apple had disabled `fsync()` in Mac OS X for internal disk drives, which caused corruption at power outages.
- InnoDB: Implemented fast `TRUNCATE TABLE`. The old approach (deleting rows one by one) may be used if the table is being referenced by foreign keys. (Bug #7150 [http://bugs.mysql.com/7150])
- Added `cp932` (SJIS for Windows Japanese) and `eucjpms` (UJIS for Windows Japanese) character sets.
- Added several `InnoDB` status variables. See Section 5.2.4, “Server Status Variables”.

- Added the FEDERATED storage engine. See Section 14.6, “The FEDERATED Storage Engine”.
- SHOW CREATE TABLE now uses USING *index_type* rather than TYPE *index_type* to specify an index type. (Bug #7233 [http://bugs.mysql.com/7233])
- InnoDB now supports a fast TRUNCATE TABLE. One visible change from this is that auto-increment values for this table are reset on TRUNCATE.
- Added an error member to the MYSQL_BIND data structure that is used in the C API for prepared statements. This member is used for reporting data truncation errors. Truncation reporting is enabled via the new MYSQL_REPORT_DATA_TRUNCATION option for the mysql_options() C API function.
- API change: the reconnect flag in the MYSQL structure is now set to 0 by mysql_real_connect(). Only those client programs which didn't explicitly set this flag to 0 or 1 after mysql_real_connect() experience a change. Having automatic reconnection enabled by default was considered too dangerous (after reconnection, table locks, temporary tables, user and session variables are lost).
- FLUSH TABLES WITH READ LOCK is now killable while it's waiting for running COMMIT statements to finish.
- MEMORY (HEAP) can have VARCHAR() fields.
- VARCHAR columns now remember end space. A VARCHAR() column can now contain up to 65535 bytes. For more details, see Section D.1, “Changes in release 5.0.x (Development)”. If the table handler doesn't support the new VARCHAR type, then it's converted to a CHAR column. Currently this happens for NDB and InnoDB tables.
- InnoDB: Introduced a compact record format that does not store the number of columns or the lengths of fixed-size columns. The old format can be requested by specifying ROW_FORMAT=REDUNDANT. The new format (ROW_FORMAT=COMPACT) is the default.
- InnoDB: Setting the initial AUTO_INCREMENT value for the table using CREATE TABLE...AUTO_INCREMENT = works now also for the InnoDB table.
- Seconds_Behind_Master is NULL (which means ``unknown'') if the slave SQL thread is not running, or if the slave I/O thread is not running or not connected to master. It is zero if the SQL thread has caught up to the I/O thread. It no longer grows indefinitely if the master is idle.
- The MySQL server aborts immediately instead of simply issuing a warning if it is started with the --log-bin option but cannot initialize the binary log at startup (that is, an error occurs when writing to the binary log file or binary log index file).
- The binary log file and binary log index file now are handled the same way as MyISAM tables when there is a ``disk full'' or ``quota exceeded'' error. See Section A.4.3, “How MySQL Handles a Full Disk”.
- The MySQL server now aborts when started with option --log-bin-index and without --log-bin, and when started with --log-slave-updates and without --log-bin.
- If the MySQL server is started without an argument to --log-bin and without --log-bin-index, thus not providing a name for the binary log index file, a warning is issued because MySQL falls back to using the hostname for that name, and this is prone to replication issues if the server's hostname's gets changed later. See Section 1.5.7.3, “Open Bugs and Design Deficiencies in MySQL”.
- Added account-specific MAX_USER_CONNECTIONS limit, which allows you to specify the maximum number of concurrent connections for the account. Also, all limited resources now are counted

per account (instead of being counted per user + host pair as it was before). Use the `--old-style-user-limits` option to get the old behavior.

- InnoDB: A shared record lock (`LOCK_REC_NOT_GAP`) is now taken for a matching record in the foreign key check because inserts can be allowed into gaps.

- InnoDB: Relaxed locking in INSERT...SELECT, single table UPDATE...SELECT and single table DELETE...SELECT clauses when innobase_locks_unsafe_for_binlog is used and isolation level of the transaction is not serializable. `InnoDB` uses consistent read in these cases for a selected table.

- Added a new global system variable `slave_transaction_retries`: If the replication slave SQL thread fails to execute a transaction because of an `InnoDB` deadlock or exceeded `innodb_lock_wait_timeout`, it automatically retries `slave_transaction_retries` times before stopping with an error. The default is 10. (Bug #8325 [http://bugs.mysql.com/8325])

- When a client releases a user-level lock, `DO RELEASE_LOCK()` will not be written to the binary log anymore (this makes the binary log smaller); as a counterpart, the slave does not actually take the lock when it executes `GET_LOCK()`. This is mainly an optimization and should not affect existing setups. (Bug #7998 [http://bugs.mysql.com/7998])

- The `LOAD DATA` statement was extended to support user variables in the target column list, and an optional `SET` clause. Now one can perform some transformations on data after they have been read and before they are inserted into the table. For example:

  ```
  LOAD DATA INFILE 'file.txt'
    INTO TABLE t1
    (column1, @var1)
    SET column2 = @var1/100;
  ```

  Also, replication of `LOAD DATA` was changed, so you can't replicate such statements from a 5.0.3 master to pre-5.0.3 slaves.

Bugs fixed:

- Fixed bug creating user with GRANT fails with password but works without, (Bug #7905 [http://bugs.mysql.com/7905])

- **mysqldump** misinterpreted '_' and '%' characters in the names of tables to be dumped as wildcard characters. (Bug #9123 [http://bugs.mysql.com/9123])

- The definition of the enumeration-valued `sql_mode` column of the `mysql.proc` table was missing some of the current allowable SQL modes, so stored routines would not necessarily execute with the SQL mode in effect at the time of routine definition. (Bug #8902 [http://bugs.mysql.com/8902])

- `REPAIR TABLE` did not invalidate query results in the query cache that were generated from the table. (Bug #8480 [http://bugs.mysql.com/8480])

- In strict or traditional SQL mode, too-long string values assigned to string columns (`CHAR`, `VARCHAR`, `BINARY`, `VARBINARY`, `TEXT`, or `BLOB`) were correctly truncated, but the server returned an SQLSTATE value of `01000` (should be `22001`). (Bug #6999 [http://bugs.mysql.com/6999], Bug #9029 [http://bugs.mysql.com/9029])

- Stored functions that used cursors could return incorrect results. (Bug #8386 [http://bugs.mysql.com/8386])

- `AES_DECRYPT(`*`col_name`*`,`*`key`*`)` could fail to return `NULL` for invalid values in *`col_name`*, if *`col_name`* was declared as `NOT NULL`. (Bug #8669 [http://bugs.mysql.com/8669])

- Ordering by unsigned expression (more complex than a column reference) was treating the value as signed, producing incorrectly sorted results. (Bug #7425 [http://bugs.mysql.com/7425])
- HAVING was treating unsigned columns as signed. (Bug #7425 [http://bugs.mysql.com/7425])
- Fixed a problem with boolean full-text searches on utf8 columns where a double quote in the search string caused a server crash. (Bug #8351 [http://bugs.mysql.com/8351])
- For a query with both GROUP BY and COUNT(DISTINCT) clauses and a FROM clause with a subquery, NULL was returned for any VARCHAR column selected by the subquery. (Bug #8218 [http://bugs.mysql.com/8218])
- Fixed a bug in TRUNCATE, which did not work within stored procedures. A workaround has been made so that within stored procedures, TRUNCATE is executed like DELETE. This was necessary because TRUNCATE is implicitly locking tables. (Bug #8850 [http://bugs.mysql.com/8850])
- Fixed an optimizer bug that caused incorrectly ordered result from a query that used a FULLTEXT index to retrieve rows and there was another index that was usable for ORDER BY. For such a query, EXPLAIN showed fulltext join type, but regular (not FULLTEXT) index in the Key column. (Bug #6635 [http://bugs.mysql.com/6635])
- If SELECT DISTINCT named an index column multiple times in the select list, the server tried to access different key fields for each instance of the column, which could result in a crash. (Bug #8532 [http://bugs.mysql.com/8532])
- For a stored function that refers to a given table, invoking the function while selecting from the same table resulted in a server crash. (Bug #8405 [http://bugs.mysql.com/8405])
- Comparison of a DECIMAL column containing NULL to a subquery that produced DECIMAL values resulted in a server crash. (Bug #8397 [http://bugs.mysql.com/8397])
- The --set-character-set option for **myisamchk** was changed to --set-collation. The value needed for specifying how to sort indexes is a collation name, not a character set name. (Bug #8349 [http://bugs.mysql.com/8349])
- Hostname matching didn't work if a netmask was specified for table-specific privileges. (Bug #3309 [http://bugs.mysql.com/3309])
- Corruption of MyISAM table indexes could occur with TRUNCATE TABLE if the table had already been opened. For example, this was possible if the table had been opened implicitly by selecting from a MERGE table that mapped to the MyISAM table. The server now issues an error message for TRUNCATE TABLE under these conditions. (Bug #8306 [http://bugs.mysql.com/8306])
- Setting the connection collation to a value different from the server collation followed by a CREATE TABLE statement that included a quoted default value resulted in a server crash. (Bug #8235 [http://bugs.mysql.com/8235])
- Fixed handling of table-name matching in **mysqlhotcopy** to accommodate DBD::mysql 2.9003 and up (which implement identifier quoting). (Bug #8136 [http://bugs.mysql.com/8136])
- Selecting from a view defined as a join caused a server crash if the query cache was enabled. (Bug #8054 [http://bugs.mysql.com/8054])
- Results in the query cache generated from a view were not properly invalidated after ALTER VIEW or DROP VIEW on that view. (Bug #8050 [http://bugs.mysql.com/8050])
- FOUND_ROWS() returned an incorrect value after a SELECT SQL_CALC_FOUND_ROWS DISTINCT statement that selected constants and included GROUP BY and LIMIT clauses. (Bug #7945 [http://bugs.mysql.com/7945])

- Selecting from an INFORMATION_SCHEMA table combined with a subselect on an INFORMATION_SCHEMA table caused an error with the message Table *tbl_name* is corrupted. (Bug #8164 [http://bugs.mysql.com/8164])
- Fixed a problem with equality propagation optimization for prepared statements and stored procedures that caused a server crash upon re-execution of the prepared statement or stored procedure. (Bug #8115 [http://bugs.mysql.com/8115], Bug #8849 [http://bugs.mysql.com/8849])
- LEFT OUTER JOIN between an empty base table and a view on an empty base table caused a server crash. (Bug #7433 [http://bugs.mysql.com/7433])
- Use of GROUP_CONCAT() in the select list when selecting from a view caused a server crash. (Bug #7116 [http://bugs.mysql.com/7116])
- Use of a view in a correlated subquery that contains HAVING but no GROUP BY caused a server crash. (Bug #6894 [http://bugs.mysql.com/6894])
- Handling by mysql_list_fields() of references to stored functions within views was incorrect and could result in a server crash. (Bug #6814 [http://bugs.mysql.com/6814])
- **mysqldump** now avoids writing SET NAMES to the dump output if the server is older than version 4.1 and would not understand that statement. (Bug #7997 [http://bugs.mysql.com/7997])
- Fixed problems when selecting from a view that had an EXISTS or NOT EXISTS subquery. Selecting columns by name caused a server crash. With SELECT *, a crash did not occur, but columns in outer query were not resolved properly. (Bug #6394 [http://bugs.mysql.com/6394])
- DDL statements for views were not being written to the binary log (and thus not subject to replication). (Bug #4838 [http://bugs.mysql.com/4838])
- The CHAR() function was not ignoring NULL arguments, contrary to the documentation. (Bug #6317 [http://bugs.mysql.com/6317])
- Creating a table using a name containing a character that is illegal in character_set_client resulted in the character being stripped from the name and no error. The character now is considered an error. (Bug #8041 [http://bugs.mysql.com/8041])
- Fixed a problem with the Cyrillic letters I and SHORT I being treated the same by the utf8_general_ci collation. (Bug #8385 [http://bugs.mysql.com/8385])
- Some INFORMATION_SCHEMA columns that contained catalog identifiers were of type LONGTEXT. These were changed to VARCHAR(*N*, where *N* is the appropriate maximum identifier length. (Bug #7215 [http://bugs.mysql.com/7215])
- Some INFORMATION_SCHEMA columns that contained timestamp values were of type VARBINARY. These were changed to TIMESTAMP. (Bug #7217 [http://bugs.mysql.com/7217])
- An expression that tested a case-insensitive character column against string constants that differed in lettercase could fail because the constants were treated as having a binary collation. (For example, WHERE city='London' AND city='london' could fail.) (Bug #7098 [http://bugs.mysql.com/7098], Bug #8690 [http://bugs.mysql.com/8690])
- The output of the STATUS (\s) command in **mysql** had the values for the server and client character sets reversed. (Bug #7571 [http://bugs.mysql.com/7571])
- If the slave was running with --replicate-*-table options which excluded one temporary table and included another, and the two tables were used in a single DROP TEMPORARY TABLE IF EXISTS statement, as the ones the master automatically writes to its binary log upon client's disconnection when client has not explicitly dropped these, the slave could forget to delete the included

replicated temporary table. Only the slave needs to be upgraded. (Bug #8055 [http://bugs.mysql.com/8055])

- When setting integer system variables to a negative value with `SET VARIABLES`, the value was treated as a positive value modulo 2^32. (Bug #6958 [http://bugs.mysql.com/6958])

- Corrected a problem with references to `DUAL` where statements such as `SELECT 1 AS a FROM DUAL` would succeed but statements such as `SELECT 1 AS a FROM DUAL LIMIT 1` would fail. (Bug #8023 [http://bugs.mysql.com/8023])

- Fixed a server crash caused by `DELETE FROM tbl_name ... WHERE ... ORDER BY tbl_name.col_name` when the `ORDER BY` column was qualified with the table name. (Bug #8392 [http://bugs.mysql.com/8392])

- Fixed a bug in `MATCH ... AGAINST` in natural language mode that could cause a server crash if the `FULLTEXT` index was not used in a join (`EXPLAIN` did not show `fulltext` join mode) and the search query matched no rows in the table (Bug #8522 [http://bugs.mysql.com/8522]).

- `InnoDB`: Honor the `--tmpdir` startup option when creating temporary files. Previously, `InnoDB` temporary files were always created in the temporary directory of the operating system. On Netware, `InnoDB` will continue to ignore `--tmpdir`. (Bug #5822 [http://bugs.mysql.com/5822])

- Platform and architecture information in version information produced for `--version` option on Windows was always `Win95/Win98 (i32)`. More accurately determine platform as `Win32` or `Win64` for 32-bit or 64-bit Windows, and architecture as `ia32` for x86, `ia64` for Itanium, and `axp` for Alpha. (Bug #4445 [http://bugs.mysql.com/4445])

- If multiple semicolon-separated statements were received in a single packet, they were written to the binary log as a single event rather than as separate per-statement events. For a server serving as a replication master, this caused replication to fail when the event was sent to slave servers. (Bug #8436 [http://bugs.mysql.com/8436])

- Fixed `LOAD INDEX` statement to actually load index in memory. (Bug #8452 [http://bugs.mysql.com/8452])

- Fixed a failure of multiple-table updates to replicate properly on slave servers when `--replicate-*-table` options had been specified. (Bug #7011 [http://bugs.mysql.com/7011])

- Fixed failure of `CREATE TABLE ... LIKE` Windows when the source or destination table was located in a symlinked database directory. (Bug #6607 [http://bugs.mysql.com/6607])

- With `lower_case_table_names` set to 1, **mysqldump** on Windows could write the same table name in different lettercase for different SQL statements. Fixed so that consistent lettercase is used. (Bug #5185 [http://bugs.mysql.com/5185])

- **mysqld_safe** now understands the `--help` option. Previously, it ignored the option and attempted to start the server anyway. (Bug #7931 [http://bugs.mysql.com/7931])

- Fixed problem in `NO_BACKSLASH_ESCAPES` SQL mode for strings that contained both the string quoting character and backslash. (Bug #6368 [http://bugs.mysql.com/6368])

- Fixed some portability issues with overflow in floating point values.

- Prepared statements now gives warnings on prepare.

- Fixed bug in prepared statements with `SUM(DISTINCT...)`.

- Fixed bug in prepared statements with `OUTER JOIN`.

- Fixed a bug in `CONV()` function returning unsigned `BIGINT` number (third argument is positive, and return value does not fit in 32 bits). (Bug #7751 [http://bugs.mysql.com/7751])
- Fixed a failure of the `IN()` operator to return correct result if all values in the list were constants and some of them were using substring functions, for example, `LEFT()`, `RIGHT()`, or `MID()`. (Bug #7716 [http://bugs.mysql.com/7716])
- Fixed a crash in `CONVERT_TZ()` function when its second or third argument was from a `const` table (see Section 7.2.1, "`EXPLAIN` Syntax (Get Information About a `SELECT`)"). (Bug #7705 [http://bugs.mysql.com/7705])
- Fixed a problem with calculation of number of columns in row comparison against subquery. (Bug #8020 [http://bugs.mysql.com/8020])
- Fixed erroneous output resulting from `SELECT DISTINCT` combined with a subquery and `GROUP BY`. (Bug #7946 [http://bugs.mysql.com/7946])
- Fixed server crash in comparing a nested row expression (for example `row(1,(2,3))`) with a subquery. (Bug #8022 [http://bugs.mysql.com/8022])
- Fixed server crash resulting from certain correlated subqueries with forward references (references to an alias defined later in the outer query). (Bug #8025 [http://bugs.mysql.com/8025])
- Fixed server crash resulting from re-execution of prepared statements containing subqueries. (Bug #8125 [http://bugs.mysql.com/8125])
- Fixed a bug where `ALTER TABLE` improperly would accept an index on a `TIMESTAMP` column that `CREATE TABLE` would reject. (Bug #7884 [http://bugs.mysql.com/7884])
- `SHOW CREATE TABLE` now reports `ENGINE=MEMORY` rather than `ENGINE=HEAP` for a `MEMORY` table (unless the `MYSQL323` SQL mode is enabled). (Bug #6659 [http://bugs.mysql.com/6659])
- Fixed a bug where the use of `GROUP_CONCAT()` with `HAVING` caused a server crash. (Bug #7769 [http://bugs.mysql.com/7769])
- Fixed a bug where comparing the result of a subquery to a non-existent column caused a server crash on Windows. (Bug #7885 [http://bugs.mysql.com/7885])
- Fixed a bug in a combination of `-not` and `trunc*` operators of full-text search. Using more than one truncated negative search term, was causing empty result set.
- InnoDB: Corrected the handling of trailing spaces in the `ucs2` character set. (Bug #7350 [http://bugs.mysql.com/7350], Bug #8771 [http://bugs.mysql.com/8771])
- InnoDB: Use native `tmpfile()` function on Netware. All InnoDB temporary files are created under `sys:\tmp`. Previously, InnoDB temporary files were never deleted on Netware.
- Fixed a bug in `max_heap_table_size` handling, that resulted in `Table is full` error when the table was still smaller than the limit. (Bug #7791 [http://bugs.mysql.com/7791]).
- Fixed a symlink vulnerability in the **mysqlaccess** script. Reported by Javier Fernandez-Sanguino Pena and Debian Security Audit Team [http://www.debian.org/security/audit]. (CAN-2005-0004 [http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2005-0004])
- Fixed a bug that caused server crash if some error occured during filling of temporary table created for derived table or view handling. (Bug #7413 [http://bugs.mysql.com/7413])
- Fixed a bug which caused server crash if query containing `CONVERT_TZ()` function with constant

arguments was prepared. (Bug #6849 [http://bugs.mysql.com/6849])

- Prevent adding `CREATE TABLE .. SELECT` query to the binary log when the insertion of new records partially failed. (Bug #6682 [http://bugs.mysql.com/6682])

- Fixed a bug which caused a crash when only the slave I/O thread was stopped and started. (Bug #6148 [http://bugs.mysql.com/6148])

- Giving **mysqld** a `SIGHUP` caused it to crash.

- Changed semantics of `CREATE/ALTER/DROP DATABASE` statements so that replication of `CREATE DATABASE` is possible when using `--binlog-do-db` and `--binlog-ignore-db`. (Bug #6391 [http://bugs.mysql.com/6391])

- A sequence of `BEGIN` (or `SET AUTOCOMMIT=0`), `FLUSH TABLES WITH READ LOCK`, transactional update, `COMMIT`, `FLUSH TABLES WITH READ LOCK` could hang the connection forever and possibly the MySQL server itself. This happened for example when running the `innobackup` script several times. (Bug #6732 [http://bugs.mysql.com/6732])

- **mysqlbinlog** did not print `SET PSEUDO_THREAD_ID` statements in front of `LOAD DATA INFILE` statements inserting into temporary tables, thus causing potential problems when rolling forward these statements after restoring a backup. (Bug #6671 [http://bugs.mysql.com/6671])

- InnoDB: Fixed a bug no error message for ALTER with InnoDB and AUTO_INCREMENT (Bug #7061 [http://bugs.mysql.com/7061]). `InnoDB` now supports `ALTER TABLE...AUTO_INCREMENT = x` query to set auto increment value for a table.

- Made the MySQL server accept executing `SHOW CREATE DATABASE` even if the connection has an open transaction or locked tables; refusing it made **mysqldump --single-transaction** sometimes fail to print a complete `CREATE DATABASE` statement for some dumped databases. (Bug #7358 [http://bugs.mysql.com/7358])

- Fixed that, when encountering a ``disk full'' or ``quota exceeded'' write error, `MyISAM` sometimes didn't sleep and retry the write, thus resulting in a corrupted table. (Bug #7714 [http://bugs.mysql.com/7714])

- Fixed that `--expire-log-days` was not honored if using only transactions. (Bug #7236 [http://bugs.mysql.com/7236])

- Fixed that a slave could crash after replicating many `ANALYZE TABLE`, `OPTIMIZE TABLE`, or `REPAIR TABLE` statements from the master. (Bug #6461 [http://bugs.mysql.com/6461], Bug #7658 [http://bugs.mysql.com/7658])

- **mysqlbinlog** forgot to add backquotes around the collation of user variables (causing later parsing problems as `BINARY` is a reserved word). (Bug #7793 [http://bugs.mysql.com/7793])

- Ensured that **mysqldump --single-transaction** sets its transaction isolation level to `REPEATABLE READ` before proceeding (otherwise if the MySQL server was configured to run with a default isolation level lower than `REPEATABLE READ` it could give an inconsistent dump). (Bug #7850 [http://bugs.mysql.com/7850])

- Fixed that when using the `RPAD()` function (or any function adding spaces to the right) in a query that had to be resolved by using a temporary table, all resulting strings had rightmost spaces removed (i.e. `RPAD()` did not work) (Bug #4048 [http://bugs.mysql.com/4048])

- Fixed that a 5.0.3 slave can connect to a master < 3.23.50 without hanging (the reason for the hang is a bug in these quite old masters -- `SELECT @@unknown_var` hangs them -- which was fixed in MySQL 3.23.50). (Bug #7965 [http://bugs.mysql.com/7965])

- InnoDB: Fixed a deadlock without any locking, simple select and update (Bug #7975 [http://bugs.mysql.com/7975]). `InnoDB` now takes an exclusive lock when `INSERT ON DUPLICATE KEY UPDATE` is checking duplicate keys.
- Fixed a bug where MySQL was allowing concurrent updates (inserts, deletes) to a table if binary logging is enabled. Changed to ensure that all updates are executed in a serialized fashion, because they are executed serialized when binlog is replayed. (Bug #7879 [http://bugs.mysql.com/7879])
- Fixed a rare race condition which could lead to `FLUSH TABLES WITH READ LOCK` hanging. (Bug #8682 [http://bugs.mysql.com/8682])
- Fixed a bug that caused the slave to stop on statements that produced an error on the master. (Bug #8412 [http://bugs.mysql.com/8412])

D.1.3. Changes in release 5.0.2 (01 Dec 2004)

Functionality added or changed:

- Added support for the `INFORMATION_SCHEMA` ``information database'' that provides database metadata. See Chapter 21, *The `INFORMATION_SCHEMA` Information Database*.
- A `HAVING` clause in a `SELECT` statement now can refer to columns in the `GROUP BY` clause, as required by standard SQL.
- Added the `CREATE USER` and `RENAME USER` statements.
- Modify `DROP USER` so that it drops the account, including all its privileges. Formerly, it removed the account record only for an account that had had all privileges revoked.
- **Warning: Incompatible change!** The precedence of `NOT` operator has changed so that expressions such as `NOT a BETWEEN b AND c` are parsed correctly as `NOT (a BETWEEN b AND c)` rather than as `(NOT a) BETWEEN b AND c`. The pre-5.0 higher-precedence behavior can be obtained by enabling the new `HIGH_NOT_PRECEDENCE` SQL mode.
- Added `IS [NOT] boolean_value` syntax, where `boolean_value` is `TRUE`, `FALSE`, or `UNKNOWN`.
- Added several `InnoDB` status variables. See Section 5.2.4, “Server Status Variables”.
- Implemented the `WITH CHECK OPTION` clause for `CREATE VIEW`.
- `CHECK TABLE` now works for views.
- The `SCHEMA` and `SCHEMAS` keywords are now accepted as synonyms for `DATABASE` and `DATABASES`.
- Added initial support for rudimentary triggers (the `CREATE TRIGGER` and `DROP TRIGGER` statements).
- Added basic support for read-only server side cursors.
- **mysqldump --single-transaction --master-data** is now able to take an online (non-blocking) dump of InnoDB and report the corresponding binary log coordinates, which makes a backup suitable for point-in-time recovery, roll-forward or replication slave creation. See Section 8.8, “The **mysqldump** Database Backup Program”.
- Added `--start-datetime`, `--stop-datetime`, `--start-position`, `-`

-stop-position options to **mysqlbinlog** (makes point-in-time recovery easier).

- Made the MySQL server not react to signals SIGHUP and SIGQUIT on Mac OS X 10.3. This is needed because under this OS, the MySQL server receives lots of these signals (reported as Bug #2030 [http://bugs.mysql.com/2030]).

- New --auto-increment-increment and --auto-increment-offset startup options. These allow you to set up a server to generate auto-increment values that don't conflict with another server.

- MySQL now by default checks dates and in strict mode allows only fully correct dates. If you want MySQL to behave as before, you should enable the new ALLOW_INVALID_DATES SQL mode.

- Added STRICT_TRANS_TABLES, STRICT_ALL_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, and TRADITIONAL SQL modes. The TRADITIONAL mode is shorthand for all the preceding modes. When using mode TRADITIONAL, MySQL generates an error if you try to insert a wrong value in a column. It does not adjust the value to the closest possible legal value.

- MySQL now remembers which columns were declared to have default values. In STRICT_TRANS_TABLES/STRICT_ALL_TABLES mode, you now get an error if you do an INSERT without specifying all columns that don't have a default value. A side effect of this is that when you do SHOW CREATE for a new table, you no longer see a DEFAULT value for a column for which you didn't specify a default value.

- The compilation flag DONT_USE_DEFAULT_FIELDS was removed because you can get the same behavior by setting the sql_mode system variable to STRICT_TRANS_TABLES.

- Added NO_AUTO_CREATE_USER SQL mode to prevent GRANT from automatically creating new users if it would otherwise do so, unless a password also is specified.

- We now detect too-large floating point numbers during statement parsing and generate an error messages for them.

- Renamed the sql_updatable_view_key system variable to updatable_views_with_limit. This variable now can have only two values:

  - 1 or YES: Don't issue an error message (warning only) if a VIEW without presence of a key in the underlying table is used in queries with a LIMIT clause for updating. (This is the default value.)

  - 0 or NO: Prohibit update of a VIEW, which does not contain a key in the underlying table and the query uses a LIMIT clause (usually get from GUI tools).

- Reverted output format of SHOW TABLES to old pre-5.0.1 format that did not include a table type column. To get the additional column that lists the table type, use SHOW FULL TABLES now.

- The **mysql_fix_privilege_tables** script now initializes the global CREATE VIEW and SHOW VIEW privileges in the user table to the value of the CREATE privilege in that table.

- If the server finds that the user table has not been upgraded to include the view-related privilege columns, it treats each account as having view privileges that are the same as its CREATE privilege.

- InnoDB: If you specify the option innodb_locks_unsafe_for_binlog in my.cnf, InnoDB in an UPDATE or a DELETE only locks the rows that it updates or deletes. This greatly reduces the probability of deadlocks.

- A connection doing a rollback now displays "Rolling back" in the State column of SHOW PROCESSLIST.

- **mysqlbinlog** now prints an informative commented line (thread id, timestamp, server id, etc) before each `LOAD DATA INFILE`, like it does for other queries; unless `--short-form` is used.

Bugs fixed:

- Fixed that **mysqlbinlog --read-from-remote-server** sometimes couldn't accept two binary logfiles on the command line. (Bug #4507 [http://bugs.mysql.com/4507])
- Fixed that **mysqlbinlog --position --read-from-remote-server** had incorrect `# at` lines. (Bug #4506 [http://bugs.mysql.com/4506])
- Fixed that `CREATE TABLE ... TYPE=HEAP ... AS SELECT...` caused replication slave to stop. (Bug #4971 [http://bugs.mysql.com/4971])
- Fixed that `mysql_options(...,MYSQL_OPT_LOCAL_INFILE,...)` failed to disable `LOAD DATA LOCAL INFILE`. (Bug #5038 [http://bugs.mysql.com/5038])
- Fixed that `disable-local-infile` option had no effect if client read it from a configuration file using `mysql_options(...,MYSQL_READ_DEFAULT,...)`. (Bug #5073 [http://bugs.mysql.com/5073])
- Fixed that `SET GLOBAL SYNC_BINLOG` did not work on some platforms (Mac OS X). (Bug #5064 [http://bugs.mysql.com/5064])
- Fixed that **mysql-test-run** failed on the `rpl_trunc_binlog` test if running test from the installed (the target of 'make install') directory. (Bug #5050)
- Fixed that **mysql-test-run** failed on the `grant_cache` test when run as Unix user 'root'. (Bug #4678 [http://bugs.mysql.com/4678])
- Fixed an unlikely deadlock which could happen when using `KILL`. (Bug #4810)
- Fixed a crash when one connection got `KILL`ed while it was doing `START SLAVE`. (Bug #4827 [http://bugs.mysql.com/4827])
- Made `FLUSH TABLES WITH READ LOCK` block `COMMIT` if server is running with binary logging; this ensures that the binary log position can be trusted when doing a full backup of tables and the binary log. (Bug #4953 [http://bugs.mysql.com/4953])
- Fixed that the counter of an `auto_increment` column was not reset by `TRUNCATE TABLE` is the table was a temporary one. (Bug #5033 [http://bugs.mysql.com/5033])
- Fixed slave SQL thread so that the `SET COLLATION_SERVER...` statements it replicates don't advance its position (so that if it gets interrupted before the actual update query, it later redoes the `SET`). (Bug #5705 [http://bugs.mysql.com/5705])
- Fixed that if the slave SQL thread found a syntax error in a query (which should be rare, as the master parsed it successfully), it stops. (Bug #5711 [http://bugs.mysql.com/5711])
- Fixed that if a write to a MyISAM table fails because of a full disk or an exceeded disk quota, it prints a message to the error log every 10 minutes, and waits until disk becomes free. (Bug #3248 [http://bugs.mysql.com/3248])
- Fixed problem introduced in 4.0.21 where a connection starting a transaction, doing updates, then `FLUSH TABLES WITH READ LOCK`, then `COMMIT`, would cause replication slaves to stop (complaining about error 1223). Bug surfaced when using the InnoDB `innobackup` script. (Bug #5949 [http://bugs.mysql.com/5949])

- `OPTIMIZE TABLE`, `REPAIR TABLE`, and `ANALYZE TABLE` are now replicated without any error code in the binary log. (Bug #5551 [http://bugs.mysql.com/5551])
- If a connection had an open transaction but had done no updates to transactional tables (for example if had just done a `SELECT FOR UPDATE` then executed a non-transactional update, that update automatically committed the transaction (thus releasing InnoDB's row-level locks etc). (Bug #5714 [http://bugs.mysql.com/5714])
- If a connection was interrupted by a network error and did a rollback, the network error code got stored into the `BEGIN` and `ROLLBACK` binary log events; that caused superfluous slave stops. (Bug #6522 [http://bugs.mysql.com/6522])
- Fixed a bug which prevented **mysqlbinlog** from being able to read from `stdin`, for example, when piping the output from **zcat** to **mysqlbinlog**. (Bug #7853 [http://bugs.mysql.com/7853])

D.1.4. Changes in release 5.0.1 (27 Jul 2004)

Note: This build passes our test suite and fixes a lot of reported bugs found in the previous 5.0.0 release. However, please be aware that this is not a ``standard MYSQL build'' in the sense that there are still some open critical bugs in our bugs database at http://bugs.mysql.com/ that affect this release as well. We are actively fixing these and will make a new release where these are fixed as soon as possible. However, this binary should be a good candidate for testing new MySQL 5.0 features for future products.

Functionality added or changed:

- When installing a MySQL server as a Windows service, the installation command can include a `--local-service` option following the service name to cause the server to run using the `LocalService` Windows account that has limited privileges. This is in addition to the `--defaults-file` option that also can be given following the service name.
- Added support for read-only and updatable views based on a single table or other updatable views. View use requires that you upgrade your grant tables to add the view-related privileges. See Section 2.10.7, “Upgrading the Grant Tables”.
- Implemented a new ``greedy search'' optimizer that can significantly reduce the time spent on query optimization for some many-table joins. (You are affected if not only some particular `SELECT` is slow, but even using `EXPLAIN` for it takes a noticeable amount of time.) Two new system variables, `optimizer_search_depth` and `optimizer_prune_level`, can be used to fine-tune optimizer behavior.
- **Warning: Incompatible change!** C API change: `mysql_shutdown()` now requires a second argument. This is a source-level incompatibility that affects how you compile client programs; it does not affect the ability of compiled clients to communicate with older servers. See Section 22.2.3.54, “`mysql_shutdown()`”.
- A stored procedure is no longer ``global.'' That is, it now belongs to a specific database:
  - When a database is dropped, all routines belonging to that database are also dropped.
  - Procedure names may be qualified, for example, `db.p()`
  - When executed from another database, an implicit `USE` *`db_name`* is in effect.
  - Explicit `USE` *`db_name`* statements no longer are allowed in a stored procedure.

See Chapter 19, *Stored Procedures and Functions*.

- Fixed `SHOW TABLES` output field name and values according to standard. Field name changed from `Type` to `table_type`, values are `BASE TABLE`, `VIEW` and `ERROR`. (Bug #4603 [http://bugs.mysql.com/4603])
- Added the `sql_updatable_view_key` system variable.
- Added the `--replicate-same-server-id` server option.
- Added `Last_query_cost` status variable that reports optimizer cost for last compiled query.
- Added the `--to-last-log` option to **mysqlbinlog**, for use in conjunction with `--read-from-remote-server`.
- Added the `--innodb-safe-binlog` server option, which adds consistency guarantees between the content of `InnoDB` tables and the binary log. See Section 5.9.4, “The Binary Log”.
- `OPTIMIZE TABLE` for `InnoDB` tables is now mapped to `ALTER TABLE` instead of `ANALYZE TABLE`.
- `sync_frm` is now a settable global variable (not only a startup option).
- For replication of `MEMORY` (`HEAP`) tables: Made the master automatically write a `DELETE FROM` statement to its binary log when a `MEMORY` table is opened for the first time since master's startup. This is for the case where the slave has replicated a non-empty `MEMORY` table, then the master is shut down and restarted: the table is now empty on master; the `DELETE FROM` empties it on slave too. Note that even with this fix, between the master's restart and the first use of the table on master, the slave still has out-of-date data in the table. But if you use the `--init-file` option to populate the `MEMORY` table on the master at startup, it ensures that the failing time interval is zero. (Bug #2477 [http://bugs.mysql.com/2477])
- When a session having open temporary tables terminates, the statement automatically written to the binary log is now `DROP TEMPORARY TABLE IF EXISTS` instead of `DROP TEMPORARY TABLE`, for more robustness.
- The MySQL server now returns an error if `SET SQL_LOG_BIN` is issued by a user without the `SUPER` privilege (in previous versions it just silently ignored the statement in this case).
- Changed that when the MySQL server has binary logging disabled (that is, no `--log-bin` option was used), then no transaction binary log cache is allocated for connections. This should save `binlog_cache_size` bytes of memory (32KB by default) for every connection.
- Added the `sync_binlog=N` global variable and startup option, which makes the MySQL server synchronize its binary log to disk (`fdatasync()`) after every Nth write to the binary log.
- Changed the slave SQL thread to print less useless error messages (no more message duplication; no more messages when an error is skipped because of `slave-skip-errors`).
- `DROP DATABASE IF EXISTS`, `DROP TABLE IF EXISTS`, single-table `DELETE`, and single-table `UPDATE` now are written to the binary log even if they changed nothing on the master (for example, even if a `DELETE` matched no rows). The old behavior sometimes caused bad surprises in replication setups.
- Replication and **mysqlbinlog** now have better support for the case that the session character set and collation variables are changed within a given session. See Section 6.7, “Replication Features and Known Problems”.

- Killing a CHECK TABLE statement does not result in the table being marked as ``corrupted'' any more; the table remains as if CHECK TABLE had not even started. See Section 13.5.5.3, “KILL Syntax”.

Bugs fixed:

- Strange results with index (x, y) ... WHERE x=val_1 AND y>=val_2 ORDER BY pk; (Bug #3155 [http://bugs.mysql.com/3155])
- Subquery and order by (Bug #3118 [http://bugs.mysql.com/3118])
- ALTER DATABASE caused the client to hang if the database did not exist. (Bug #2333 [http://bugs.mysql.com/2333])
- SLAVE START (which is a deprecated syntax, START SLAVE should be used instead) could crash the slave. (Bug #2516 [http://bugs.mysql.com/2516])
- Multiple-table DELETE statements were never replicated by the slave if there were any --replicate-*-table options. (Bug #2527 [http://bugs.mysql.com/2527])
- The MySQL server did not report any error if a statement (submitted through mysql_real_query() or mysql_stmt_prepare()) was terminated by garbage characters. This can happen if you pass a wrong length parameter to these functions. The result was that the garbage characters were written into the binary log. (Bug #2703 [http://bugs.mysql.com/2703])
- Replication: If a client connects to a slave server and issues an administrative statement for a table (for example, OPTIMIZE TABLE or REPAIR TABLE), this could sometimes stop the slave SQL thread. This does not lead to any corruption, but you must use START SLAVE to get replication going again. (Bug #1858 [http://bugs.mysql.com/1858])
- Made clearer the error message that one gets when an update is refused because of the --read-only option. (Bug #2757 [http://bugs.mysql.com/2757])
- Fixed that --replicate-wild-*-table rules apply to ALTER DATABASE when the table pattern is %, as is the case for CREATE DATABASE and DROP DATABASE. (Bug #3000 [http://bugs.mysql.com/3000])
- Fixed that when a Rotate event is found by the slave SQL thread in the middle of a transaction, the value of Relay_Log_Pos in SHOW SLAVE STATUS remains correct. (Bug #3017 [http://bugs.mysql.com/3017])
- Corrected the master's binary log position that InnoDB reports when it is doing a crash recovery on a slave server. (Bug #3015 [http://bugs.mysql.com/3015])
- Changed the column Seconds_Behind_Master in SHOW SLAVE STATUS to never show a value of #1. (Bug #2826 [http://bugs.mysql.com/2826])
- Changed that when a DROP TEMPORARY TABLE statement is automatically written to the binary log when a session ends, the statement is recorded with an error code of value zero (this ensures that killing a SELECT on the master does not result in a superfluous error on the slave). (Bug #3063 [http://bugs.mysql.com/3063])
- Changed that when a thread handling INSERT DELAYED (also known as a delayed_insert thread) is killed, its statements are recorded with an error code of value zero (killing such a thread does not endanger replication, so we thus avoid a superfluous error on the slave). (Bug #3081 [http://bugs.mysql.com/3081])

- Fixed deadlock when two `START SLAVE` commands were run at the same time. (Bug #2921 [http://bugs.mysql.com/2921])
- Fixed that a statement never triggers a superfluous error on the slave, if it must be excluded given the `--replicate-*` options. The bug was that if the statement had been killed on the master, the slave would stop. (Bug #2983 [http://bugs.mysql.com/2983])
- The `--local-load` option of **mysqlbinlog** now requires an argument.
- Fixed a segmentation fault when running `LOAD DATA FROM MASTER` after `RESET SLAVE`. (Bug #2922 [http://bugs.mysql.com/2922])
- **mysqlbinlog --read-from-remote-server** read all binary logs following the one that was requested. It now stops at the end of the requested file, the same as it does when reading a local binary log. There is an option `--to-last-log` to get the old behavior. (Bug #3204 [http://bugs.mysql.com/3204])
- Fixed **mysqlbinlog --read-from-remote-server** to print the exact positions of events in the "at #" lines. (Bug #3214 [http://bugs.mysql.com/3214])
- Fixed a rare error condition that caused the slave SQL thread spuriously to print the message `Binlog has bad magic number` and stop when it was not necessary to do so. (Bug #3401 [http://bugs.mysql.com/3401])
- Fixed **mysqlbinlog** not to forget to print a `USE` statement under rare circumstances where the binary log contained a `LOAD DATA INFILE` statement. (Bug #3415 [http://bugs.mysql.com/3415])
- Fixed a memory corruption when replicating a `LOAD DATA INFILE` when the master had version 3.23. (Bug #3422 [http://bugs.mysql.com/3422])
- Multiple-table `DELETE` statements were always replicated by the slave if there were some `--replicate-*-ignore-table` options and no `--replicate-*-do-table` options. (Bug #3461 [http://bugs.mysql.com/3461])
- Fixed a crash of the MySQL slave server when it was built with `--with-debug` and replicating itself. (Bug #3568 [http://bugs.mysql.com/3568])
- Fixed that in some replication error messages, a very long query caused the rest of the message to be invisible (truncated), by putting the query last in the message. (Bug #3357 [http://bugs.mysql.com/3357])
- If `server-id` was not set using startup options but with `SET GLOBAL`, the replication slave still complained that it was not set. (Bug #3829 [http://bugs.mysql.com/3829])
- **mysql_fix_privilege_tables** didn't correctly handle the argument of its `--password=#` option. (Bug #4240 [http://bugs.mysql.com/4240])
- Fixed potential memory overrun in `mysql_real_connect()` (which required a compromised DNS server and certain operating systems). (Bug #4017 [http://bugs.mysql.com/4017])
- During the installation process of the server RPM on Linux, **mysqld** was run as the `root` system user, and if you had `--log-bin=<somewhere_out_of_var_lib_mysql>` it created binary log files owned by `root` in this directory, which remained owned by `root` after the installation. This is now fixed by starting **mysqld** as the `mysql` system user instead. (Bug #4038 [http://bugs.mysql.com/4038])
- Made `DROP DATABASE` honor the value of `lower_case_table_names`. (Bug #4066 [http://bugs.mysql.com/4066])

- The slave SQL thread refused to replicate `INSERT ... SELECT` if it examined more than 4 billion rows. (Bug #3871 [http://bugs.mysql.com/3871])
- **mysqlbinlog** didn't escape the string content of user variables, and did not deal well when these variables were in non-ASCII character sets; this is now fixed by always printing the string content of user variables in hexadecimal. The character set and collation of the string is now also printed. (Bug #3875 [http://bugs.mysql.com/3875])
- Fixed incorrect destruction of expression that led to a server crash on complex `AND`/`OR` expressions if query was ignored (either by a replication server because of `--replicate-*-table` rules, or by any MySQL server because of a syntax error). (Bug #3969 [http://bugs.mysql.com/3969], Bug #4494 [http://bugs.mysql.com/4494])
- If `CREATE TEMPORARY TABLE t SELECT` failed while loading the data, the temporary table was not dropped. (Bug #4551 [http://bugs.mysql.com/4551])
- Fixed that when a multiple-table `DROP TABLE` failed to drop a table on the master server, the error code was not written to the binary log. (Bug #4553 [http://bugs.mysql.com/4553])
- When the slave SQL thread was replicating a `LOAD DATA INFILE` statement, it didn't show the statement in the output of `SHOW PROCESSLIST`. (Bug #4326 [http://bugs.mysql.com/4326])

D.1.5. Changes in release 5.0.0 (22 Dec 2003: Alpha)

Functionality added or changed:

- Added support for `SUM(DISTINCT)`, `MIN(DISTINCT)`, and `MAX(DISTINCT)`.
- The `KILL` statement now takes `CONNECTION` and `QUERY` modifiers. The first is the same as `KILL` with no modifier (it kills a given connection thread). The second kills only the statement currently being executed by the connection.
- Added `TIMESTAMPADD()` and `TIMESTAMPDIFF()` functions.
- Added `WEEK` and `QUARTER` values as `INTERVAL` arguments for the `DATE_ADD()` and `DATE_SUB()` functions.
- New binary log format that enables replication of these session variables: `sql_mode`, `SQL_AUTO_IS_NULL`, `FOREIGN_KEY_CHECKS` (which was replicated since 4.0.14, but here it's done more efficiently and takes less space in the binary logs), `UNIQUE_CHECKS`. Other variables (like character sets, `SQL_SELECT_LIMIT`, ...) will be replicated in upcoming 5.0.x releases.
- Implemented Index Merge optimization for `OR` clauses. See Section 7.2.6, “Index Merge Optimization”.
- Basic support for stored procedures (SQL:2003 style). See Chapter 19, *Stored Procedures and Functions*.
- Added `SELECT INTO` *`list_of_vars`*, which can be of mixed (that is, global and local) types. See Section 19.2.9.3, “`SELECT ... INTO` Statement”.
- Easier replication upgrade (5.0.0 masters can read older binary logs and 5.0.0 slaves can read older relay logs). See Section 6.5, “Replication Compatibility Between MySQL Versions” for more details). The format of the binary log and relay log is changed compared to that of MySQL 4.1 and older.

- **Important note:** If you upgrade to MySQL 4.1.1 or higher, it is difficult to downgrade back to 4.0 or 4.1.0! That is because, for earlier versions, `InnoDB` is not aware of multiple tablespaces.

Bugs fixed:

D.2. Changes in release 4.1.x (Production)

Version 4.1 of the MySQL server includes many enhancements and new features. Binaries for this version are available for download at http://dev.mysql.com/downloads/mysql-4.1.html.

- Subqueries and derived tables (unnamed views). See Section 13.1.8, “Subquery Syntax”.
- `INSERT ... ON DUPLICATE KEY UPDATE ...` syntax. This allows you to `UPDATE` an existing row if the insert would cause a duplicate value in a `PRIMARY` or `UNIQUE` key. (`REPLACE` allows you to overwrite an existing row, which is something entirely different.) See Section 13.1.4, “`INSERT` Syntax”.
- A newly designed `GROUP_CONCAT()` aggregate function. See Section 12.9, “Functions and Modifiers for Use with `GROUP BY` Clauses”.
- Extensive Unicode (UTF8) support.
- Table names and column names now are stored in `UTF8`. This makes MySQL more flexible, but might cause some problems upgrading if you have table or column names that use characters outside of the standard 7-bit US-ASCII range. See Section 2.10.2, “Upgrading from Version 4.0 to 4.1”.
- Character sets can be defined per column, table, and database.
- New key cache for `MyISAM` tables with many tunable parameters. You can have multiple key caches, preload index into caches for batches...
- `BTREE` index on `HEAP` tables.
- Support for OpenGIS spatial types (geographical data). See Chapter 18, *Spatial Extensions in MySQL*.
- `SHOW WARNINGS` shows warnings for the last command. See Section 13.5.4.20, “`SHOW WARNINGS` Syntax”.
- Faster binary protocol with prepared statements and parameter binding. See Section 22.2.4, “C API Prepared Statements”.
- You can now issue multiple statements with a single C API call and then read the results in one go. See Section 22.2.9, “C API Handling of Multiple Query Execution”.
- Create Table: `CREATE [TEMPORARY] TABLE [IF NOT EXISTS] table2 LIKE table1`.
- Server based `HELP` command that can be used in the **mysql** command-line client (and other clients) to get help for SQL statements.

For a full list of changes, please refer to the changelog sections for each individual 4.1.x release.

D.2.1. Changes in release 4.1.11 (not released yet)

Functionality added or changed:

- `ONLY_FULL_GROUP_BY` no longer is included in the `ANSI` composite SQL mode. (Bug #8510 [http://bugs.mysql.com/8510])
- **mysqld_safe** will create the directory where the UNIX socket file is to be located if the directory does not exist. This applies only to the last component of the directory pathname. (Bug #8513 [http://bugs.mysql.com/8513])
- The coercibility for the return value of functions such as `USER()` or `VERSION()` now is ``system constant'' rather than ``implicit.'' This makes these functions more coercible than column values so that comparisons of the two do not result in `Illegal mix of collations` errors. `COERCIBILITY()` was modified to accommodate this new coercibility value. See Section 12.8.3, “Information Functions”.
- User variable coercibility has been changed from ``coercible'' to ``implicit.'' That is, user variables have the same coercibility as column values.
- Modified the parser to allow `SELECT` statements following the `UNION` keyword to be subqueries in parentheses. (Bug #2435 [http://bugs.mysql.com/2435])
- For slave replication servers started with `--replicate-*` options, statements that should not be replicated according those options no longer are written to the slave's general query log. (Bug #8297 [http://bugs.mysql.com/8297])
- Added `SQL_NOTES` session variable to cause `Note`-level warnings not to be recorded. (Bug #6662 [http://bugs.mysql.com/6662])
- `InnoDB`: Commit after every 10,000 copied rows when executing `CREATE INDEX`, `DROP INDEX` or `OPTIMIZE TABLE`, which are internally implemented as `ALTER TABLE`. This makes it much faster to recover from an aborted operation.
- Added a new global system variable `slave_transaction_retries`: If the replication slave SQL thread fails to execute a transaction because of an `InnoDB` deadlock or exceeded `innodb_lock_wait_timeout`, it automatically retries `slave_transaction_retries` times before stopping with an error. The default in MySQL 4.1 is 0. You must explicitly set the value greater than 0 to enable the ``retry'' behavior. (In MySQL 5.0.3 or newer, the default is 10.) (Bug #8325 [http://bugs.mysql.com/8325])

Bugs fixed:

- Fixed server crash resulting from queries that combined `SELECT DISTINCT`, `SUM()`, and `ROLLUP`. (Bug #8615 [http://bugs.mysql.com/8615])
- Incorrect results were returned from queries that combined `SELECT DISTINCT`, `GROUP BY` , and `ROLLUP`. (Bug #8616 [http://bugs.mysql.com/8616])
- Too many rows were returned from queries that combined `ROLLUP` and `LIMIT` if `SQL_CALC_FOUND_ROWS` was given. (Bug #8617 [http://bugs.mysql.com/8617])
- **mysqldump** misinterpreted '_' and '%' characters in the names of tables to be dumped as wildcard characters. (Bug #9123 [http://bugs.mysql.com/9123])
- Made the `relay_log_space_limit` system variable show up in the output of `SHOW VARIABLES`. (Bug #7100 [http://bugs.mysql.com/7100])

- Use of GROUP_CONCAT(*x*) in a subquery, where *x* was an alias to a column in the outer query, resulted in a server crash. (Bug #8656 [http://bugs.mysql.com/8656])
- The CHARSET(), COLLATION(), and COERCIBILITY() functions sometimes returned NULL. CHARSET() and COLLATION() returned NULL when given any of these arguments that evaluated to NULL: A system function such as DATABASE(); a column value; and a user variable. Now CHARSET() and COLLATION() return the system character set and collation; the column character set and collation; and binary. COERCIBILITY(NULL) now returns ``ignorable'' coercibility rather than NULL. (Bug #9129 [http://bugs.mysql.com/9129])
- Expressions involving nested CONCAT() calls and character set conversion of string constants could return an incorrect result. (Bug #8785 [http://bugs.mysql.com/8785])
- The MEMORY storage engine did not properly increment an AUTO_INCREMENT column if there was a second composite index that included the column. (Bug #8489 [http://bugs.mysql.com/8489])
- Fixed a bug in the filesort routine such that killing a filesort could cause an assertion failure. (Bug #8799 [http://bugs.mysql.com/8799])
- REPAIR TABLE did not invalidate query results in the query cache that were generated from the table. (Bug #8480 [http://bugs.mysql.com/8480])
- If max_join_size was set, a query containing a subquery that exceeded the examined-rows limit could hang. (Bug #8726 [http://bugs.mysql.com/8726])
- Mixed-case database and table names in the grant tables were ignored for authentication if the lower_case_table_names system variable was set. GRANT will not create such privileges when lower_case_table_names is set, but it is possible to create them by direct manipulation of the grant tables, or that old grant records were present before setting the variable. (Bug #7989 [http://bugs.mysql.com/7989])
- AES_DECRYPT(*col_name*,*key*) could fail to return NULL for invalid values in *col_name*, if *col_name* was declared as NOT NULL. (Bug #8669 [http://bugs.mysql.com/8669])
- Ordering by unsigned expression (more complex than a column reference) was treating the value as signed, producing incorrectly sorted results. (Bug #7425 [http://bugs.mysql.com/7425])
- HAVING was treating unsigned columns as signed. (Bug #7425 [http://bugs.mysql.com/7425])
- Fixed a problem with boolean full-text searches on utf8 columns where a double quote in the search string caused a server crash. (Bug #8351 [http://bugs.mysql.com/8351])
- MIN(*col_name*) and MAX(*col_name*) could fail to produce the correct result if *col_name* was contained in multiple indexes and the optimizer did not choose the first index that contained the column. (Bug #8893 [http://bugs.mysql.com/8893])
- Table creation for a MyISAM table failed if DATA DIRECTORY or INDEX DIRECTORY options were given that specified the pathname to the database directory where the table files would be created by default. (Bug #8707 [http://bugs.mysql.com/8707])
- Fixed a problem with LIKE pattern-matching for strings with the cp1251_bin binary collation. (Bug #8560 [http://bugs.mysql.com/8560])
- A join on two tables failed when each contained a BIGINT UNSIGNED column that were compared when their values exceeded 2^63 # 1. The match failed and the join returned no rows. (Bug #8562 [http://bugs.mysql.com/8562])
- For a query with both GROUP BY and COUNT(DISTINCT) clauses and a FROM clause with a subquery, NULL was returned for any VARCHAR column selected by the subquery. (Bug #8218

[http://bugs.mysql.com/8218])

- Fixed an optimizer bug that caused incorrectly ordered result from a query that used a `FULLTEXT` index to retrieve rows and there was another index that was usable for `ORDER BY`. For such a query, `EXPLAIN` showed `fulltext` join type, but regular (not `FULLTEXT`) index in the `Key` column. (Bug #6635 [http://bugs.mysql.com/6635])

- For a statement string that contained multiple slow queries, only the last one would be written to the slow query log. (Bug #8475 [http://bugs.mysql.com/8475])

- When the server was started with `--skip-name-resolve`, specifying hostname values that included netmasks in `GRANT` statements did not work. (Bug #8471 [http://bugs.mysql.com/8471])

- The `--set-character-set` option for **myisamchk** was changed to `--set-collation`. The value needed for specifying how to sort indexes is a collation name, not a character set name. (Bug #8349 [http://bugs.mysql.com/8349])

- Hostname matching didn't work if a netmask was specified for table-specific privileges. (Bug #3309 [http://bugs.mysql.com/3309])

- Binary data stored in `BLOB` or `BINARY` columns would be erroneously dumped if **mysqldump** was invoked with `--hex-blob` and `--skip-extended-insert` arguments. This happened if data contained characters larger then 0x7F (Bug #8830 [http://bugs.mysql.com/8830]).

- Corruption of `MyISAM` table indexes could occur with `TRUNCATE TABLE` if the table had already been opened. For example, this was possible if the table had been opened implicitly by selecting from a `MERGE` table that mapped to the `MyISAM` table. The server now issues an error message for `TRUNCATE TABLE` under these conditions. (Bug #8306 [http://bugs.mysql.com/8306])

- Fixed handling of table-name matching in **mysqlhotcopy** to accommodate `DBD::mysql` 2.9003 and up (which implement identifier quoting). (Bug #8136 [http://bugs.mysql.com/8136])

- In the `mysql_real_escape_string()` C API function, when a multi-byte character is encountered that is illegal in the current character set, escape only the first byte, not each byte. This avoids creating a valid character from an invalid one. (Bug #8378 [http://bugs.mysql.com/8378])

- Fixed a problem with the `cp1250_czech_cs` collation that caused empty literal strings not to compare equal to empty character columns. (Bug #8840 [http://bugs.mysql.com/8840])

- Fixed a problem in index cost calculation that caused a `USE INDEX` or `FORCE INDEX` hint not to be used properly for a `LEFT JOIN` across indexed `BLOB` columns. (Bug #7520 [http://bugs.mysql.com/7520])

- The column type for `MAX(`*`datetime_col`*`)` was returned as `VARCHAR` rather than `DATETIME` if the query included a `GROUP BY` clause. (Bug #5615 [http://bugs.mysql.com/5615])

- `FOUND_ROWS()` returned an incorrect value for preceding `SELECT` statements that used no table or view. (Bug #6089 [http://bugs.mysql.com/6089])

- In string literals with an escape character ('\') followed by a multi-byte character that has a second byte of '\', the literal was not interpreted correctly. The next character now is escaped, not just the next byte. (Bug #8303 [http://bugs.mysql.com/8303])

- InnoDB: Work around a problem in AIX 5.1 patched with ML7 security patch: InnoDB would refuse to open its `ibdata` files, complaining about an operating system error 0.

- InnoDB: Fixed a memory corruption bug if one created a table with a primary key that contained at least two column prefixes. An example: `CREATE TABLE t(a char(100), b tinyblob, PRIMARY KEY(a(5), b(10)))`.

- `InnoDB`: Do not try to space-pad `BLOB` columns containing `ucs2` characters. This avoids an assertion failure that was introduced when fixing Bug #7350 [http://bugs.mysql.com/7350]. (Bug #8771 [http://bugs.mysql.com/8771])

- InnoDB: Fixed a bug: MySQL-4.1.8 - 4.1.10 could complain that an InnoDB table created with MySQL-3.23.49 or earlier was in the new compact InnoDB table format of 5.0.3 or later, and InnoDB would refuse to use that table. There is nothing wrong with the table, it is mysqld that is in error. Workaround: wait that 4.1.11 is released before doing an upgrade, or dump the table and recreate it with any MySQL version >= 3.23.50 before upgrading.

- `InnoDB`: Honor the `--tmpdir` startup option when creating temporary files. Previously, `InnoDB` temporary files were always created in the temporary directory of the operating system. On Netware, `InnoDB` will continue to ignore `--tmpdir`. (Bug #5822 [http://bugs.mysql.com/5822])

- **mysqldump** now avoids writing `SET NAMES` to the dump output if the server is older than version 4.1 and would not understand that statement. (Bug #7997 [http://bugs.mysql.com/7997])

- Fixed a bug in **my_print_defaults** that made it ignore the `--defaults-extra-file` and `--defaults-file` options.

- Retrieving from a view defined as a `SELECT` that mixed `UNION ALL` and `UNION DISTINCT` resulted in a different result than retrieving from the original `SELECT`. (Bug #6565 [http://bugs.mysql.com/6565])

- Worked around a bug in support for NSS support in `glibc` when static linking is used and LDAP is one of the NSS sources. The workaround is to detect when the bug causes a segfault and issue a diagnostic message with information about the problem. (Bug #3037 [http://bugs.mysql.com/3037], Bug #4872 [http://bugs.mysql.com/4872])

- If the **mysql** prompt was configured to display the default database name, and that database was dropped, **mysql** did not update the prompt. (Bug #4802 [http://bugs.mysql.com/4802])

- **perror** was printing a spurious extra line of output ("Error code ###: Unknown error ###" printed directly before the correct line with the error message). (Bug #8517 [http://bugs.mysql.com/8517])

- The `CHAR()` function was not ignoring `NULL` arguments, contrary to the documentation. (Bug #6317 [http://bugs.mysql.com/6317])

- Neither `SHOW ERRORS` nor `SHOW WARNINGS` were displaying Error-level messages. (Bug #6572 [http://bugs.mysql.com/6572])

- Creating a table using a name containing a character that is illegal in `character_set_client` resulted in the character being stripped from the name and no error. The character now is considered an error. (Bug #8041 [http://bugs.mysql.com/8041])

- Fixed a problem with the Cyrillic letters I and SHORT I being treated the same by the `utf8_general_ci` collation. (Bug #8385 [http://bugs.mysql.com/8385])

- The `MAX_CONNECTIONS_PER_HOUR` resource limit was not being reset hourly and thus imposed an absolute limit on number of connections per account until the server is restarted or the limits flushed. (Bug #8350 [http://bugs.mysql.com/8350])

- With a database was dropped with `lower_case_table_names=2`, tables in the database also were dropped but not being flushed properly from the table cache. If the database was recreated, the tables also would appear to have been recreated. (Bug #8355 [http://bugs.mysql.com/8355])

- Changed `mysql_server_end()` C API function to restore more variables to their initial state so that a subsequent call to `mysql_server_init()` would not cause a client program crash. (Bug #7344 [http://bugs.mysql.com/7344])

- Fixed a problem with accented letters improperly being treated as distinct with the `utf_general_ci` collation. (Bug #7878 [http://bugs.mysql.com/7878])
- `ENUM` and `SET` columns in privilege tables incorrectly had a case-sensitive collation, resulting in failure of assignments of values that did not have the same lettercase as given in the column definitions. The collation was changed to be case insensitive. (Bug #7617 [http://bugs.mysql.com/7617])
- An expression that tested a case-insensitive character column against string constants that differed in lettercase could fail because the constants were treated as having a binary collation. (For example, `WHERE city='London' AND city='london'` could fail.) (Bug #7098 [http://bugs.mysql.com/7098], Bug #8690 [http://bugs.mysql.com/8690])
- The output of the `STATUS` (`\s`) command in **mysql** had the values for the server and client character sets reversed. (Bug #7571 [http://bugs.mysql.com/7571])
- If the slave was running with `--replicate-*-table` options which excluded one temporary table and included another, and the two tables were used in a single `DROP TEMPORARY TABLE IF EXISTS` statement, as the ones the master automatically writes to its binary log upon client's disconnection when client has not explicitly dropped these, the slave could forget to delete the included replicated temporary table. Only the slave needs to be upgraded. (Bug #8055 [http://bugs.mysql.com/8055])
- Treat user variables as having `IMPLICIT` derivation (coercibility) to avoid ``Illegal mix of collations'' errors when replicating user variables. (Bug #6676 [http://bugs.mysql.com/6676])
- When setting integer system variables to a negative value with `SET VARIABLES`, the value was treated as a positive value modulo 2^32. (Bug #6958 [http://bugs.mysql.com/6958])
- Fixed a bug in bundled `readline` library that caused segmentation fault in **mysql** when user entered Shift+Enter. (Bug #5672 [http://bugs.mysql.com/5672])
- Fix conversion of strings -> double to get higher accuracy for floating point values that are integers, like: `123.45E+02` (Bug #7840 [http://bugs.mysql.com/7840]).
- Fixed a bug in `MATCH ... AGAINST` in natural language mode that could cause a server crash if the `FULLTEXT` index was not used in a join (`EXPLAIN` did not show `fulltext` join mode) and the search query matched no rows in the table (Bug #8522 [http://bugs.mysql.com/8522]).
- Platform and architecture information in version information produced for `--version` option on Windows was always `Win95/Win98 (i32)`. More accurately determine platform as `Win32` or `Win64` for 32-bit or 64-bit Windows, and architecture as `ia32` for x86, `ia64` for Itanium, and `axp` for Alpha. (Bug #4445 [http://bugs.mysql.com/4445])
- Fixed a rare race condition which could lead to `FLUSH TABLES WITH READ LOCK` hanging. (Bug #8682 [http://bugs.mysql.com/8682])
- Fixed a bug that caused the slave to stop on statements that produced an error on the master. (Bug #8412 [http://bugs.mysql.com/8412])
- If multiple semicolon-separated statements were received in a single packet, they were written to the binary log as a single event rather than as separate per-statement events. For a server serving as a replication master, this caused replication to fail when the event was sent to slave servers. (Bug #8436 [http://bugs.mysql.com/8436])

D.2.2. Changes in release 4.1.10 (12 Feb 2005)

Note: The security improvements related to creation of table files and to user-defined functions were

made after MySQL 4.1.10 was released and are present in MySQL 4.1.10a. We would like to thank Stefano Di Paola <stefano.dipaola@wisec.it> for making us aware of these.

Functionality added or changed:

- Security improvement: The server creates `.frm`, `.MYD`, `.MYI`, `.MRG`, `.ISD`, and `.ISM` table files only if a file with the same name does not already exist.
- Security improvement: User-defined functions should have at least one symbol defined in addition to the `xxx` symbol that corresponds to the main `xxx()` function. These auxiliary symbols correspond to the `xxx_init()`, `xxx_deinit()`, `xxx_reset()`, `xxx_clear()`, and `xxx_add()` functions. **mysqld** by default no longer loads UDFs unless they have at least one auxiliary symbol defined in addition to the main symbol. The `--allow-suspicious-udfs` option controls whether UDFs that have only an `xxx` symbol can be loaded. By default, the option is off. `mysqld` also checks UDF filenames when it reads them from the `mysql.func` table and rejects those that contain directory pathname separator characters. (It already checked names as given in `CREATE FUNCTION` statements.) See Section 25.2.3.1, “UDF Calling Sequences for simple functions”, Section 25.2.3.2, “UDF Calling Sequences for aggregate functions”, and Section 25.2.3.6, “User-defined Function Security Precautions”.
- Setting the connection collation to a value different from the server collation followed by a `CREATE TABLE` statement that included a quoted default value resulted in a server crash. (Bug #8235 [http://bugs.mysql.com/8235])
- Thread stack size was increased from 192KB to 256KB on Linux/IA-64 (too small stack size was causing server crashes on some queries). (Bug #8391 [http://bugs.mysql.com/8391])
- From the Windows distribution, predefined accounts without passwords for remote users ("root@%", "@%") were removed (other distributions never had them).
- Added `mysql_library_init()` and `mysql_library_end()` as synonyms for the `mysql_server_init()` and `mysql_server_end()` C API functions. `mysql_library_init()` and `mysql_library_end()` are `#define` symbols, but the names more clearly indicate that they should be called when beginning and ending use of a MySQL C API library no matter whether the application uses `libmysqlclient` or `libmysqld`. (Bug #6149 [http://bugs.mysql.com/6149])
- The server now issues a warning when `lower_case_table_names=2` and the data directory is on a case-sensitive filesystem, just as when `lower_case_table_names=0` on a case-insensitive filesystem. (Bug #7887 [http://bugs.mysql.com/7887])
- The server now issues a warning to the error log when it encounters older tables that contain character columns that might be interpreted by newer servers to have a different column length. (Bug #6913 [http://bugs.mysql.com/6913]) See Section 2.10.2, “Upgrading from Version 4.0 to 4.1” for a discussion of this problem and what to do about it.
- InnoDB: When MySQL/InnoDB is compiled on Mac OS X 10.2 or earlier, detect the operating system version at run time and use the `fcntl()` file flush method on Mac OS X versions 10.3 and later. Apple had disabled `fsync()` in Mac OS X for internal disk drives, which caused corruption at power outages.
- InnoDB: A shared record lock (`LOCK_REC_NOT_GAP`) is now taken for a matching record in the foreign key check because inserts can be allowed into gaps.
- InnoDB: Relaxed locking in INSERT...SELECT, single table UPDATE...SELECT and single table DELETE...SELECT clauses when innobase_locks_unsafe_for_binlog is used and isolation level of the transaction is not serializable. `InnoDB` uses consistent read in these cases for a selected table.

Bugs fixed:

- `FOUND_ROWS()` returned an incorrect value after a `SELECT SQL_CALC_FOUND_ROWS DISTINCT` statement that selected constants and included `GROUP BY` and `LIMIT` clauses. (Bug #7945 [http://bugs.mysql.com/7945])
- Fixed a bug in cardinality estimations for `HASH` indexes of `TEMPORARY` tables created using `MEMORY` storage engine. As a result queries that were using this index (as shown by `EXPLAIN`) could have returned incorrect results. (Bug #8371 [http://bugs.mysql.com/8371])
- Corrected a problem with references to `DUAL` where statements such as `SELECT 1 AS a FROM DUAL` would succeed but statements such as `SELECT 1 AS a FROM DUAL LIMIT 1` would fail. (Bug #8023 [http://bugs.mysql.com/8023])
- Fixed a server crash caused by `DELETE FROM` *`tbl_name`* `... WHERE ... ORDER BY` *`tbl_name.col_name`* when the `ORDER BY` column was qualified with the table name. (Bug #8392 [http://bugs.mysql.com/8392])
- **mysqld** had problems finding its language files if the `basedir` value was specified as a very long pathname. (Bug #8015 [http://bugs.mysql.com/8015])
- Updates were being written to the binary log when there were `binlog-do-db` or `binlog-ignore-db` options even when there was no current database, contrary to Section 5.9.4, “The Binary Log”. (Bug #6749 [http://bugs.mysql.com/6749])
- Fixed conversion of floating-point values to character fields when the absolute value of the float was less than 1, and also fixed calculation of length for negative values. (Bug #7774 [http://bugs.mysql.com/7774])
- Column headers in query results retrieved from the query cache could be corrupted when a non-4.1 client was served a result originally generated for a 4.1 client. The query cache was not keeping track of which client/server protocol was being used. (Bug #6511 [http://bugs.mysql.com/6511])
- Fixed `LOAD INDEX` statement to actually load index in memory. (Bug #8452 [http://bugs.mysql.com/8452])
- If multiple prepared statements were executed without retrieving their results, executing one of them again would cause the client program to crash. (Bug #8330 [http://bugs.mysql.com/8330])
- Non-numeric values inserted into a `YEAR` column were being stored as `2000` rather than as `0000`. (Bug #6067 [http://bugs.mysql.com/6067])
- Fixed a failure of multiple-table updates to replicate properly on slave servers when `--replicate-*-table` options had been specified. (Bug #7011 [http://bugs.mysql.com/7011])
- `mysql_stmt_close()` C API function was not clearing an error indicator when a previous prepare call failed, causing subsequent invocations of error-retrieving calls to indicate spurious error values. (Bug #7990 [http://bugs.mysql.com/7990])
- Fixed failure of `CREATE TABLE ... LIKE` Windows when the source or destination table was located in a symlinked database directory. (Bug #6607 [http://bugs.mysql.com/6607])
- With `lower_case_table_names` set to 1, **mysqldump** on Windows could write the same table name in different lettercase for different SQL statements. Fixed so that consistent lettercase is used. (Bug #5185 [http://bugs.mysql.com/5185]) `HAVING` that referred to `RAND()` or a user-defined function in the `SELECT` part through an alias could cause a crash or wrong value. (Bug #8216 [http://bugs.mysql.com/8216])

- If one used `CONVERT_TZ()` function in `SELECT`, which in its turn was used in `CREATE TABLE` statements, then system time zone tables were added to list of tables joined in SELECT and thus erroneous result was produced. (Bug #7899 [http://bugs.mysql.com/7899])
- Fixed a bug in `CONV()` function returning unsigned `BIGINT` number (third argument is positive, and return value does not fit in 32 bits). (Bug #7751 [http://bugs.mysql.com/7751])
- Fixed a failure of the `IN()` operator to return correct result if all values in the list were constants and some of them were using substring functions, for example, `LEFT()`, `RIGHT()`, or `MID()`.
- Fixed problem with `SHOW INDEX` reporting `Sub_part` values in bytes rather than characters for columns with a multi-byte character set. (Bug #7943 [http://bugs.mysql.com/7943])
- Fixed a crash in `CONVERT_TZ()` function when its second or third argument was from a `const` table (see Section 7.2.1, "`EXPLAIN` Syntax (Get Information About a `SELECT`)"). (Bug #7705 [http://bugs.mysql.com/7705])
- Correct a problem with **mysql_config**, which was failing to produce proper `zlib` option for linking under some circumstances. (Bug #6273 [http://bugs.mysql.com/6273])
- Fixed a problem with calculation of number of columns in row comparison against a subquery. (Bug #8020 [http://bugs.mysql.com/8020])
- Fixed erroneous output resulting from `SELECT DISTINCT` combined with a subquery and `GROUP BY`. (Bug #7946 [http://bugs.mysql.com/7946])
- Fixed server crash in comparing a nested row expression (for example `row(1,(2,3))`) with a subquery. (Bug #8022 [http://bugs.mysql.com/8022])
- Fixed server crash resulting from certain correlated subqueries with forward references (referring to an alias defined later in the outer query). (Bug #8025 [http://bugs.mysql.com/8025])
- Fixed server crash resulting from re-execution of prepared statements containing subqueries. (Bug #8125 [http://bugs.mysql.com/8125])
- Removed a dependence of boolean full-text search on `--default-character-set` option. (Bug #8159 [http://bugs.mysql.com/8159])
- Fixed a crash in a boolean full-text search in certain joins. (Bug #8234 [http://bugs.mysql.com/8234])
- Fixed erroneous comparison where strings that began with `CHAR(31)` were considered equal to the empty string. (Bug #8134 [http://bugs.mysql.com/8134])
- Add description of `debug` command to **mysqladmin** help output. (Bug #8207 [http://bugs.mysql.com/8207])
- **perror.exe** was always returning ``Unknown error'' on Windows. See Section 8.12, "**perror**, Explaining Error Codes". (Bug #7390 [http://bugs.mysql.com/7390])
- Modify `SET` statements produced by **mysqldump** to write quoted strings using single quotes rather than double quotes. This avoids problems if the dump file is reloaded while the `ANSI_QUOTES` SQL mode is in effect. (Bug #8148 [http://bugs.mysql.com/8148])
- Fixed a bug where `ALTER TABLE` improperly would accept an index on a `TIMESTAMP` column that `CREATE TABLE` would reject. (Bug #7884 [http://bugs.mysql.com/7884])
- Fixed a bug in multiple-table `UPDATE` statements that could cause spurious `Table '#sql_....' is full` errors if the number of rows to update is big enough. (Bug #7788

[http://bugs.mysql.com/7788])

- Fixed a problem where `SHOW INDEX` on a `MERGE` table could crash a debugging version of the server. (Bug #7377 [http://bugs.mysql.com/7377])
- Fixed a problem where adding an `ORDER BY` clause for an indexed column would cause a `SELECT` to return an empty result. (Bug #7331 [http://bugs.mysql.com/7331])
- Fixed a problem where `ALTER TABLE` on a `TEMPORARY` table with a mixed-lettercase name could cause the table to disappear when `lower_case_table_names` was set to 2. (Bug #7261 [http://bugs.mysql.com/7261])
- Fixed a problem with key cache statistics being reported incorrectly by the server after receipt of a `SIGHUP` signal. (Bug #4285 [http://bugs.mysql.com/4285])
- Fixed a problem that caused `mysql_stmt_prepare()` to be very slow when used in client programs on Windows. (Bug #5787 [http://bugs.mysql.com/5787])
- For indexes, `SHOW CREATE TABLE` now displays the index type even if it is the default, for storage engines that support multiple index types. (Bug #7235 [http://bugs.mysql.com/7235])
- Fixed a bug where the use of `GROUP_CONCAT()` with `HAVING` caused a server crash. (Bug #7769 [http://bugs.mysql.com/7769])
- Fixed a bug where comparing the result of a subquery to a non-existent column caused a server crash on Windows. (Bug #7885 [http://bugs.mysql.com/7885])
- Fixed a bug which caused `TIMEDIFF()` function to return wrong results if one of its arguments had non-zero microsecond part (Bug #7586 [http://bugs.mysql.com/7586]).
- Fixed a bug which caused `TIMESTAMP` columns with display width specified to be not identical to `DATETIME` columns when server was run in `MAXDB` mode (Bug #7418 [http://bugs.mysql.com/7418]).
- Fixed a bug in `UNION` statements that resulted in the wrong number of the examined rows reported in the slow query log.
- Fixed a bug in a combination of `-not` and `trunc*` operators of full-text search. Using more than one truncated negative search term, was causing empty result set.
- InnoDB: Fixed a bug introduced in 4.1.9 to the Windows version if you used `innodb_file_per_table`. **mysqld** would stop and complain about Windows error number 87 in a file operation. (See the Bugs database or the 4.1.9 change notes about a workaround for that bug in 4.1.9). (Bug #8021 [http://bugs.mysql.com/8021])
- InnoDB: Corrected the handling of trailing spaces in the `ucs2` character set. (Bug #7350 [http://bugs.mysql.com/7350])
- InnoDB: Use native `tmpfile()` function on Netware. All InnoDB temporary files are created under `sys:\tmp`. Previously, InnoDB temporary files were never deleted on Netware.
- InnoDB: Fix a race condition that could cause the assertion `space->n_pending_flushes == 0` to fail in `fil0fil.c`, in `fil_space_free()`, in `DROP TABLE` or in `ALTER TABLE`.
- InnoDB: `ALTER TABLE ... ADD CONSTRAINT PRIMARY KEY ...` complained about bad foreign key definition. (Bug #7831 [http://bugs.mysql.com/7831])
- InnoDB: Fix a theoretical hang over the adaptive hash latch in InnoDB if one runs `INSERT ... SELECT ...` (binlog not enabled), or a multi-table `UPDATE` or `DELETE`, and only the read tables

are InnoDB type, the rest are MyISAM. (Bug #7879 [http://bugs.mysql.com/7879])

- Fixed a bug in `max_heap_table_size` handling, that resulted in `Table is full` error when the table was still smaller than the limit. (Bug #7791 [http://bugs.mysql.com/7791]).
- Fixed a symlink vulnerability in the **mysqlaccess** script. Reported by Javier Fernandez-Sanguino Pena and Debian Security Audit Team [http://www.debian.org/security/audit]. (CAN-2005-0004 [http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2005-0004])
- **mysqlbinlog** forgot to add backquotes around the collation of user variables (causing later parsing problems as `BINARY` is a reserved word). (Bug #7793 [http://bugs.mysql.com/7793])
- Ensured that **mysqldump --single-transaction** sets its transaction isolation level to `REPEATABLE READ` before proceeding (otherwise if the MySQL server was configured to run with a default isolation level lower than `REPEATABLE READ` it could give an inconsistent dump). (Bug #7850 [http://bugs.mysql.com/7850])
- Changed **mysql** client so that including `\p` as part of a prompt command uses the name of the shared memory connection when the connection is using shared memory. (Bug #7922 [http://bugs.mysql.com/7922])
- Fixed a problem in the server where executing a multi-statement query more than once with the query cache active could yield incorrect result sets. (Bug #7966 [http://bugs.mysql.com/7966])
- Fixed that a 4.1.10 slave can connect to a master < 3.23.50 without hanging (the reason for the hang is a bug in these quite old masters -- `SELECT @@unknown_var` hangs them -- which was fixed in MySQL 3.23.50). (Bug #7965 [http://bugs.mysql.com/7965])
- Fixed a bug where MySQL was allowing concurrent updates (inserts, deletes) to a table if binary logging is enabled. Changed to ensure that all updates are executed in a serialized fashion, because they are executed serialized when binlog is replayed. (Bug #7879 [http://bugs.mysql.com/7879])

D.2.3. Changes in release 4.1.9 (11 Jan 2005)

Functionality added or changed:

- **mysqld_safe** no longer tests for the presence of the data directory when using a relatively-located server binary. It just assumes the directory is there, and fails to start up if it is not. This allows the data directory location to be specified on the command line, and avoids running a server binary that was not intended. (Bug #7249 [http://bugs.mysql.com/7249])
- The naming scheme of the Windows installation packages has changed slightly:
  - The platform suffix was changed from `-win` to `-win32`
  - The product descriptions `-noinstall` and `-essential` have been moved in front of the version number

  Examples: `mysql-essential-4.1.9-win32.msi`, `mysql-noinstall-4.1.9-win32.zip` See Section 2.3, “Installing MySQL on Windows”.
- The Mac OS X 10.3 installation disk images now include a MySQL Preference Pane for the Mac OS X Control Panel that enables the user to start and stop the MySQL server via the GUI and activate and deactivate the automatic MySQL server startup on bootup.
- The `MySQL-shared-compat` Linux RPM now includes the 3.23 as well as the 4.0 `libysql-`

client.so shared libraries. (Bug #6342 [http://bugs.mysql.com/6342])

- Seconds_Behind_Master is NULL (which means ``unknown'') if the slave SQL thread is not running, or if the slave I/O thread is not running or not connected to master. It is zero if the SQL thread has caught up with the I/O thread. It no longer grows indefinitely if the master is idle.
- InnoDB: Do not acquire an internal InnoDB table lock in LOCK TABLES if AUTOCOMMIT=1. This helps in porting old MyISAM applications to InnoDB. InnoDB table locks in that case caused deadlocks very easily.
- InnoDB: Print a more descriptive error and refuse to start InnoDB if the size of ibdata files is smaller than what is stored in the tablespace header; innodb_force_recovery overrides this.
- The MySQL server aborts immediately instead of simply issuing a warning if it is started with the --log-bin option but cannot initialize the binary log at startup (that is, an error occurs when writing to the binary log file or binary log index file).
- The binary log file and binary log index file now behave like MyISAM when there is a "disk full" or "quota exceeded" error. See Section A.4.3, “How MySQL Handles a Full Disk”.

Bugs fixed:

- Fixed problem where running **mysql_fix_privilege_tables** could result in grant table columns with too-short lengths if the server character set had been set to a multi-byte character set first. (Bug #7539 [http://bugs.mysql.com/7539])
- InnoDB: Fixed the **critical bug** if you enabled innodb_file_per_table in my.cnf. If you shut down **mysqld**, records could disappear from the secondary indexes of a table. Unfortunately, on Windows a new Bug #8021 [http://bugs.mysql.com/8021] was introduced. Windows users of innodb_file_per_table should put a line innodb_flush_method=unbuffered to their my.cnf or my.ini to work around 8021. (Bug #7496 [http://bugs.mysql.com/7496])
- InnoDB: Fixed a bug: 32-bit **mysqld** binaries built on HP-UX-11 did not work with InnoDB files greater than 2 GB in size. (Bug #6189 [http://bugs.mysql.com/6189])
- InnoDB: Return a sensible error code from DISCARD TABLESPACE if it fails because the table is referenced by a FOREIGN KEY.
- InnoDB: Fixed a bug: InnoDB failed to drop a table in the background drop queue if the table was referenced by a FOREIGN KEY constraint.
- InnoDB: Fixed a bug: if we dropped a table where an INSERT was waiting for a lock to check a FOREIGN KEY constraint, then an assertion would fail in lock_reset_all_on_table().
- InnoDB: Fix a little bug: we looked at the physical size of a stored SQL NULL value from a wrong field in the index; this has probably caused no bugs visible to the user. It caused only some extra space to be used in some rare cases.
- InnoDB: Use the fcntl() file flush method on Mac OS X versions 10.3 and up. Apple had disabled fsync() in Mac OS X for internal disk drives, which caused corruption at power outages.
- **mysqladmin password** now checks whether the server has --old-passwords turned on or predates 4.1 and uses the old-format password if so. (Bug #7451 [http://bugs.mysql.com/7451])
- Added a --default-character-set option to **mysqladmin** to avoid problems when the default character set is not latin1. (Bug #7524 [http://bugs.mysql.com/7524])

- Fix a problem with truncation of `FLOAT` values. (Bug #7361 [http://bugs.mysql.com/7361])
- Fixed a bug in `PROCEDURE ANALYSE()`, which did not quote some `ENUM` values properly. (Bug #2813 [http://bugs.mysql.com/2813])
- Fixed a bug that caused incorrect results for complex datetime expressions containing casts of datetime values to `TIME` or `DATE` values. (Bug #6914 [http://bugs.mysql.com/6914])
- Include compression library flags in the output from **mysql_config --lib_r**. (Bug #7021 [http://bugs.mysql.com/7021])
- Corrected a problem with **mysql_config** not producing all relevant flags from `CFLAGS`. (Bug #6964 [http://bugs.mysql.com/6964])
- Corrected a problem with **mysqld_safe** not properly capturing output from **ps**. (Bug #5878 [http://bugs.mysql.com/5878])
- Fixed a bug that caused a linking failure when linking both the MySQL client library and IMAP library. (Bug #7428 [http://bugs.mysql.com/7428])
- Fixed table corruption bug when using `INSERT DELAYED` with prepared statements.
- Fixed a bug that caused microseconds to be gobbled from the string result of the `STR_TO_DATE` function, if there is some other specifier in the format string following `%f`. (Bug #7458 [http://bugs.mysql.com/7458])
- Made the MySQL server accept executing `SHOW CREATE DATABASE` even if the connection has an open transaction or locked tables. Refusing it made **mysqldump --single-transaction** sometimes fail to print a complete `CREATE DATABASE` statement for some dumped databases. (Bug #7358 [http://bugs.mysql.com/7358])
- Fixed that, when encountering a ``disk full'' or ``quota exceeded'' write error, `MyISAM` sometimes didn't sleep and retry the write, thus resulting in a corrupted table. (Bug #7714 [http://bugs.mysql.com/7714])
- Fixed that `--expire-log-days` was not honored if using only transactions. (Bug #7236 [http://bugs.mysql.com/7236])
- Fixed that a slave could crash after replicating many `ANALYZE TABLE`, `OPTIMIZE TABLE`, or `REPAIR TABLE` statements from the master. (Bug #6461 [http://bugs.mysql.com/6461], Bug #7658 [http://bugs.mysql.com/7658])

D.2.4. Changes in release 4.1.8 (14 Dec 2004)

Note: Due to a **libtool**-related bug in the source distribution, the creation of shared `libmysqlclient` libraries was not possible (the resulting files were missing the `.so` file name extension). The file `ltmain.sh` was updated to fix this problem and the resulting source distribution was released as `mysql-4.1.8a.tar.gz`. This modification did not affect the binary packages. (Bug #7401 [http://bugs.mysql.com/7401])

Functionality added or changed:

- Automatic character set conversion formerly was done for operations that mix a column and a string such as assigning a string to a column, when this was possible without loss of information. Automatic conversion for operations that mix columns and strings has been expanded to cover many functions (such as `CONCAT()`) and assignment operators. This reduces the frequency of `Illegal mix`

of collations errors.

- For ALTER DATABASE, the database name now can be omitted to apply the change to the default database.
- Added WITH CONSISTENT SNAPSHOT clause to START TRANSACTION to begin a transaction with a consistent read.
- Added --order-by-primary to **mysqldump**, to sort each table's data in a dump file. This may be useful when dumping a MyISAM table which will be loaded into an InnoDB table. Dumping a MyISAM table with this option is considerably slower than without.
- InnoDB: Do not periodically write SHOW INNODB STATUS information to a temporary file unless the configuration option innodb_status_file=1 is set.
- InnoDB: Commit after every 10,000 copied rows when executing ALTER TABLE. This makes it much faster to recover from an aborted ALTER TABLE or OPTIMIZE TABLE.
- FULLTEXT index block size is changed to be 1024 instead of 2048.
- Added --disable-log-bin option to **mysqlbinlog**. Using this option you can disable binary logging for the statements produced by **mysqlbinlog**. That is, mysqlbinlog --disable-log-bin <file_name> | mysql won't write any statements to the MySQL server binary log.
- The --master-data option for **mysqldump** now takes an optional argument of 1 or 2 to produce a non-commented or commented CHANGE MASTER TO statement. The default is 1 for backward compatibility.
- mysqldump --single-transaction --master-data now is able to take an online (non-blocking) dump of InnoDB and report the corresponding binary log coordinates. This makes a backup suitable for point-in-time recovery, roll-forward or replication slave creation. See Section 8.8, “The **mysqldump** Database Backup Program”.
- Added --lock-all-tables to **mysqldump** to lock all tables by acquiring a global read lock.
- Added --hex-blob option to **mysqldump** for dumping binary string columns using hexadecimal notation.
- Added mysql_hex_string() C API function that hex-encodes a string.
- In the normal log MySQL now prints the log position for Binlog Dump requests.
- Added [mysql_cluster] section to my.cnf file for configuration settings specific to MySQL Cluster. ndb-connectstring variable moved here.
- A connection doing a rollback now displays "Rolling back" in the State column of SHOW PROCESSLIST.
- mysqlbinlog now prints an informative commented line (thread id, timestamp, server id, etc) before each LOAD DATA INFILE, like it does for other queries; unless --short-form is used.

Bugs fixed:

- Fixed incorrect referencing to column by name from subquery to outer query in case of using temporary table by outer query and placing subquery in the WHERE clause. (Bug #7079 [http://bugs.mysql.com/7079])

- Fixed a bug in authentication code that allowed a malicious user to crash the server with specially crafted packets (using a modified client library). (Bug #7187 [http://bugs.mysql.com/7187])
- Fixed a crashing bug in a string function `LEFT`, when this function is part of the expression which is used as `GROUP BY` field. (Bug #7101 [http://bugs.mysql.com/7101])
- Fixed bug which caused MySQL to require privileges on system time zone description tables for implicit access to them (i.e. if one set `time_zone` variable or used `CONVERT_TZ()` function) in case when some table-level or column-level privileges existed. (Bug #6765 [http://bugs.mysql.com/6765])
- `mysql_stmt_data_seek(stmt,0)` now rewinds a counter and enables buffered rows to be refetched on the client side. (Bug #6996 [http://bugs.mysql.com/6996])
- Fixed an insufficient privilege check in `SHOW CREATE TABLE` command. (Bug #7043 [http://bugs.mysql.com/7043])
- Fixed a rare memory corruption (that resulted in a crash) in `MATCH ... AGAINST` on columns that use multi-byte character sets. (Bug #6269 [http://bugs.mysql.com/6269])
- Fixed `NULL` processing in `ALL`/`SOME` subqueries. (Bug #6247 [http://bugs.mysql.com/6247])
- Fixed execution of complex queries with subqueries. (Bug #6406 [http://bugs.mysql.com/6406], Bug #6841 [http://bugs.mysql.com/6841])
- Fixed initialization of some internal structures for first execution. (Bug #6517 [http://bugs.mysql.com/6517])
- Backported a fix for the full-text interface from MySQL 5.0. (Bug #6523 [http://bugs.mysql.com/6523])
- Fixed `NULL` value handling in case of empty results in subqueries. (Bug #6806 [http://bugs.mysql.com/6806])
- Prevent adding `CREATE TABLE .. SELECT` query to the binary log when the insertion of new records partially failed. (Bug #6682 [http://bugs.mysql.com/6682])
- `INSERT ... SELECT` no longer reports spurious "column truncated" warnings (Bug #6284 [http://bugs.mysql.com/6284])
- Fixed a bug that could cause "Record has changed since last read in table" error message in some queries on `HEAP` tables that contain only one row. (Bug #6748 [http://bugs.mysql.com/6748])
- **mysqld_safe** was in many cases ignoring any `--no-defaults`, `--defaults-file`, or `--defaults-extra-file` arguments. Those arguments are now honored, and this may change what options are passed to **mysqld** in some installations.
- The server was interpreting `CHAR BINARY` and `VARCHAR BINARY` columns from 4.0 tables as having the `BINARY` and `VARBINARY` data types. Now they are interpreted as `CHAR` and `VARCHAR` columns that have the binary collation of the column's character set. (This is the same way that `CHAR BINARY` and `VARCHAR BINARY` are handled for new tables created in 4.1.)
- Fixed spurious "duplicate key" error from `REPLACE` or `INSERT ... ON DUPLICATE KEY UPDATE` statements performing multiple-row insert in the table that had unique and full-text indexes. (Bug #6784 [http://bugs.mysql.com/6784])
- Fixed a bug in execution of subqueries in `SET` and `DO` statements which caused wrong results to be returned from subsequent queries. (Bug #6462 [http://bugs.mysql.com/6462])

- Fixed a bug that allowed server to accept datetime values with wrong year part. The server now also performs same checks for datetime values passed through `MYSQL_TIME` structures as for datetime values passed as strings. (Bug #6266 [http://bugs.mysql.com/6266])

- Fixed a bug with `INSERT` for a table with `FULLTEXT` indexes. Under rare circumstances, this could result in a corrupted table if words of different lengths may be considered equal. This is possible in some collations, for example, in `utf8_general_ci` or `latin1_german2_ci`. (Bug #6265 [http://bugs.mysql.com/6265])

- InnoDB: Do not intentionally crash `mysqld` if the buffer pool is exhausted by the lock table; return error 1206 instead. Do not intentionally crash `mysqld` if we cannot allocate the memory for the InnoDB buffer pool. (Bug #6817 [http://bugs.mysql.com/6817]) (Bug #6827 [http://bugs.mysql.com/6827])

- InnoDB: Let InnoDB's `FOREIGN KEY` parser to remove the `latin1` character `0xA0` from the end of an unquoted identifier. The EMS MySQL Manager in `ALTER TABLE` adds that character after a table name, which caused error 121 when we tried to add a new constraint.

- InnoDB: Refuse to open new-style tables created with MySQL 5.0.3 or later. (Bug #7089 [http://bugs.mysql.com/7089])

- InnoDB: Do not call `rewind()` when displaying `SHOW INNODB STATUS` information on `stderr`.

- InnoDB: Made the foreign key parser better aware of quotes. (Bug #6340 [http://bugs.mysql.com/6340])

- InnoDB: If one used `INSERT IGNORE` to insert several rows at a time, and the first inserts were ignored because of a duplicate key collision, then InnoDB in a replication slave assigned `AUTO_INCREMENT` values 1 bigger than in the master. This broke the MySQL replication. (Bug #6287 [http://bugs.mysql.com/6287])

- InnoDB: Fixed a bug: InnoDB ignored in `innodb_data_file_path` the `max` specification in `:autoextend:max:2000M`. This bug was introduced in 4.1.1.

- InnoDB: Fixed a bug: `innodb_locks_unsafe_for_binlog` still uses next-key locking (Bug #6747 [http://bugs.mysql.com/6747]). InnoDB used next-key locking when record matched completely to search tuple. This unnecessary next-key locking is now removed when `innodb_locks_unsafe_for_binlog` option is used.

- InnoDB: Fix two hangs: `FOREIGN KEY` constraints treated table and database names as case-insensitive. `RENAME TABLE t TO T` would hang in an endless loop if `t` had a foreign key constraint defined on it. Fix also a hang over the dictionary mutex that would occur if one tried in `ALTER TABLE` or `RENAME TABLE` to create a foreign key constraint name that collided with another existing name. (Bug #3478 [http://bugs.mysql.com/3478])

- If `STMT_ATTR_UPDATE_MAX_LENGTH` is set for a prepared statement, `mysql_stmt_store_result()` updates `field->max_length` for numeric columns as well. (Bug#6096)

- Prepared statements now handle `ZEROFILL` when converting `integer` to `string`.

- Fixed crash when a call to `mysql_stmt_store_result()` occurred without a preceding call to `mysql_stmt_bind_result()`.

- Fixed crash in prepared statements when using `SELECT * FROM t1 NATURAL JOIN t2....`

- Fixed crash in prepared statements when using `SELECT ... PROCEDURE`.

- Fixed crash in prepared statements when using subqueries.
- `GROUP_CONCAT(...ORDER BY)` when used with prepared statements gave wrong sorting order.
- `CREATE TABLE created_table` didn't signal when table was created. This could cause a `DROP TABLE created_table` in another thread to wait "forever".
- Server warnings now are reset when you execute a prepared statement.
- Improved performance of identifier comparisons (if many tables or columns are specified).
- `OPTIMIZE TABLE`, `REPAIR TABLE`, and `ANALYZE TABLE` are now replicated without any error code in the binary log. (Bug #5551 [http://bugs.mysql.com/5551])
- `LOAD DATA INFILE` now works with option replicate-rewrite-db. (Bug #6353 [http://bugs.mysql.com/6353])
- Fixed a bug which caused a crash when only the slave I/O thread was stopped and started. (Bug #6148 [http://bugs.mysql.com/6148])
- Changed semantics of `CREATE/ALTER/DROP DATABASE` statements so that replication of `CREATE DATABASE` is possible when using `--binlog-do-db` and `--binlog-ignore-db`. (Bug #6391 [http://bugs.mysql.com/6391])
- If a connection had an open transaction but had done no updates to transactional tables (for example if had just done a `SELECT FOR UPDATE` then executed a non-transactional update, that update automatically committed the transaction (thus releasing InnoDB's row-level locks etc). (Bug #5714 [http://bugs.mysql.com/5714])
- If a connection was interrupted by a network error and did a rollback, the network error code got stored into the `BEGIN` and `ROLLBACK` binary log events; that caused superfluous slave stops. (Bug #6522 [http://bugs.mysql.com/6522])
- A sequence of `BEGIN` (or `SET AUTOCOMMIT=0`), `FLUSH TABLES WITH READ LOCK`, transactional update, `COMMIT`, `FLUSH TABLES WITH READ LOCK` could hang the connection forever and possibly the MySQL server itself. This happened for example when running the `innobackup` script several times. (Bug #6732 [http://bugs.mysql.com/6732])
- **mysqlbinlog** did not print `SET PSEUDO_THREAD_ID` statements in front of `LOAD DATA INFILE` statements inserting into temporary tables, thus causing potential problems when rolling forward these statements after restoring a backup. (Bug #6671 [http://bugs.mysql.com/6671])

D.2.5. Changes in release 4.1.7 (23 Oct 2004: Production)

Functionality added or changed:

- `MOD()` no longer rounds arguments with a fractional part to integers. Now it returns exact remainder after division. (Bug #6138 [http://bugs.mysql.com/6138])
- InnoDB: Added a startup option and settable system variable `innodb_table_locks` for making `LOCK TABLE` acquire also `InnoDB` locks. The default value is 1, which means that `LOCK TABLES` causes also InnoDB internally to take a table lock. In applications using `AUTOCOMMIT=1` and `LOCK TABLES`, InnoDB's internal table locks that were added in 4.0.20 and 4.1.2 can cause deadlocks. You can set `innodb_table_locks=0` in `my.cnf` to remove that problem. (Bug #3299 [http://bugs.mysql.com/3299], Bug #5998 [http://bugs.mysql.com/5998])

- See Section 15.17, “Restrictions on `InnoDB` Tables”. (Bug #3299 [http://bugs.mysql.com/3299], Bug #5998 [http://bugs.mysql.com/5998]) InnoDB: `SHOW TABLE STATUS` now shows the creation time of the table for InnoDB. Note that this timestamp might not be the correct time because, e.g., `ALTER TABLE` changes this timestamp.
- InnoDB: If `innodb_thread_concurrency` would be exceeded, let a thread sleep 10 ms before entering the FIFO queue; previously, the value was 50 ms.

Bugs fixed:

- Fixed a bug with `FOUND_ROWS()` used together with `LIMIT` clause in prepared statements. (Bug#6088)
- Fixed a bug with `NATURAL JOIN` in prepared statements. (Bug#6046).
- Fixed a bug in join of tables from different databases having columns with identical names (prepared statements). (Bug#6050)
- Now implicit access to system time zone description tables (which happens when you set `time_zone` variable or use `CONVERT_TZ()` function) does not require any privileges. (Bug #6116 [http://bugs.mysql.com/6116])
- Fixed a bug which caused the server to crash when the deprecated `libmysqlclient` function `mysql_create_db()` was called. (Bug #6081 [http://bugs.mysql.com/6081])
- Fixed `REVOKE ALL PRIVILEGES, GRANT OPTION FROM` *`user`* so that all privileges are revoked correctly. (Bug #5831 [http://bugs.mysql.com/5831]). This corrects a case that the fix in 4.1.6 could miss.
- Fixed crash when selecting from a `HEAP` table with `key_column IS NOT NULL`. This could also cause a crash if not all index parts where used. (Bug #6082 [http://bugs.mysql.com/6082])
- Fixed a bug that could cause `MyISAM` index corruption when key values start with character codes below `BLANK`. This was caused by the new key sort order in 4.1. (Bug #6151 [http://bugs.mysql.com/6151])
- InnoDB: Fixed a bug in `LOAD DATA INFILE…REPLACE` printing duplicate key error when executing the same load query several times. (Bug #5835 [http://bugs.mysql.com/5835])
- Fixed a bug in the prepared statements protocol when wrong metadata was sent for `SELECT` statements not returning a result set (such as `SELECT ... INTO OUTFILE`). (Bug #6059 [http://bugs.mysql.com/6059])
- Fixed bug which allowed one to circumvent missing UPDATE privilege if one had INSERT and SELECT privileges for table with primary key. (Bug #6173 [http://bugs.mysql.com/6173])
- Fixed a bug in `libmysqlclient` with wrong conversion of negative time values to strings. (Bug #6049 [http://bugs.mysql.com/6049]).
- Fixed a bug in `libmysqlclient` with wrong conversion of zero date values (`0000-00-00`) to strings. (Bug #6058 [http://bugs.mysql.com/6058])
- Fixed a bug that caused the server to crash on attempt to prepare a statement with `RAND(?)`. (Bug #5985 [http://bugs.mysql.com/5985])
- Fixed a bug with handling of `DATE`, `TIME`, and `DATETIME` columns in the binary protocol. The problem is compiler-specific and could have been observed on HP-UX, AIX, Solaris9, when compil-

ing with native compiler. (Bug #6025 [http://bugs.mysql.com/6025])

- Fixed a bug with handling of `TINYINT` columns in the binary protocol. The problem is specific to platforms where the C compiler has the `char` data type unsigned by default. (Bug #6024 [http://bugs.mysql.com/6024])

- InnoDB: Fixed problem introduced in MySQL 4.0.21 where a connection starting a transaction, doing updates, then `FLUSH TABLES WITH READ LOCK`, then `COMMIT`, would cause replication slaves to stop (complaining about error 1223). Bug surfaced when using the InnoDB `innobackup` script. (Bug #5949 [http://bugs.mysql.com/5949])

- InnoDB: Release the dictionary latch during a long cascaded `FOREIGN KEY` operation, so that we do not starve other users doing `CREATE TABLE` or other DDL operation. This caused a notorious 'Long semaphore wait' message to be printed to the `.err` log. (Bug #5961 [http://bugs.mysql.com/5961])

D.2.6. Changes in release 4.1.6 (10 Oct 2004)

Functionality added or changed:

- Added option `--sigint-ignore` to the `mysql` command line client to make it ignore `SIGINT` signals (typically the result of the user pressing Control-C).

- InnoDB: Added the startup option and settable global variable `innodb_max_purge_lag` for delaying `INSERT`, `UPDATE` and `DELETE` operations when the purge operations are lagging. The default value of this parameter is zero, meaning that there are no delays. See Section 15.13, “Implementation of Multi-Versioning”.

- InnoDB: The `innodb_autoextend_increment` startup option that was introduced in release 4.1.5 was made a settable global variable. (Bug #5736 [http://bugs.mysql.com/5736])

- InnoDB: If `DROP TABLE` is invoked on an InnoDB table for which the `.ibd` file is missing, print to error log that the table was removed from the `InnoDB` data dictionary, and allow MySQL to delete the `.frm` file. Maybe `DROP TABLE` should issue a warning in this case.

- `TIMESTAMP` columns now can store `NULL` values. To create such a column, you must explicitly specify the `NULL` attribute in the column specification. (Unlike all other column types, `TIMESTAMP` columns are `NOT NULL` by default.)

- Now if `ALTER TABLE` converts one `AUTO_INCREMENT` column to another `AUTO_INCREMENT` column it preserves zero values (this includes the case that we don't change such column at all).

- Now if `ALTER TABLE` converts some column to `TIMESTAMP NOT NULL` column it converts `NULL` values to current timestamp value (One can still get old behavior by setting system `TIMESTAMP` variable to zero).

- On Windows, the MySQL configuration files included in the package now use `.ini` instead of `.cnf` as the file name suffix.

Bugs fixed:

- Fixed a bug that caused the server to crash on attempt to execute a prepared statement with a subselect inside a boolean expression. (Bug #5987 [http://bugs.mysql.com/5987])

- Fixed a bug that caused the server to sometimes choose non-optimal execution plan for a prepared statement executed with changed placeholder values. (Bug #6042 [http://bugs.mysql.com/6042])
- InnoDB: Make the check for excessive semaphore waits tolerate glitches in the system clock (do not crash the server if the system time is adjusted while InnoDB is under load.). (Bug #5898 [http://bugs.mysql.com/5898])
- InnoDB: Fixed a bug in the InnoDB `FOREIGN KEY` parser that prevented `ALTER TABLE` of tables containing '#' in their names. (Bug #5856 [http://bugs.mysql.com/5856])
- InnoDB: Fixed a bug that prevented `ALTER TABLE t DISCARD TABLESPACE` from working. (Bug #5851 [http://bugs.mysql.com/5851])
- InnoDB: `SHOW CREATE TABLE` now obeys the `SET SQL_MODE=ANSI` and `SET SQL_QUOTE_SHOW_CREATE=0` settings. (Bug #5292 [http://bugs.mysql.com/5292])
- InnoDB: Fixed a bug that caused `CREATE TEMPORARY TABLE ... ENGINE=InnoDB` to terminate **mysqld** when running in `innodb_file_per_table` mode. Per-table tablespaces for temporary tables from now on are created in the temporary directory of **mysqld**. (Bug #5137 [http://bugs.mysql.com/5137])
- InnoDB: Fixed some (not all) UTF-8 bugs in column prefix indexes. (Bug #5975 [http://bugs.mysql.com/5975])
- InnoDB: If one updated a column so that its size changed, or updated it to an externally stored (`TEXT` or `BLOB`) value, then ANOTHER externally stored column would show up as 512 bytes of good data + 20 bytes of garbage in a consistent read that fetched the old version of the row. (Bug #5960 [http://bugs.mysql.com/5960])
- InnoDB: Change error code to `HA_ERR_ROW_IS_REFERENCED` if we cannot `DROP` a parent table referenced by a `FOREIGN KEY` constraint; this error number is less misleading than the previous number `HA_ERR_CANNOT_ADD_FOREIGN`, but misleading still. (Bug #6202 [http://bugs.mysql.com/6202])
- Fixed `REVOKE ALL PRIVILEGES, GRANT OPTION FROM user` so that all privileges are revoked correctly. (Bug #5831 [http://bugs.mysql.com/5831])
- Fixed a bug that caused the server to crash when character set conversion was implicitly used in prepared mode; for example, as in `'abc' LIKE CONVERT('abc' as utf8)`. (Bug #5688 [http://bugs.mysql.com/5688])
- The `mysql_change_user()` C API function now frees all prepared statements associated with the connection. (Bug #5315 [http://bugs.mysql.com/5315])
- Fixed a bug when inserting `NULL` into an `AUTO_INCREMENT` column failed, when using prepared statements. (Bug #5510 [http://bugs.mysql.com/5510])
- Fixed slave SQL thread so that the `SET COLLATION_SERVER...` statements it replicates don't advance its position (so that if it gets interrupted before the actual update query, it later redoes the `SET`). (Bug #5705 [http://bugs.mysql.com/5705])
- Fixed that if the slave SQL thread found a syntax error in a query (which should be rare, as the master parsed it successfully), it stops. (Bug #5711 [http://bugs.mysql.com/5711])
- Fixed that if a write to a `MyISAM` table fails because of a full disk or an exceeded disk quota, it prints a message to the error log every 10 minutes, and waits until disk space becomes available. (Bug #3248 [http://bugs.mysql.com/3248])
- Now MySQL does not prefer columns, which are mentioned in select list but are renamed, over

columns from other tables participating in FROM clause when it resolves GROUP BY clause (e.g. SELECT t1.a AS c FROM t1, t2 ORDER BY a produces an error if both t1 and t2 tables contain a column). (Bug #4302 [http://bugs.mysql.com/4302])

- Behavior of ALTER TABLE converting column containing NULL values to AUTO_INCREMENT column is no longer affected by NO_AUTO_VALUE_ON_ZERO mode. (Bug #5915 [http://bugs.mysql.com/5915]).

D.2.7. Changes in release 4.1.5 (16 Sep 2004)

Functionality added or changed:

- InnoDB: Added configuration option innodb_autoextend_increment for setting the size in megabytes by which InnoDB tablespaces are extended when they become full. The default value is 8, corresponding to the fixed increment of 8MB in previous versions of MySQL.
- InnoDB: The new Windows installation wizard of MySQL makes InnoDB as the MySQL default table type on Windows, unless explicitly specified otherwise. Note that it places the my.ini file in the installation directory of the MySQL server. See Section 2.3.5.14, “The Location of the my.ini File”.

Bugs fixed:

- Fixed a bug which caused the server to crash on attempt to execute a prepared statement with BETWEEN ? AND ? and a datetime column. (Bug #5748 [http://bugs.mysql.com/5748])
- Fixed name resolving of external fields of subqueries if subquery placed in select list of query with grouping. (Bug #5326 [http://bugs.mysql.com/5326])
- Fixed detection of using same table for updating and selecting in multi-update queries. (Bug #5455 [http://bugs.mysql.com/5455])
- The values of the max_sort_length, sql_mode, and group_concat_max_len system variables now are stored in the query cache with other query information to avoid returning an incorrect result from the query cache. (Bug #5394 [http://bugs.mysql.com/5394]) (Bug #5515 [http://bugs.mysql.com/5515])
- Fixed syntax analyzer with sql_mode=IGNORE_SPACE. It happened to take phrases like default .07 as identifier.identifier. (Bug #5318 [http://bugs.mysql.com/5318])
- Fixed illegal internal field length of user variables of integer type. This showed up when creating a table as SELECT @*var_name*. (Bug #4788 [http://bugs.mysql.com/4788])
- Fixed a buffer overflow in prepared statements API (libmysqlclient) when a statement containing thousands of placeholders was executed. (Bug #5194 [http://bugs.mysql.com/5194])
- Fixed a bug in the server when after reaching a certain limit of prepared statements per connection (97), statement ids began to overlap, so occasionally wrong statements were chosen for execution. (Bug #5399 [http://bugs.mysql.com/5399])
- Fixed a bug in prepared statements when LIKE used with arguments in different character sets crashed server on first execute. (Bug #4368 [http://bugs.mysql.com/4368])
- Fixed a bug in prepared statements when providing '0000-00-00' date to a parameter lead to server

crash. (Bug #4231 [http://bugs.mysql.com/4231], Bug #4562 [http://bugs.mysql.com/4562])

- Fixed a bug in `OPTIMIZE TABLE` that could cause table corruption on `FULLTEXT` indexes. (Bug #5327 [http://bugs.mysql.com/5327])

- InnoDB: Fixed a bug that InnoDB only allowed a maximum of 1000 connections inside InnoDB at the same time. A higher number could cause an assertion failure in sync0arr.c, line 384. Now we allow 1000, 10000, or 50000, depending on the buffer pool size. (Bug #5414 [http://bugs.mysql.com/5414])

D.2.8. Changes in release 4.1.4 (26 Aug 2004: Gamma)

Note: To fix a compile problem on systems that do not have `automake` 1.7 installed, an updated 4.1.4a source tarball has been published. In addition to resolving this `automake` dependency (Bug #5319), it also fixes some reported `libedit` compile errors when using a non-`gcc` compiler (Bug #5353 [http://bugs.mysql.com/5353]).

Functionality added or changed:

- Added the `CSV` storage engine.

- Made internal representation of `TIMESTAMP` values in `InnoDB` in 4.1 to be the same as in 4.0. This difference resulted in incorrect datetime values in `TIMESTAMP` columns in `InnoDB` tables after an upgrade from 4.0 to 4.1. (Bug #4492 [http://bugs.mysql.com/4492]) **Warning: extra steps during upgrade required!** Unfortunately this means that if you are upgrading from 4.1.x, where x <= 3, to 4.1.4 you should use **mysqldump** for saving and then restoring your `InnoDB` tables with `TIMESTAMP` columns.

- The **mysqld-opt** Windows server was renamed to **mysqld**. This completes the Windows server renaming begun in MySQL 4.1.2. See Section 2.3.9, “Selecting a MySQL Server type”.

- Added Latin language collations for the `ucs2` and `utf8` Unicode character sets. These are called `ucs2_roman_ci` and `utf8_roman_ci`.

- Corrected the name of the Mac OS X StartupItem script (it must match the name of the subdirectory, which was renamed to `MySQLCOM` in MySQL 4.1.2). Thanks to Bryan McCormack for reporting this.

- Added `--start-datetime`, `--stop-datetime`, `--start-position`, and `--stop-position` options to `mysqlbinlog`. These make point-in-time recovery easier.

- Killing a `CHECK TABLE` statement does not result in the table being marked as ``corrupted'' any more; the table remains as if `CHECK TABLE` had not even started. See Section 13.5.5.3, “`KILL` Syntax”.

- Made the MySQL server ignore `SIGHUP` and `SIGQUIT` on Mac OS X 10.3. This is needed because under this OS, the MySQL server receives lots of these signals (reported as Bug #2030 [http://bugs.mysql.com/2030]).

- Support of usage of column aliases qualified by table name or alias in `ORDER BY` and `GROUP BY` was dropped. For example the following query `SELECT a AS b FROM t1 ORDER BY t1.b` is not allowed. One should use `SELECT a AS b FROM t1 ORDER BY t1.a` or `SELECT a AS b FROM t1 ORDER BY b` instead. This was non-standard (since aliases are defined on query level not on table level) and caused problems with some queries.

Bugs fixed:

- Fixed a bug that caused libmysql to crash when attempting to fetch a value of `MEDIUMINT` column. (Bug #5126 [http://bugs.mysql.com/5126])
- Fixed a bug that caused the MySQL server to crash when attempting to execute a prepared statement with `SELECT ... INTO @var` for a second time. (Bug #5034 [http://bugs.mysql.com/5034])
- Fixed execution of optimized `IN` subqueries that use compound indexes. (Bug #4435 [http://bugs.mysql.com/4435])
- Prohibited resolving of table fields in inner queries if fields do not take part in grouping for queries with grouping (inside aggregate function arguments, all table fields are still allowed). (Bug #4814 [http://bugs.mysql.com/4814])
- Fixed a crash after `SLAVE STOP` if the IO thread was in a special state. (Bug #4629 [http://bugs.mysql.com/4629])
- Fixed an old bug in concurrent accesses to `MERGE` tables (even one `MERGE` table and `MyISAM` tables), that could have resulted in a crash or hang of the server. (Bug #2408 [http://bugs.mysql.com/2408])
- Fixed a bug that caused server crash on attempt to execute for a second time a prepared statement with `NOT` in `WHERE` or `ON` clauses. (Bug #4912 [http://bugs.mysql.com/4912])
- `MATCH ... AGAINST` now works in a subquery. (Bug #4769 [http://bugs.mysql.com/4769])
- Fixed a bug that omitted the `.err` extension of the error log file (`--log-error`) when the hostname contained a domain name. The domain name is now replaced by the extension. (Bug #4997 [http://bugs.mysql.com/4997])
- Fixed a crash in **myisamchk**. (Bug #4901 [http://bugs.mysql.com/4901])
- Fixed a bug which caused server crash if one used the `CONVERT_TZ()` function with time zone described in database as parameter and this time zone was not used before. (Bug #4508 [http://bugs.mysql.com/4508])
- Support for `%T, %r, %V, %v` and `%X, %x` format specifiers was added to `STR_TO_DATE()` function. (Bug #4756 [http://bugs.mysql.com/4756])
- Fixed a bug (hang) in `NATURAL JOIN` where joined table had no common column. (Bug #4807 [http://bugs.mysql.com/4807])
- Fixed a crash caused by `UNHEX(NULL)`. (Bug #4441 [http://bugs.mysql.com/4441])
- **mysql_fix_privilege_tables** didn't correctly handle the argument of its `--password=#` option. (Bug #4240 [http://bugs.mysql.com/4240], Bug #4543 [http://bugs.mysql.com/4543])
- Fixed that **mysqlbinlog --read-from-remote-server** sometimes couldn't accept 2 binary logs on command line. (Bug #4507 [http://bugs.mysql.com/4507])
- Fixed that **mysqlbinlog --position --read-from-remote-server** had wrong `# at` lines. (Bug #4506 [http://bugs.mysql.com/4506])
- If `CREATE TEMPORARY TABLE t SELECT` failed while loading the data, the temporary table was not dropped. (Bug #4551 [http://bugs.mysql.com/4551])
- Fixed that when a multiple-table `DROP TABLE` failed to drop a table on the master server, the error

code was not written to the binary log. (Bug #4553 [http://bugs.mysql.com/4553])

- When the slave SQL thread was replicating a `LOAD DATA INFILE` statement, it didn't show the statement in the output of `SHOW PROCESSLIST`. (Bug #4326 [http://bugs.mysql.com/4326])
- Fixed an assertion failure when reading the grant tables (Bug #4407 [http://bugs.mysql.com/4407])
- Fixed that `CREATE TABLE ... TYPE=HEAP ... AS SELECT...` caused replication slave to stop. (Bug #4971 [http://bugs.mysql.com/4971])
- Fixed that `mysql_options(...,MYSQL_OPT_LOCAL_INFILE,...)` failed to disable `LOAD DATA LOCAL INFILE`. (Bug #5038 [http://bugs.mysql.com/5038])
- Fixed that `disable-local-infile` option had no effect if client read it from a configuration file using `mysql_options(...,MYSQL_READ_DEFAULT,...)`. (Bug #5073 [http://bugs.mysql.com/5073])
- Fixed that `SET GLOBAL SYNC_BINLOG` did not work on some platforms (Mac OS X). (Bug #5064 [http://bugs.mysql.com/5064])
- Fixed that **mysql-test-run** failed on the `rpl_trunc_binlog` test if running test from the installed (the target of 'make install') directory. (Bug #5050 [http://bugs.mysql.com/5050])
- Fixed that **mysql-test-run** failed on the `grant_cache` test when run as Unix user 'root'. (Bug #4678 [http://bugs.mysql.com/4678])
- Fixed an unlikely deadlock which could happen when using `KILL`. (Bug #4810 [http://bugs.mysql.com/4810])
- Fixed a crash when one connection got `KILL`ed while it was doing `START SLAVE`. (Bug #4827 [http://bugs.mysql.com/4827])
- Made `FLUSH TABLES WITH READ LOCK` block `COMMIT` if server is running with binary logging; this ensures that the binary log position is trustable when doing a full backup of tables and the binary log. (Bug #4953 [http://bugs.mysql.com/4953])
- Fixed that the counter of an `auto_increment` column was not reset by `TRUNCATE TABLE` if the table was a temporary table. (Bug #5033 [http://bugs.mysql.com/5033])
- Fixed bug which caused error to be reported when column from `ORDER BY` clause was present in two tables participating in `SELECT` even if the second instance of column in select list was renamed. (Bug #4302 [http://bugs.mysql.com/4302])

D.2.9. Changes in release 4.1.3 (28 Jun 2004: Beta)

Note: The initial release of MySQL 4.1.3 for Windows accidentally was not compiled with support for the Spatial Extensions (OpenGIS). This was fixed by rebuilding from the same 4.1 code snapshot with the missing option and releasing those packages as version 4.1.3a.

To enable compiling the newly released PHP 5 against MySQL 4.1.3 on Windows, the Windows packages had to be rebuilt once more to add a few missing symbols to the MySQL client library. These packages were released as MySQL 4.1.3b.

Functionality added or changed:

- Added the `ARCHIVE` storage engine.

- Added SQL syntax for prepared statements. See Section 13.7, “SQL Syntax for Prepared Statements”.
- Language-specific collations were added for the `ucs2` and `utf8` Unicode character sets: Icelandic, Latvian, Romanian, Slovenian, Polish, Estonian, Swedish, Turkish, Czech, Danish, Lithuanian, Slovak, Spanish, Traditional Spanish.
- Support for per-connection time zones was added. Now you can set the current time zone for a connection by setting the `@@time_zone` system variable to a value such as `'+10:00'` or `'Europe/Moscow'` (where `'Europe/Moscow'` is the name of one of the time zones described in the system tables). Functions like `CURRENT_TIMESTAMP`, `UNIX_TIMESTAMP`, and so forth honor this time zone. Values of `TIMESTAMP` type are also interpreted as values in this time zone. So now our `TIMESTAMP` type behaves similar to Oracle's `TIMESTAMP WITH LOCAL TIME ZONE`. That is, values stored in such a column are normalized toward UTC and converted back to the current connection time zone when they are retrieved from such a column. To set up the tables that store time zone information, see Section 2.9, “Post-Installation Setup and Testing”.
- **Warning: Incompatible change!** The `timezone` system variable has been removed and replaced by `system_time_zone`. See Section 5.2.3, “Server System Variables”.
- Basic time zone conversion function `CONVERT_TZ()` was added. It assumes that its first argument is a datetime value in the time zone specified by its second argument and returns the equivalent datetime value in the time zone specified by its third argument.
- `CHECK TABLE` now can be killed. It then marks the table as corrupted. See Section 13.5.5.3, “`KILL` Syntax”.
- **Warning: Incompatible change!** C API change: `mysql_shutdown()` now requires a second argument. This is a source-level incompatibility that affects how you compile client programs; it does not affect the ability of compiled clients to communicate with older servers. See Section 22.2.3.54, “`mysql_shutdown()`”.
- `OPTIMIZE TABLE` for `InnoDB` tables is now mapped to `ALTER TABLE` instead of `ANALYZE TABLE`.
- `sync_frm` is now a settable global variable (not only a startup option).
- Added the `sync_binlog=N` global variable and startup option, which makes the MySQL server synchronize its binary log to disk (`fdatasync()`) after every *N*th write to the binary log.
- Changed the slave SQL thread to print fewer useless error messages (no more message duplication; no more messages when an error is skipped (because of `slave-skip-errors`).
- `DROP DATABASE IF EXISTS`, `DROP TABLE IF EXISTS`, single-table `DELETE` and single-table `UPDATE` are now written to the binary log even if they changed nothing on the master (for example, even if the `DELETE` matched no row). The old behavior sometimes caused bad surprises in replication setups.
- Replication and **mysqlbinlog** now have better support for the case that the session character set and collation variables are changed within a given session. See Section 6.7, “Replication Features and Known Problems”.
- Added `--innodb-safe-binlog` server option, which adds consistency guarantees between the content of `InnoDB` tables and the binary log. See Section 5.9.4, “The Binary Log”.
- `LIKE` now supports the use of a prepared statement parameter or delimited constant expression as the argument to `ESCAPE` (Bug #4200 [http://bugs.mysql.com/4200]).

Bugs fixed:

- Fixed `CREATE DATABASE IF NOT EXISTS` for Win32 which caused an error if database existed. (Bug #4378 [http://bugs.mysql.com/4378])
- Added missing `root` account to Windows version of **mysqld**. (Bug #4242 [http://bugs.mysql.com/4242])
- Fixed bug in prepared `EXPLAIN` statement which led to server crash. (Bug #4271 [http://bugs.mysql.com/4271])
- Fixed a bug of using parameters in some prepared statements via SQL syntax. (Bug #4280 [http://bugs.mysql.com/4280])
- Fixed a bug in `MERGE` tables created with `INSERT_METHOD=LAST`, that were not able to report a key number that caused ``Duplicate entry'' error for `UNIQUE` key in `INSERT`. As a result, error message was not precise enough (error 1022 instead of error 1062) and `INSERT ... ON DUPLICATE KEY UPDATE` did not work. (Bug #4008 [http://bugs.mysql.com/4008])
- Fixed a bug in `DELETE` from a table with `FULLTEXT` indexes which under rare circumstances could result in a corrupted table, if words of different lengths may be considered equal (which is possible in some collations, for example, in `utf8_general_ci` or `latin1_german2_ci`.) (Bug #3808 [http://bugs.mysql.com/3808])
- Fixed too-early unlocking of tables if we have subquery in `HAVING` clause. (Bug #3984 [http://bugs.mysql.com/3984])
- Fixed a bug in **mysqldump** when it didn't return an error if the output device was filled (Bug #1851 [http://bugs.mysql.com/1851])
- Fixed a bug in client-side conversion of string column to `MYSQL_TIME` application buffer (prepared statements API). (Bug #4030 [http://bugs.mysql.com/4030])
- Fixed a bug with server crash on attempt to execute a non-prepared statement. (Bug #4236 [http://bugs.mysql.com/4236])
- Fixed a bug with server crash on attempt to prepare a statement with character set introducer. (Bug #4105 [http://bugs.mysql.com/4105])
- Fixed bug which caused different number of warnings to be generated when bad datetime as string or as number was inserted into `DATETIME` or `TIMESTAMP` column. (Bug #2336 [http://bugs.mysql.com/2336])
- Fixed some byte order bugs with prepared statements on machines with high-byte-first. (Bug #4173 [http://bugs.mysql.com/4173])
- Fixed unlikely bug in the range optimizer when using many `IN()` queries on different key parts. (Bug #4157 [http://bugs.mysql.com/4157])
- Fixed problem with `NULL` and derived tables. (Bug #4097 [http://bugs.mysql.com/4097])
- Fixed wrong `UNION` results if display length of fields for numeric types was set less then real length of values in them. (Bug #4067 [http://bugs.mysql.com/4067])
- Fixed a bug in `mysql_stmt_close()`, which hung up when attempting to close statement after failed `mysql_stmt_fetch()`. (Bug #4079 [http://bugs.mysql.com/4079])
- Fixed bug of re-execution optimized `COUNT(*)`, `MAX()` and `MIN()` functions in prepared state-

ments. (Bug #2687 [http://bugs.mysql.com/2687])

- Fixed a bug with `COUNT(DISTINCT)` performance degradation in cases like `COUNT(DISTINCT a TEXT, b CHAR(1))` (no index used). (Bug #3904 [http://bugs.mysql.com/3904])
- Fixed a bug in `MATCH ... AGAINST(... IN BOOLEAN MODE)` that under rare circumstances could cause wrong results if in the data's collation one byte could match many (like in `utf8_general_ci` or `latin1_german2_ci`.) (Bug #3964 [http://bugs.mysql.com/3964])
- Fixed a bug in prepared statements protocol, when microseconds part of `MYSQL_TYPE_TIME`/`MYSQL_TYPE_DATETIME` columns was not sent to the client. (Bug #4026 [http://bugs.mysql.com/4026])
- Fixed a bug that using `--with-charset` with **configure** didn't affect the MySQL client library. (Bug #3990 [http://bugs.mysql.com/3990])
- Fixed a bug in authentication code that allowed a malicious user to bypass password verification with specially crafted packets (using a modified client library).
- Fixed bug with wrong result of `CONCAT(?, col_name)` in prepared statements. (Bug #3796 [http://bugs.mysql.com/3796])
- Fixed potential memory overrun in `mysql_real_connect()` (which required a compromised DNS server and certain operating systems). (Bug #4017 [http://bugs.mysql.com/4017])
- During the installation process of the server RPM on Linux, **mysqld** was run as the `root` system user, and if you had `--log-bin=file_name`, where the file was located somewhere outside of the data directory, it created binary log files owned by `root` in this directory that remained owned by `root` after the installation. This is now fixed by starting **mysqld** as the `mysql` system user instead. (Bug #4038 [http://bugs.mysql.com/4038])
- Made `DROP DATABASE` honor the value of `lower_case_table_names`. (Bug #4066 [http://bugs.mysql.com/4066])
- The slave SQL thread refused to replicate `INSERT ... SELECT` if it examined more than 4 billion rows. (Bug #3871 [http://bugs.mysql.com/3871])
- **mysqlbinlog** didn't escape the string content of user variables, and did not deal well when these variables were in non-ASCII character sets; this is now fixed by always printing the string content of user variables in hexadecimal. The character set and collation of the string is now also printed. (Bug #3875 [http://bugs.mysql.com/3875])
- Fixed incorrect destruction of expression which led to crash of server on complex `AND`/`OR` expressions if query was ignored (either by a replication server because of `--replicate-*-table` rules, or by any MySQL server because of a syntax error). (Bug #3969 [http://bugs.mysql.com/3969], Bug #4494 [http://bugs.mysql.com/4494])

D.2.10. Changes in release 4.1.2 (28 May 2004)

Functionality added or changed:

- Added the `EXAMPLE` storage engine.
- The **mysqld** Windows server was renamed to **mysqld-debug**. See Section 2.3.9, “Selecting a MySQL Server type”.

- Added `Handler_discover` status variable.
- Added support for character set conversion and `MYSQL_TYPE_BLOB` type code in prepared statement protocol.
- Added explanation of hidden `SELECT` of `UNION` in output of `EXPLAIN SELECT` statement.
- **mysql** command-line client now supports multiple `-e` options. (Bug #591 [http://bugs.mysql.com/591])
- New `myisam_data_pointer_size` system variable. See Section 5.2.3, “Server System Variables”.
- The `--log-warnings` server option now is enabled by default. Disable with `--skip-log-warnings`.
- The `--defaults-file=`*`file_name`* option now requires that the filename must exist (safety fix). (Bug #3413 [http://bugs.mysql.com/3413])
- **mysqld_multi** now creates the log in the directory named by `datadir` (from the `[mysqld]` section in `my.cnf` or compiled in), not in `/tmp` (vulnerability ID CAN-2004-0388). Thanks to Christian Hammers from Debian Security Team for reporting this.
- **Warning: Incompatible change!** String comparison now works according to the SQL standard. Because we have that `'a' = 'a '` then from it must follow that `'a' > 'a\t'`. (The latter was not the case before MySQL 4.1.2.) To implement it, we had to change how storage engines compare strings internally. As a side effect, if you have a table where a `CHAR` or `VARCHAR` column in some row has a value with the last character less than `ASCII(32)`, you have to repair this table. `CHECK TABLES` tells you if this problem exists. (Bug #3152 [http://bugs.mysql.com/3152])
- Added support for `DEFAULT CURRENT_TIMESTAMP` and for `ON UPDATE CURRENT_TIMESTAMP` specifications for `TIMESTAMP` columns. Now you can explicitly say that a `TIMESTAMP` column should be set automatically to the current timestamp for `INSERT` and/or `UPDATE` statements, or even prevent the column from updating automatically. Only one column with such an auto-set feature per table is supported. `TIMESTAMP` columns created with earlier versions of MySQL behave as before. Behavior of `TIMESTAMP` columns that were created without explicit specification of default/on as earlier depends on its position in table: If it is the first `TIMESTAMP` column, it be treated as having been specified as `TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP`. In other cases, it would be treated as a `TIMESTAMP DEFAULT 0` column. `NOW` is supported as an alias for `CURRENT_TIMESTAMP`. **Warning: Incompatible change!** Unlike in previous versions, explicit specification of default values for `TIMESTAMP` column is never ignored and turns off the auto-set feature (unless you have `CURRENT_TIMESTAMP` as the default).
- **Warning: Incompatible change!** Renamed prepared statements C API functions:

Old Name	New Name
`mysql_bind_param()`	`mysql_stmt_bind_param()`
`mysql_bind_result()`	`mysql_stmt_bind_result()`
`mysql_prepare()`	`mysql_stmt_prepare()`
`mysql_execute()`	`mysql_stmt_execute()`
`mysql_fetch()`	`mysql_stmt_fetch()`
`mysql_fetch_column()`	`mysql_stmt_fetch_column()`

mysql_param_count()	mysql_stmt_param_count()
mysql_param_result()	mysql_stmt_param_metadata()
mysql_get_metadata()	mysql_stmt_result_metadata()
mysql_send_long_data()	mysql_stmt_send_long_data()

Now all functions that operate with a MYSQL_STMT structure begin with the prefix mysql_stmt_.

- **Warning: Incompatible change!** The signature of the mysql_stmt_prepare() function was changed to int mysql_stmt_prepare(MYSQL_STMT *stmt, const char *query, unsigned long length). To create a MYSQL_STMT handle, you should use the mysql_stmt_init() function, not mysql_stmt_prepare().
- SHOW GRANTS with no FOR clause or with FOR CURRENT_USER() shows the privileges for the current session.
- The improved character set support introduced in MySQL 4.1.0 for the MyISAM and HEAP storage engines is now available for InnoDB as well.
- A name of ``Primary'' no longer can be specified as an index name. (That name is reserved for the PRIMARY KEY if the table has one.) (Bug #856 [http://bugs.mysql.com/856])
- MySQL now issues a warning when a SET or ENUM column with duplicate values in the list is created. (Bug #1427 [http://bugs.mysql.com/1427])
- Now SQL_SELECT_LIMIT variable has no influence on subqueries. (Bug #2600 [http://bugs.mysql.com/2600])
- UNHEX() function implemented. See Section 12.3, “String Functions”.
- The **mysql** command-line client no longer stores in the history file multiple copies of identical queries that are run consecutively.
- Multi-line statements in the mysql command-line client now are stored in the history file as a single line.
- UUID() function implemented. Note that it does not work with replication yet. See Section 12.8.4, “Miscellaneous Functions”.
- Prepared statements with all types of subqueries fixed.
- MySQL now supports up to 64 indexes per table.
- MyISAM tables now support keys up to 1000 bytes long.
- MyISAM and InnoDB tables now support index prefix lengths up to 1000 bytes long.
- If you try to create a key with a key part that is too long, and it is safe to auto-truncate it to a smaller length, MySQL now does so. A warning is generated, rather than an error.
- The ft_boolean_syntax variable now can be changed while the server is running. See Section 5.2.3, “Server System Variables”.

- `REVOKE ALL PRIVILEGES, GRANT FROM user_list` is changed to a more consistent `REVOKE ALL PRIVILEGES, GRANT OPTION FROM user_list`. (Bug #2642 [http://bugs.mysql.com/2642])

- Internal string-to-number conversion now supports only SQL:2003 compatible syntax for numbers. In particular, `'0x10'+0` does not work anymore. (Actually, it worked only on some systems before, such as Linux. It did not work on others, such as FreeBSD or Solaris. Making these queries OS-independent was the goal of this change.) Use `CONV()` to convert hexadecimal numbers to decimal. E.g. `CONV(MID('0x10',3),16,10)+0`.

- **mysqlhotcopy** now works on NetWare.

- `ALTER TABLE DROP PRIMARY KEY` no longer drops the first `UNIQUE` index if there is no primary index. (Bug #2361 [http://bugs.mysql.com/2361])

- Added `latin1_spanish_ci` (Modern Spanish) collation for the `latin1` character set.

- Added the `ENGINE` table option as a synonym for the `TYPE` option for `CREATE TABLE` and `ALTER TABLE`.

- Added the `--default-storage-engine` server option as a synonym for `--default-table-type`.

- Added the `storage_engine` system variable as a synonym for `table_type`.

- **Warning: Incompatible change!** The `Type` output column for `SHOW TABLE STATUS` now is labeled `Engine`.

- Added `init_connect` and `init_slave` system variables. The values should be SQL statements to be executed when each client connects or each time a slave's SQL thread starts, respectively.

- C API enhancement: `SERVER_QUERY_NO_INDEX_USED` and `SERVER_QUERY_NO_GOOD_INDEX_USED` flags are now set in the `server_status` field of the `MYSQL` structure. It is these flags that make the query to be logged as slow if **mysqld** was started with `--log-slow-queries --log-queries-not-using-indexes`.

- For replication of `MEMORY` (`HEAP`) tables: Made the master automatically write a `DELETE FROM` statement to its binary log when a `MEMORY` table is opened for the first time since master's startup. This is for the case where the slave has replicated a non-empty `MEMORY` table, then the master is shut down and restarted: the table is now empty on master; the `DELETE FROM` empties it on slave too. Note that even with this fix, between the master's restart and the first use of the table on master, the slave still has out-of-date data in the table. But if you use the `init-file` option to populate the `MEMORY` table on the master at startup, it ensures that the failing time interval is zero. (Bug #2477 [http://bugs.mysql.com/2477])

- When a session having open temporary tables terminates, the statement automatically written to the binary log is now `DROP TEMPORARY TABLE IF EXISTS` instead of `DROP TEMPORARY TABLE`, for more robustness.

- The MySQL server now returns an error if `SET SQL_LOG_BIN` or `SET SQL_LOG_UPDATE` is issued by a user without the `SUPER` privilege (in previous versions it just silently ignored the statement in this case).

- Changed that when the MySQL server has binary logging disabled (that is, no `--log-bin` option was used), then no transaction binary log cache is allocated for connections. This should save `binlog_cache_size` bytes of memory (32KB by default) for every connection.

- Added `Binlog_cache_use` and `Binlog_cache_disk_use` status variables that count the

number of transactions that used transaction binary log and that had to flush this temporary binary log to disk instead of using only the in-memory buffer. They can be used for tuning the `binlog_cache_size` system variable.

- Added option `--replicate-same-server-id`.
- The Mac OS X Startup Item has been moved from the directory `/Library/StartupItems/MySQL` to `/Library/StartupItems/MySQLCOM` to avoid a file name collision with the MySQL Startup Item installed with Mac OS X Server. See Section 2.12.2, “Mac OS X Notes”.
- Added option `--to-last-log` to **mysqlbinlog**, for use in conjunction with `--read-from-remote-server`.

Bugs fixed:

- Fixed check of `EXPLAIN` of `UNION`. (Bug #3639 [http://bugs.mysql.com/3639])
- Fixed a bug in a query that used `DISTINCT` and `ORDER BY` by column's real name, while the column had an alias, specified in `SELECT` clause. (Bug #3681 [http://bugs.mysql.com/3681])
- **mysqld** could crash when a table was altered and used at the same time. This was a 4.1.2-specific bug. (Bug #3643 [http://bugs.mysql.com/3643]).
- Fixed bug when using impossible `WHERE` with `PROCEDURE ANALYSE()`. (Bug #2238 [http://bugs.mysql.com/2238]).
- Fixed security problem in new authentication where password was not checked for changed `GRANT` accounts until `FLUSH PRIVILEGES` was executed. (Bug #3404 [http://bugs.mysql.com/3404])
- Fixed crash of `GROUP_CONCAT()` on expression with `ORDER BY` and external `ORDER BY` in a query. (Bug #3752 [http://bugs.mysql.com/3752])
- Fixed a bug in `ALL`/`SOME` subqueries in case of optimization (key field present in subquery). (Bug #3646 [http://bugs.mysql.com/3646])
- Fixed a bug in `SHOW GRANTS` and `EXPLAIN SELECT` character set conversion. (Bug #3403 [http://bugs.mysql.com/3403])
- Prepare statements parameter do not cause error message as fields used in select list but not included in `ORDER BY` list.
- `UNION` statements did not consult `SQL_SELECT_LIMIT` value when set. This is now fixed properly, which means that this limit is applied to the top level query, unless `LIMIT` for entire `UNION` is used.
- Fixed a bug in multiple-table `UPDATE` statements that resulted in an error when one of the tables was not updated but was used in the nested query, contained therein.
- Fixed `mysql_stmt_send_long_data()` behavior on second execution of prepared statement and in case when long data had zero length. (Bug #1664 [http://bugs.mysql.com/1664])
- Fixed crash on second execution of prepared statement with `UNION`. (Bug #3577 [http://bugs.mysql.com/3577])
- Fixed incorrect results of aggregate functions in subquery with empty result set. (Bug #3505 [http://bugs.mysql.com/3505])

- You can now call `mysql_stmt_attr_set(..., STMT_ATTR_UPDATE_MAX_LENGTH)` to tell the client library to update `MYSQL_FIELD->max_length` when doing `mysql_stmt_store_result()`. (Bug #1647 [http://bugs.mysql.com/1647]).
- Added support for unsigned integer types to prepared statement API (Bug #3035 [http://bugs.mysql.com/3035]).
- Fixed crash in prepared statements when subquery in the `FROM` clause with parameter used. (Bug #3020 [http://bugs.mysql.com/3020])
- Fixed unknown error when negative value bind to unsigned. (Bug #3223 [http://bugs.mysql.com/3223])
- Fixed aggregate function in prepared statements. (Bug #3360 [http://bugs.mysql.com/3360])
- Incorrect error message when wrong table used in multiple-table `DELETE` statement in prepared statements. (Bug #3411 [http://bugs.mysql.com/3411])
- Requiring `UPDATE` privilege for tables which are not updated in multiple-table `UPDATE` statement in prepared statements.
- Fixed prepared statement support for `INSERT`, `REPLACE`, `CREATE`, `DELETE`, `SELECT`, `DO`, `SET` and `SHOW`. All other commands are prohibited via prepared statement interface. (Bug #3398 [http://bugs.mysql.com/3398], Bug #3406 [http://bugs.mysql.com/3406], Bug #2811 [http://bugs.mysql.com/2811])
- Fixed a lot of bugs in `GROUP_CONCAT()`. (Bug #2695 [http://bugs.mysql.com/2695], Bug #3381 [http://bugs.mysql.com/3381], Bug #3319 [http://bugs.mysql.com/3319])
- Added optimization that allows for prepared statements using a large number of tables or tables with a large number of columns to be re-executed significantly faster. (Bug #2050 [http://bugs.mysql.com/2050])
- Fixed bug that caused execution of prepared statements to fail then table that this statement were using left table cache. This bug showed up as if this prepared statement used random garbage as column names or as server crashes. (Bug #3307 [http://bugs.mysql.com/3307])
- Fixed a problem resulting from setting the `character_set_results` variable to `NULL`. (Bug #3296 [http://bugs.mysql.com/3296])
- Fixed query cache statistics.
- Fixed bug in `ANALYZE TABLE` on a `BDB` table inside a transaction that hangs server thread. (Bug #2342 [http://bugs.mysql.com/2342])
- Fixed a symlink vulnerability in the `mysqlbug` script. (Bug #3284 [http://bugs.mysql.com/3284])
- Fixed a bug in parallel repair (**myisamchk -p**, `myisam_repair_threads`); sometimes the repair process failed to repair a table. (Bug #1334 [http://bugs.mysql.com/1334])
- A query that uses both `UNION [DISTINCT]` and `UNION ALL` now works correctly. (Bug #1428 [http://bugs.mysql.com/1428])
- Table default character set affects `LONGBLOB` columns. (Bug #2821 [http://bugs.mysql.com/2821])
- `CONCAT_WS()` makes the server die in case of illegal mix of collations. (Bug #3087 [http://bugs.mysql.com/3087])
- UTF8 charset breaks joins with mixed column/string constant. (Bug #2959

[http://bugs.mysql.com/2959])

- Fixed `DROP DATABASE` to report number of tables deleted.
- Fixed memory leak in the client library when statement handle was freed on closed connection (call to `mysql_stmt_close` after `mysql_close`). (Bug #3073 [http://bugs.mysql.com/3073])
- Fixed server segfaults when processing malformed prepared statements commands. (Bug #2795 [http://bugs.mysql.com/2795], Bug #2274 [http://bugs.mysql.com/2274])
- Fixed using subqueries with `OR` and `AND` functions. (Bug #2838 [http://bugs.mysql.com/2838])
- Fixed comparison of tables/database names with `--lower_case_table_names` option. (Bug #2880 [http://bugs.mysql.com/2880])
- Removed try to check `NULL` if index built on column where `NULL` is impossible in `IN` subquery optimization. (Bug #2393 [http://bugs.mysql.com/2393])
- Fixed incorrect parsing of subqueries in the `FROM` clause. (Bug #2421 [http://bugs.mysql.com/2421])
- Fixed processing of `RAND()` in subqueries with static tables. (bug #2645 [http://bugs.mysql.com/2645])
- Fixed bug with quoting of table names in **mysqldump** for various values of `sql_mode` of server. (Bug #2591 [http://bugs.mysql.com/2591])
- Fixed bug with storing values that are out of range for `DOUBLE` and `FLOAT` columns. (Bug #2082 [http://bugs.mysql.com/2082])
- Fixed bug with compiling `--with-pstack` with binutils 2.13.90. (Bug #1661 [http://bugs.mysql.com/1661])
- Fixed a bug in the `GRANT` system. When a password was assigned to an account at the global level and then privileges were granted at the database level (without specifying any password), the existing password was replaced temporarily in memory until the next `FLUSH PRIVILEGES` operation or the server was restarted. (Bug #2953 [http://bugs.mysql.com/2953])
- Fixed a bug in full-text search on multi-byte character set (such as UTF8) that appeared when a search word was shorter than a matching word from the index (for example, searching for ``Uppsala'' when table data contain ``Uppsa*la''). (Bug #3011 [http://bugs.mysql.com/3011])
- Fixed a bug that made `Max_used_connections` to be less than the actual maximum number of connections in use simultaneously.
- Fixed calculation of `Index_length` in `HEAP` table status for `BTREE` indexes. (Bug #2719 [http://bugs.mysql.com/2719])
- Fixed `mysql_stmt_affected_rows()` call to always return number of rows affected by given statement. (Bug #2247 [http://bugs.mysql.com/2247])
- Fixed crash in `MATCH ... AGAINST()` on a phrase search operator with a missing closing double quote. (Bug #2708 [http://bugs.mysql.com/2708])
- Fixed output of **mysqldump --tab**. (Bug #2705 [http://bugs.mysql.com/2705])
- Fix for a bug in `UNION` operations that prevented proper handling of `NULL` columns. This happened only if a column in the first `SELECT` node was `NOT NULL`. (Bug #2508 [http://bugs.mysql.com/2508])

- Fix for a bug in `UNION` operations with `InnoDB` storage engine, when some columns from one table were used in one `SELECT` statement and some were used in another `SELECT` statement. (Bug #2552 [http://bugs.mysql.com/2552])
- Fixed a few years old bug in the range optimizer that caused a segmentation fault on some very rare queries. (Bug #2698 [http://bugs.mysql.com/2698])
- Fixed bug with `SHOW CREATE TABLE ...` which didn't properly double quotes. (Bug #2593 [http://bugs.mysql.com/2593])
- Queries with subqueries in `FROM` clause locks all tables at once for now. This also fixed bugs in `EXPLAIN` of subqueries in `FROM` output. (Bug #2120 [http://bugs.mysql.com/2120])
- Fixed bug with **mysqldump** not quoting ``tricky'' names correctly. (Bug #2592 [http://bugs.mysql.com/2592])
- Fix for a bug that prevented table / column privileges from being loaded on startup. (Bug #2546 [http://bugs.mysql.com/2546])
- Fixed bug in replication with `CREATE TABLE ... LIKE ...` that resulted in a statement not being written to the binary log. (Bug #2557 [http://bugs.mysql.com/2557])
- Fixed memory leak in `INSERT ... ON DUPLICATE KEY UPDATE ....` (Bug #2438 [http://bugs.mysql.com/2438])
- Fixed bug in the parser, making the syntax `CONVERT(`*`expr`*`,`*`type`*`)` legal again.
- Fixed parsing of short-form IP addresses in `INET_ATON()`. (Bug #2310 [http://bugs.mysql.com/2310])
- Fixed a bug in `CREATE ... SELECT` that sometimes caused a string column with a multi-byte character set (such as `utf8`) to have insufficient length to hold the data.
- Fixed a rare table corruption on adding data (`INSERT`, `REPLACE`, `UPDATE`, etc. but not `DELETE`) to a `FULLTEXT` index. (Bug #2417 [http://bugs.mysql.com/2417])
- Compile the `MySQL-client` RPM package against `libreadline` instead of `libedit`. (Bug #2289 [http://bugs.mysql.com/2289])
- Fix for a crashing bug that was caused by not setting `vio_timeout()` virtual function for all protocols. This bug occurred on Windows. (Bug #2025 [http://bugs.mysql.com/2025])
- Fix for a bug that caused **mysql** client program to erroneously cache the value of the current database. (Bug #2025 [http://bugs.mysql.com/2025])
- Fix for a bug that caused client/server communication to be broken when `mysql_set_server_option()` or `mysql_get_server_option()` were invoked. (Bug #2207 [http://bugs.mysql.com/2207])
- Fix for a bug that caused wrong results when `CAST()` was applied on `NULL` to signed or unsigned integer column. (Bug #2219 [http://bugs.mysql.com/2219])
- Fix for a crashing bug that occurred in the **mysql** client program when database name was longer then expected. (Bug #2221 [http://bugs.mysql.com/2221])
- Fixed a bug in `CHECK TABLE` that sometimes resulted in a spurious error `Found key at page ... that points to record outside datafile` for a table with a `FULLTEXT` index. (Bug #2190 [http://bugs.mysql.com/2190])

- Fixed bug in `GRANT` with table-level privilege handling. (Bug #2178 [http://bugs.mysql.com/2178])
- Fixed bug in `ORDER BY` on a small column. (Bug #2147 [http://bugs.mysql.com/2147])
- Fixed a bug with the `INTERVAL()` function when 8 or more comparison arguments are provided. (Bug #1561 [http://bugs.mysql.com/1561])
- Packaging: Fixed a bug in the Mac OS PKG `postinstall` script (**mysql_install_db** was called with an obsolete argument).
- Packaging: Added missing file `mysql_create_system_tables` to the server RPM package. This bug was fixed for the 4.1.1 RPMs by updating the MySQL-server RPM from `MySQL-server-4.1.1-0` to `MySQL-server-4.1.1-1`. The other RPMs were not affected by this change.
- Fixed a bug in **myisamchk** and `CHECK TABLE` that sometimes resulted in a spurious error `Found key at page ... that points to record outside datafile` for a table with a `FULLTEXT` index. (Bug #1977 [http://bugs.mysql.com/1977])
- Fixed a hang in full-text indexing of strings in multi-byte (all besides `utf8`) charsets. (Bug #2065 [http://bugs.mysql.com/2065])
- Fixed a crash in full-text indexing of UTF8 data. (Bug #2033 [http://bugs.mysql.com/2033])
- Replication: a rare race condition in the slave SQL thread that could lead to an incorrect complaint that the relay log is corrupted. (Bug #2011 [http://bugs.mysql.com/2011])
- Replication: If a client connects to a slave server and issues an administrative statement for a table (for example, `OPTIMIZE TABLE` or `REPAIR TABLE`), this could sometimes stop the slave SQL thread. This does not lead to any corruption, but you must use `START SLAVE` to get replication going again. (Bug #1858 [http://bugs.mysql.com/1858])
- Replication: in the slave SQL thread, a multiple-table `UPDATE` could produce an incorrect complaint that some record was not found in one table, if the `UPDATE` was preceded by a `INSERT ... SELECT`. (Bug #1701 [http://bugs.mysql.com/1701])
- Replication: sometimes the master gets a non-fatal error during the execution of a statement but finally the statements succeeds (for example, a write to a `MyISAM` table first receives "no space left on device" but is able to finally complete, see Section A.4.3, “How MySQL Handles a Full Disk”); the bug was that the master forgot to reset the error code to 0 after success, so the error code got into its binary log, thus making the slave giving false alarms like "did not get the same error as on master". (Bug #2083 [http://bugs.mysql.com/2083])
- Removed a misleading "check permissions on master.info" from a replication error message, because the cause of the problem could be different from permissions. (Bug #2121 [http://bugs.mysql.com/2121])
- Fixed a crash when the replication slave was unable to create the first relay log. (Bug #2145 [http://bugs.mysql.com/2145])
- `ALTER DATABASE` caused the client to hang if the database did not exist. (Bug #2333 [http://bugs.mysql.com/2333])
- Multiple-table `DELETE` statements were never replicated by the slave if there were any `--replicate-*-table` options. (Bug #2527 [http://bugs.mysql.com/2527])
- Fixed bug in `ALTER TABLE RENAME`, when rename to the table with the same name in another database silently dropped destination table if it existed. (Bug #2628 [http://bugs.mysql.com/2628])
- The MySQL server did not report any error if a statement (submitted through

mysql_real_query() or mysql_stmt_prepare()) was terminated by garbage characters. This can happen if you pass a wrong length parameter to these functions. The result was that the garbage characters were written into the binary log. (Bug #2703 [http://bugs.mysql.com/2703])

- Fixed bug in client library that caused mysql_stmt_fetch and mysql_stmt_store_result() to hang if they were called without prior call of mysql_stmt_execute(). Now they give an error instead. (Bug #2248 [http://bugs.mysql.com/2248])

- Made clearer the error message that one gets when an update is refused because of the --read-only option. (Bug #2757 [http://bugs.mysql.com/2757])

- Fixed that --replicate-wild-*-table rules apply to ALTER DATABASE when the table pattern is %, as is the case for CREATE DATABASE and DROP DATABASE. (Bug #3000 [http://bugs.mysql.com/3000])

- Fixed that when a Rotate event is found by the slave SQL thread in the middle of a transaction, the value of Relay_Log_Pos in SHOW SLAVE STATUS remains correct. (Bug #3017 [http://bugs.mysql.com/3017])

- Corrected the master's binary log position that InnoDB reports when it is doing a crash recovery on a slave server. (Bug #3015 [http://bugs.mysql.com/3015])

- Changed the column Seconds_Behind_Master in SHOW SLAVE STATUS to never show a value of #1. (Bug #2826 [http://bugs.mysql.com/2826])

- Changed that when a DROP TEMPORARY TABLE statement is automatically written to the binary log when a session ends, the statement is recorded with an error code of value zero (this ensures that killing a SELECT on the master does not result in a superfluous error on the slave). (Bug #3063 [http://bugs.mysql.com/3063])

- Changed that when a thread handling INSERT DELAYED (also known as a delayed_insert thread) is killed, its statements are recorded with an error code of value zero (killing such a thread does not endanger replication, so we thus avoid a superfluous error on the slave). (Bug #3081 [http://bugs.mysql.com/3081])

- Fixed deadlock when two START SLAVE commands were run at the same time. (Bug #2921 [http://bugs.mysql.com/2921])

- Fixed that a statement never triggers a superfluous error on the slave, if it must be excluded given the --replicate-* options. The bug was that if the statement had been killed on the master, the slave would stop. (Bug #2983 [http://bugs.mysql.com/2983])

- The --local-load option of **mysqlbinlog** now requires an argument.

- Fixed a segmentation fault when running LOAD DATA FROM MASTER after RESET SLAVE. (Bug #2922 [http://bugs.mysql.com/2922])

- **mysqlbinlog --read-from-remote-server** read all binary logs following the one that was requested. It now stops at the end of the requested file, the same as it does when reading a local binary log. There is an option --to-last-log to get the old behavior. (Bug #3204 [http://bugs.mysql.com/3204])

- Fixed **mysqlbinlog --read-from-remote-server** to print the exact positions of events in the "at #" lines. (Bug #3214 [http://bugs.mysql.com/3214])

- Fixed a rare error condition that caused the slave SQL thread spuriously to print the message Binlog has bad magic number and stop when it was not necessary to do so. (Bug #3401 [http://bugs.mysql.com/3401])

- Fixed the `Exec_master_log_pos` column and its disk image in the `relay-log.info` file to be correct if the master had version 3.23. (The value was too big by six bytes.) This bug does not exist in MySQL 5.0. (Bug #3400 [http://bugs.mysql.com/3400])
- Fixed **mysqlbinlog** not to forget to print a `USE` statement under rare circumstances where the binary log contained a `LOAD DATA INFILE` statement. (Bug #3415 [http://bugs.mysql.com/3415])
- Fixed a memory corruption when replicating a `LOAD DATA INFILE` when the master had version 3.23. Some smaller problems remain in this setup, See Section 6.7, “Replication Features and Known Problems”. (Bug #3422 [http://bugs.mysql.com/3422])
- Multiple-table `DELETE` statements were always replicated by the slave if there were some `--replicate-*-ignore-table` options and no `--replicate-*-do-table` options. (Bug #3461 [http://bugs.mysql.com/3461])
- Fixed a crash of the MySQL slave server when it was built with `--with-debug` and replicating itself. (Bug #3568 [http://bugs.mysql.com/3568])
- Fixed that in some replication error messages, a very long query caused the rest of the message to be invisible (truncated), by putting the query last in the message. (Bug #3357 [http://bugs.mysql.com/3357])
- Fixed a bug in `REPAIR TABLE` that resulted sometimes in a corrupted table, if the table contained `FULLTEXT` indexes and many words of different lengths that are considered equal (which is possible in certain collations, such as `latin1_german2_ci` or `utf8_general_ci`). (Bug #3835 [http://bugs.mysql.com/3835])
- Fixed a crash of **mysqld** that was started with binary logging disabled, but with a non-zero value for the `expire_logs_days` system variable. (Bug #3807 [http://bugs.mysql.com/3807])
- If `server-id` was not set using startup options but with `SET GLOBAL`, the replication slave still complained that it was not set. (Bug #3829)

D.2.11. Changes in release 4.1.1 (01 Dec 2003)

This release includes all fixes in MySQL 4.0.16 and most of the fixes in MySQL 4.0.17.

Functionality added or changed:

- **Important note:** If you upgrade to `InnoDB`-4.1.1 or higher, you cannot downgrade to a version lower than 4.1.1 any more! That is because earlier versions of `InnoDB` are not aware of multiple tablespaces.
- **Warning: Incompatible change!** Renamed the C API `mysql_prepare_result()` function to `mysql_get_metadata()` because the old name was confusing.
- **Warning: Incompatible change!** Client authentication now is based on 41-byte passwords in the `user` table, not 45-byte passwords as in 4.1.0. Any 45-byte passwords created for 4.1.0 must be reset after running the **mysql_fix_privilege_tables** script.
- Added `secure_auth` global server system variable and `--secure-auth` server option that disallow authentication for accounts that have old (pre-4.1.1) passwords.
- Added `--secure-auth` option to **mysql** command-line client. If this option is set, the client refuses to send passwords in old (pre-4.1.1) format.

- Table aliases are not case sensitive if `lower_case_table_names` is non-zero.
- The `--old-protocol` option for **mysqld** is no longer supported and has been removed.
- Renamed `bdb_version` system variable to `version_bdb`.
- New `CHECKSUM TABLE` statement for reporting table checksum values.
- Added `character_set_client`, `character_set_connection`, `character_set_database`, `character_set_results`, `character_set_server`, `character_set_system`, `collation_connection`, `collation_database`, and `collation_server` system variables to provide information about character sets and collations.
- It is now possible to create multiple key caches, assign table indexes to particular caches, and to preload indexes into caches. See Section 13.5.5.1, "`CACHE INDEX` Syntax". See Section 13.5.5.4, "`LOAD INDEX INTO CACHE` Syntax". Structured system variables are introduced as a means of grouping related key cache parameters. See Section 9.4.1, "Structured System Variables".
- Added `preload_buffer_size` system variable.
- New `COERCIBILITY()` function to return the collation coercibility of a string.
- The `--quote-names` option for **mysqldump** now is enabled by default.
- **mysqldump** now includes a statement in the dump output to set `FOREIGN_KEY_CHECKS` to 0 to avoid problems with tables having to be reloaded in a particular order when the dump is reloaded. The existing `FOREIGN_KEY_CHECKS` value is saved and restored.
- You can revoke all privileges from a user with `REVOKE ALL PRIVILEGES, GRANT FROM user_list`.
- Added `IGNORE` option for `DELETE` statement.
- MySQL source distributions now also include the MySQL Internals Manual `internals.texi`.
- Added `mysql_set_server_option()` C API client function to allow multiple statement handling in the server to be enabled or disabled.
- The `mysql_next_result()` C API function now returns `-1` if there are no more result sets.
- Renamed `CLIENT_MULTI_QUERIES` connect option flag to `CLIENT_MULTI_STATEMENTS`. To allow for a transition period, the old option continues to be recognized for a while.
- Require `DEFAULT` before table and database default character set. This enables us to use `ALTER TABLE tbl_name ... CHARACTER SET=...` to change the character set for all `CHAR`, `VARCHAR`, and `TEXT` columns in a table.
- Added `MATCH ... AGAINST( ... WITH QUERY EXPANSION)` and the `ft_query_expansion_limit` system variable.
- Removed unused `ft_max_word_len_for_sort` system variable.
- Removed unused `ft_max_word_len_for_sort` variable from **myisamchk**.
- Full-text search now supports multi-byte character sets and the Unicode `utf8` character set. (The Unicode `ucs2` character set is not yet supported.)
- Phrase search in `MATCH ... AGAINST ( ... IN BOOLEAN MODE)` no longer matches partial words.

- Added aggregate function `BIT_XOR()` for bitwise XOR operations.
- Replication over SSL now works.
- The `START SLAVE` statement now supports an `UNTIL` clause for specifying that the slave SQL thread should be started but run only until it reaches a given position in the master's binary logs or in the slave's relay logs.
- Produce warnings even for single-row `INSERT` statements, not just for multiple-row `INSERT` statements. Previously, it was necessary to set `SQL_WARNINGS=1` to generate warnings for single-row statements.
- Added `delimiter` (`\d`) command to the **mysql** command-line client for changing the statement delimiter (terminator). The default delimiter is semicolon.
- `CHAR`, `VARCHAR`, and `TEXT` columns now have lengths measured in characters rather than in bytes. The character size depends on the column's character set. This means, for example, that a `CHAR(n)` column for a multi-byte character set takes more storage than before. Similarly, index values on such columns are measured in characters, not bytes.
- `LIMIT` no longer accepts negative arguments (they used to be treated as very big positive numbers before).
- The `DATABASE()` function now returns `NULL` rather than the empty string if there is no database selected.
- Added `--sql-mode=NO_AUTO_VALUE_ON_ZERO` option to suppress the usual behavior of generating the next sequence number when zero is stored in an `AUTO_INCREMENT` column. With this mode enabled, zero is stored as zero; only storing `NULL` generates a sequence number.
- Added `DROP USER 'user_name'@'host_name'` statement to drop an account that has no privileges.
- The interface to aggregate user-defined functions has changed a bit. You must now declare a `xxx_clear()` function for each aggregate function `XXX()`. `xxx_clear()` is used instead of `xxx_reset()`.
- Added new `ADDTIME()`, `DATE()`, `DATEDIFF()`, `LAST_DAY()`, `MAKEDATE()`, `MAKETIME()`, `MICROSECOND()`, `SUBTIME()`, `TIME()`, `TIMEDIFF()`, `TIMESTAMP()`, `UTC_DATE()`, `UTC_TIME()`, `UTC_TIMESTAMP()`, and `WEEKOFYEAR()` functions.
- Added new syntax for `ADDDATE()` and `SUBDATE()`. The second argument now may be a number representing the number of days to be added to or subtracted from the first date argument.
- Added new `type` values `DAY_MICROSECOND`, `HOUR_MICROSECOND`, `MINUTE_MICROSECOND`, `SECOND_MICROSECOND`, and `MICROSECOND` for `DATE_ADD()`, `DATE_SUB()`, and `EXTRACT()`.
- Added new `%f` microseconds format specifier for `DATE_FORMAT()` and `TIME_FORMAT()`.
- All queries in which at least one `SELECT` does not use indexes properly now are written to the slow query log when long log format is used.
- It is now possible to create a `MERGE` table from `MyISAM` tables in different databases. Formerly, all the `MyISAM` tables had to be in the same database, and the `MERGE` table had to be created in that database as well.
- Added new `COMPRESS()`, `UNCOMPRESS()`, and `UNCOMPRESSED_LENGTH()` functions.

- When using `SET sql_mode='mode'` for a complex mode (such as `ANSI`), we now update the `sql_mode` variable to include all the individual options implied by the complex mode.
- Added the OLAP (On-Line Analytical Processing) function `ROLLUP`, which provides summary rows for each `GROUP BY` level.
- Added `SQLSTATE` codes for all server errors.
- Added `mysql_sqlstate()` and `mysql_stmt_sqlstate()` C API client functions that return the `SQLSTATE` error code for the last error.
- `TIME` columns with hour values greater than 24 were returned incorrectly to the client.
- `ANALYZE TABLE`, `OPTIMIZE TABLE`, `REPAIR TABLE`, and `FLUSH` statements are now stored in the binary log and thus replicated to slaves. This logging does not occur if the optional `NO_WRITE_TO_BINLOG` keyword (or its alias `LOCAL`) is given. Exceptions are that `FLUSH LOGS`, `FLUSH MASTER`, `FLUSH SLAVE`, and `FLUSH TABLES WITH READ LOCK` are not logged in any case. For a syntax example, see Section 13.5.5.2, "`FLUSH` Syntax".
- New global system variable `relay_log_purge` to enable or disable automatic relay log purging.
- `LOAD DATA` now produces warnings that can be fetched with `SHOW WARNINGS`.
- Added support for syntax `CREATE TABLE table2 (LIKE table1)` that creates an empty table `table2` with a definition that is exactly the same as `table1`, including any indexes.
- `CREATE TABLE` *`tbl_name`* `(...) TYPE=`*`storage_engine`* now generates a warning if the named storage engine is not available. The table is still created as a `MyISAM` table, as before.
- Most subqueries are now much faster than before.
- Added `PURGE BINARY LOGS` as an alias for `PURGE MASTER LOGS`.
- Disabled the `PURGE LOGS` statement that was added in version 4.1.0. The statement now should be issued as `PURGE MASTER LOGS` or `PURGE BINARY LOGS`.
- Added `SHOW BDB LOGS` as an alias for `SHOW LOGS`.
- Added `SHOW MASTER LOGS` as an alias for `SHOW BINARY LOGS`. (In 4.1.0, `SHOW MASTER LOGS` was renamed to `SHOW BINARY LOGS`. Now you can use either one.)
- Added `Slave_IO_State` and `Seconds_Behind_Master` columns to the output of `SHOW SLAVE STATUS`. `Slave_IO_State` indicates the state of the slave I/O thread, and `Seconds_Behind_Master` indicates the number of seconds by which the slave is late compared to the master.
- The `--lower-case-table-names=1` server option now also makes aliases case insensitive. (Bug #534 [http://bugs.mysql.com/534])
- Changed that the relay log is flushed to disk by the slave I/O thread every time it reads a relay log event. This reduces the risk of losing some part of the relay log in case of brutal crash.

Bugs fixed:

- Fixed **mysql** parser not to erroneously interpret `';'` character within `/* ... */` comment as statement terminator.

- Fixed merging types and length of result set columns for `UNION` operations. The types and lengths now are determined taking into account values for all `SELECT` statements in the `UNION`, not just the first `SELECT`.
- Fixed a bug in privilege handling that caused connections from certain IP addresses to be assigned incorrect database-level privileges. A connection could be assigned the database privileges of the previous successful authentication from one of those IP addresses, even if the IP address username and database name were different. (Bug #1636 [http://bugs.mysql.com/1636])
- Error-handling functions were not called properly when an error resulted from `[CREATE | REPLACE| INSERT] ... SELECT` statements.
- `HASH`, `BTREE`, `RTREE`, `ERRORS`, and `WARNINGS` no longer are reserved words. (Bug #724 [http://bugs.mysql.com/724])
- Fix for bug in `ROLLUP` when all tables were `const` tables. (Bug #714 [http://bugs.mysql.com/714])
- Fixed a bug in `UNION` that prohibited `NULL` values from being inserted into result set columns where the first `SELECT` of the `UNION` retrieved `NOT NULL` columns. The type and max_length of the result column is now defined based on all `UNION` parts.
- Fixed name resolution of columns of reduced subqueries in unions. (Bug #745 [http://bugs.mysql.com/745])
- Fixed memory overrun in subqueries in select list with `WHERE` clause bigger than outer query `WHERE` clause. (Bug #726 [http://bugs.mysql.com/726])
- Fixed a bug that caused `MyISAM` tables with `FULLTEXT` indexes created in 4.0.x to be unreadable in 4.1.x.
- Fixed a data loss bug in `REPAIR TABLE ... USE_FRM` when used with tables that contained `TIMESTAMP` columns and were created in 4.0.x.
- Fixed reduced subquery processing in `ORDER BY`/`GROUP BY` clauses. (Bug #442 [http://bugs.mysql.com/442])
- Fixed name resolution of outer columns of subquery in `INSERT`/`REPLACE` statements. (Bug #446 [http://bugs.mysql.com/446])
- Fixed bug in marking columns of reduced subqueries. (Bug #679 [http://bugs.mysql.com/679])
- Fixed a bug that made `CREATE FULLTEXT INDEX` syntax illegal.
- Fixed a crash when a `SELECT` that required a temporary table (marked by `Using temporary` in `EXPLAIN` output) was used as a derived table in `EXPLAIN` command. (Bug #251 [http://bugs.mysql.com/251])
- Fixed a rare table corruption bug in `DELETE` from a big table with a **new** (created by MySQL-4.1) full-text index.
- `LAST_INSERT_ID()` now returns 0 if the last `INSERT` statement didn't insert any rows.
- Fixed missing last character in function output. (Bug #447 [http://bugs.mysql.com/447])
- Fixed a rare replication bug when a transaction spanned two or more relay logs, and the slave was stopped while executing the part of the transaction that was in the second or later relay log. Then replication would resume at the beginning of the second or later relay log, which was incorrect. (It should resume at `BEGIN`, in the first relay log.) (Bug #53 [http://bugs.mysql.com/53])

- CONNECTION_ID() now is properly replicated. (Bug #177 [http://bugs.mysql.com/177])
- The new PASSWORD() function in 4.1 is now properly replicated. (Bug #344 [http://bugs.mysql.com/344])
- Fixed a bug with double freed memory.
- Fixed a crashing bug in UNION operations that involved temporary tables.
- Fixed a crashing bug in DERIVED TABLES when EXPLAIN is used on a DERIVED TABLES with a join.
- Fixed a crashing bug in DELETE with ORDER BY and LIMIT caused by an uninitialized array of reference pointers.
- Fixed a bug in the USER() function caused by an error in the size of the allocated string.
- Fixed a crashing bug when attempting to create a table containing a spatial (GIS) column with a storage engine that does not support spatial types.
- Fixed a crashing bug in UNION caused by the empty select list and a non-existent column being used in some of the individual SELECT statements.
- Fixed a replication bug with a 3.23 master and a 4.0 slave: The slave lost the replicated temporary tables if FLUSH LOGS was issued on the master. (Bug #254 [http://bugs.mysql.com/254])
- Fixed a security bug: A server compiled without SSL support still allowed connections by users who had the REQUIRE SSL option specified for their accounts.
- When an undefined user variable was used in a updating query on the master (such as INSERT INTO t VALUES(@a), where @a had never been set by this connection before), the slave could replicate the query incorrectly if a previous transaction on the master used a user variable of the same name. (Bug #1331 [http://bugs.mysql.com/1331])
- Fixed bug with prepared statements: Using the ? prepared statement parameter as the argument to certain functions or statement clauses caused a server crash when mysql_prepare() was invoked. (Bug #1500 [http://bugs.mysql.com/1500])
- Fixed bug with prepared statements: after call to mysql_prepare placeholders became allowed in all consequent statements, even if they are not prepared (Bug #1946 [http://bugs.mysql.com/1946])
- SLAVE START (which is a deprecated syntax, START SLAVE should be used instead) could crash the slave. (Bug #2516 [http://bugs.mysql.com/2516])
- Fixed bug in ALTER TABLE RENAME, when rename to the table with the same name in another database silently dropped destination table if it existed. (Bug #2628 [http://bugs.mysql.com/2628])
- When not specify hostname in SET PASSWORD FOR user it's now defaulted to % instead of the current host.

D.2.12. Changes in release 4.1.0 (03 Apr 2003: Alpha)

Functionality added or changed:

- Renamed SHOW MASTER LOGS statement to SHOW BINARY LOGS.
- Allow DEFAULT(*col_name*) in expressions; it produces the column's default value.

- Added `--compatible` option to **mysqldump** for producing output that is compatible with other database systems or with older MySQL servers.
- The `--opt` option for **mysqldump** now is enabled by default, as are all the options implied by `--opt`.
- New `CHARSET()` and `COLLATION()` functions to return the character set and collation of a string.
- Allow index type to be specified explicitly for some storage engines via `USING type_name` syntax in index definition.
- New function `IS_USED_LOCK()` for determining the connection identifier of the client that holds a given advisory lock.
- New more secure client authentication based on 45-byte passwords in the `user` table.
- Added `old-password` command to **mysqladmin** for changing password but storing it using the old password-hashing format.
- New `CRC32()` function to compute cyclic redundancy check value.
- On Windows, we are now using shared memory to communicate between server and client when they are running on the same machine and you are connecting to `localhost`.
- `REPAIR TABLE` of `MyISAM` tables now uses less temporary disk space when sorting char columns.
- `DATE`/`DATETIME` checking is now a bit stricter to support the ability to automatically distinguish between date, datetime, and time with microseconds. For example, dates of type `YYYYMMDD HHMMDD` are no longer supported; you must either have separators between each `DATE`/`TIME` part or not at all.
- Server side help for all MySQL functions. One can now type `help week` in the **mysql** client and get help for the `week()` function.
- Added new `mysql_get_server_version()` C API client function.
- Fixed bug in `libmysqlclient` that fetched column defaults.
- Fixed bug in `mysql` command-line client in interpreting quotes within comments. (Bug #539 [http://bugs.mysql.com/539])
- Added `record_in_range()` method to `MERGE` tables to be able to choose the right index when there are many to choose from.
- Replication now works with `RAND()` and user variables `@var`.
- Allow one to change mode for `ANSI_QUOTES` on the fly.
- `EXPLAIN SELECT` now can be killed. See Section 13.5.5.3, “`KILL` Syntax”.
- `REPAIR TABLE` and `OPTIMIZE TABLE` now can be killed. See Section 13.5.5.3, “`KILL` Syntax”.
- Allow empty index lists to be specified for `USE INDEX`, `IGNORE INDEX`, and `FORCE INDEX`.
- `DROP TEMPORARY TABLE` now drops only temporary tables and doesn't end transactions.
- Added support for `UNION` in derived tables.
- **Warning: Incompatible change!** `TIMESTAMP` is now returned as a string of type `'YYYY-MM-DD`

HH:MM:SS' and different timestamp lengths are not supported.

This change was necessary for SQL standards compliance. In a future version, a further change will be made (backward compatible with this change), allowing the timestamp length to indicate the desired number of digits of fractions of a second.

- New faster client/server protocol that supports prepared statements, bound parameters, and bound result columns, binary transfer of data, warnings.
- Added database and real table name (in case of alias) to the MYSQL_FIELD structure.
- Multi-line queries: You can now issue several queries at once and then read the results in one go.
- In CREATE TABLE foo (a INT not null primary key) the PRIMARY word is now optional.
- In CREATE TABLE the attribute SERIAL is now an alias for BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE.
- SELECT ... FROM DUAL is an alias for SELECT .... (To be compatible with some other databases).
- If one creates a too long CHAR/VARCHAR it's now automatically changed to TEXT or BLOB; One get a warning in this case.
- One can specify the different BLOB/TEXT types with the syntax BLOB(length) and TEXT(length). MySQL automatically changes it to one of the internal BLOB/TEXT types.
- CHAR BYTE is an alias for CHAR BINARY.
- VARCHARACTER is an alias for VARCHAR.
- New operators integer MOD integer and integer DIV integer. DIV is now a reserved word.
- SERIAL DEFAULT VALUE added as an alias for AUTO_INCREMENT.
- TRUE and FALSE added as alias for 1 and 0, respectively.
- Aliases are now forced in derived tables, as per standard SQL.
- Fixed SELECT .. LIMIT 0 to return proper row count for SQL_CALC_FOUND_ROWS.
- One can specify many temporary directories to be used in a round-robin fashion with: --tmpdir=dirname1:dirname2:dirname3.
- Subqueries: SELECT * from t1 where t1.a=(SELECT t2.b FROM t2).
- Derived tables:

```
SELECT a.col1, b.col2
       FROM (SELECT MAX(col1) AS col1 FROM root_table) a,
       other_table b
       WHERE a.col1=b.col1;
```

- Character sets to be defined per column, table and database.
- Unicode (UTF8) support.

- New `CONVERT(... USING ...)` syntax for converting string values between character sets.
- `BTREE` index on `MEMORY` (`HEAP`) tables.
- Faster embedded server (new internal communication protocol).
- One can add a comment per column in `CREATE TABLE`.
- `SHOW FULL COLUMNS FROM tbl_name` shows column comments.
- `ALTER DATABASE`.
- Support for GIS (Geometrical data). See Chapter 18, *Spatial Extensions in MySQL*.
- `SHOW [COUNT(*)] WARNINGS` shows warnings from the last command.
- One can specify a column type for a column in `CREATE TABLE ... SELECT` by defining the column in the `CREATE` part.

  ```
  CREATE TABLE foo (a TINYINT NOT NULL) SELECT b+1 AS a FROM bar;
  ```

- `expr SOUNDS LIKE expr` same as `SOUNDEX(expr)=SOUNDEX(expr)`.
- Added new `VARIANCE(expr)` function returns the variance of `expr`
- One can create a table from the existing table using `CREATE [TEMPORARY] TABLE [IF NOT EXISTS] table (LIKE table)`. The table can be either normal or temporary.
- New options `--reconnect` and `--skip-reconnect` for the **mysql** client, to reconnect automatically or not if the connection is lost.
- `START SLAVE` (`STOP SLAVE`) no longer returns an error if the slave is started (stopped); it returns a warning instead.
- `SLAVE START` and `SLAVE STOP` are no longer accepted by the query parser; use `START SLAVE` and `STOP SLAVE` instead.

D.3. Changes in release 4.0.x (Production)

Version 4.0 of the MySQL server includes many enhancements and new features:

- The `InnoDB` storage engine is now included in the standard binaries, adding transactions, row-level locking, and foreign keys. See Chapter 15, *The `InnoDB` Storage Engine*.
- A query cache, offering vastly increased performance for many applications. By caching complete result sets, later identical queries can return instantly. See Section 5.11, “The MySQL Query Cache”.
- Improved full-text indexing with boolean mode, truncation, and phrase searching. See Section 12.6, “Full-Text Search Functions”.
- Enhanced `MERGE` tables, now supporting `INSERT` statements and `AUTO_INCREMENT`. See Section 14.2, “The `MERGE` Storage Engine”.
- `UNION` syntax in `SELECT`. See Section 13.1.7.2, “`UNION` Syntax”.
- Multiple-table `DELETE` statements. See Section 13.1.1, “`DELETE` Syntax”.

- `libmysqld`, the embedded server library. See Section 22.2.16, “libmysqld, the Embedded MySQL Server Library”.
- Additional `GRANT` privilege options for even tighter control and security. See Section 13.5.1.3, “`GRANT` and `REVOKE` Syntax”.
- Management of user resources in the `GRANT` system, particularly useful for ISPs and other hosting providers. See Section 5.6.4, “Limiting Account Resources”.
- Dynamic server variables, allowing configuration changes to be made without having to stop and restart the server. See Section 13.5.3, “`SET` Syntax”.
- Improved replication code and features. See Chapter 6, *Replication in MySQL*.
- Numerous new functions and options.
- Changes to existing code for enhanced performance and reliability.

For a full list of changes, please refer to the changelog sections for each individual 4.0.x release.

D.3.1. Changes in release 4.0.25 (not released yet)

Functionality added or changed:

-

Bugs fixed:

- An `UPDATE` that updated only some of the columns in a multiple-column index could result in a loop. (Bug #8942 [http://bugs.mysql.com/8942])
- `REPAIR TABLE` did not invalidate query results in the query cache that were generated from the table. (Bug #8480 [http://bugs.mysql.com/8480])
- `<=>` was not properly comparing `NULL` values in the `WHERE` clause of outer joins. (Bug #8711 [http://bugs.mysql.com/8711])
- InnoDB: Fixed a bug: MySQL-4.0.23 and 4.0.24 could complain that an InnoDB table created with MySQL-3.23.49 or earlier was in the new compact InnoDB table format of 5.0.3 or later, and InnoDB would refuse to use that table. (The same bug exists in 4.1.8 - 4.1.10.) There is nothing wrong with the table, it is mysqld that is in error. Workaround: wait that 4.0.25 or 4.1.11 is released before doing an upgrade, or dump the table and recreate it with any MySQL version >= 3.23.50 before upgrading to 4.0.23 or 4.0.24.

D.3.2. Changes in release 4.0.24 (04 Mar 2005)

Functionality added or changed:

- Security improvement: The server creates `.frm`, `.MYD`, `.MYI`, `.MRG`, `.ISD`, and `.ISM` table files only if a file with the same name does not already exist. (Thanks to Stefano Di Paola `<stefano.dipaola@wisec.it>` for finding and informing us about this issue.)
- Security improvement: User-defined functions should have at least one symbol defined in addition to

the xxx symbol that corresponds to the main xxx() function. These auxiliary symbols correspond to the xxx_init(), xxx_deinit(), xxx_reset(), xxx_clear(), and xxx_add() functions. **mysqld** by default no longer loads UDFs unless they have at least one auxiliary symbol defined in addition to the main symbol. The --allow-suspicious-udfs option controls whether UDFs that have only an xxx symbol can be loaded. By default, the option is off. mysqld also checks UDF filenames when it reads them from the mysql.func table and rejects those that contain directory pathname separator characters. (It already checked names as given in CREATE FUNCTION statements.) See Section 25.2.3.1, "UDF Calling Sequences for simple functions", Section 25.2.3.2, "UDF Calling Sequences for aggregate functions", and Section 25.2.3.6, "User-defined Function Security Precautions". (Thanks to Stefano Di Paola <stefano.dipaola@wisec.it> for finding and informing us about this issue.)

- InnoDB: Added configuration option and settable global variable innodb_autoextend_increment for setting the size in megabytes by which InnoDB tablespaces are extended when they become full. The default value is 8, corresponding to the fixed increment of 8MB in previous versions of MySQL.

- InnoDB: Do not acquire an internal InnoDB table lock in LOCK TABLES if AUTOCOMMIT=1. This helps in porting old MyISAM applications to InnoDB. InnoDB table locks in that case caused deadlocks very easily.

Bugs fixed:

- AES_DECRYPT(*col_name*,*key*) could fail to return NULL for invalid values in *col_name*, if *col_name* was declared as NOT NULL. (Bug #8669 [http://bugs.mysql.com/8669])

- FOUND_ROWS() returned an incorrect value after a SELECT SQL_CALC_FOUND_ROWS DISTINCT statement that selected constants and included GROUP BY and LIMIT clauses. (Bug #7945 [http://bugs.mysql.com/7945])

- Queries of the form (SELECT ...) ORDER BY ... were being treated as a UNION. This improperly resulted in only distinct values being returned (because UNION by default eliminates.) (Bug #7672 [http://bugs.mysql.com/7672])

- Index cardinality was not being updated properly for TEMPORARY tables under some circumstances, such as CREATE TABLE ... SELECT followed by ANALYZE TABLE. (Bug #7519 [http://bugs.mysql.com/7519])

- Fixed a server crash caused by DELETE FROM *tbl_name* ... WHERE ... ORDER BY *tbl_name.col_name* when the ORDER BY column was qualified with the table name. (Bug #8392 [http://bugs.mysql.com/8392])

- Fixed a bug in MATCH ... AGAINST in natural language mode that could cause a server crash if the FULLTEXT index was not used in a join (EXPLAIN did not show fulltext join mode) and the search query matched no rows in the table (Bug #8522 [http://bugs.mysql.com/8522]).

- Platform and architecture information in version information produced for --version option on Windows was always Win95/Win98 (i32). More accurately determine platform as Win32 or Win64 for 32-bit or 64-bit Windows, and architecture as ia32 for x86, ia64 for Itanium, and axp for Alpha. (Bug #4445 [http://bugs.mysql.com/4445])

- Fixed an optimization problem that allowed a negative number to be stored in a DOUBLE UNSIGNED column when it was assigned a value from a signed DOUBLE column. (Bug #7700 [http://bugs.mysql.com/7700])

- Fixed a failure of multiple-table updates to replicate properly on slave servers when --replicate-*-table options had been specified. (Bug #7011 [http://bugs.mysql.com/7011])

- Renamed `set_bit()` and `clear_bit()` functions in source code to avoid a conflict with functions of the same names in Linux kernel header files. (Bug #7971 [http://bugs.mysql.com/7971])
- Part of the information being used to cache access-permission lookups was not always reinitialized properly, particularly for connections from localhost on Windows. The result was connection failures that appeared to occur randomly. (Bug #5569 [http://bugs.mysql.com/5569])
- Corrected a problem with the `QUOTE()` function returning bad results. (Bug #8248 [http://bugs.mysql.com/8248])
- Fixed a problem where `INSERT INTO ...SELECT` failed when the source and target table were the same. (Bug #6034 [http://bugs.mysql.com/6034])
- Fixed a problem where RPM installation on Linux as a non-privileged user would result in incomplete installation. (Bug #7347 [http://bugs.mysql.com/7347])
- Change thread stack size used for building Linux RPM distributions to avoid warnings about stack size during server startup. (Bug #6226 [http://bugs.mysql.com/6226])
- Fixed a symlink vulnerability in the **mysqlaccess** script. Reported by Javier Fernandez-Sanguino Pena and Debian Security Audit Team [http://www.debian.org/security/audit]. (CAN-2005-0004 [http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2005-0004])
- Fixed support for C API function `mysql_list_fields()`, which was accidentally broken in 4.0.22 (Bug#6761)
- Make `query_cache_wlock_invalidate` system variable visible in `SHOW VARIABLES` output. (Bug #7594 [http://bugs.mysql.com/7594])
- Fixed a bug which caused `FROM_UNIXTIME()` function to return `NULL` for zero argument instead of the Epoch. (Bug #7515 [http://bugs.mysql.com/7515])
- Now in datetime values two digit year is interpreted as year in 20th or 21st century even with zero month and day. (Bug #7297 [http://bugs.mysql.com/7297])
- Fixed a bug in `QUOTE` function when used in conjunction with some other string functions. This lead to severe buffer overflow and server crashing. (Bug #7495 [http://bugs.mysql.com/7495])
- InnoDB: Work around a problem in AIX 5.1 patched with ML7 security patch: InnoDB would refuse to open its `ibdata` files, complaining about an operating system error 0.
- InnoDB: Fixed a memory corruption bug if one created a table with a primary key that contained at least two column prefixes. An example: `CREATE TABLE t(a char(100), b tinyblob, PRIMARY KEY(a(5), b(10)))`.
- InnoDB: Use native `tmpfile()` function on Netware. All InnoDB temporary files are created under `sys:\tmp`. Previously, InnoDB temporary files were never deleted on Netware.
- `InnoDB`: Honor the `--tmpdir` startup option when creating temporary files. Previously, `InnoDB` temporary files were always created in the temporary directory of the operating system. On Netware, `InnoDB` will continue to ignore `--tmpdir`. (Bug #5822 [http://bugs.mysql.com/5822])
- InnoDB: Fix a theoretical hang over the adaptive hash latch in InnoDB if one runs `INSERT ... SELECT ...` (binlog not enabled), or a multi-table `UPDATE` or `DELETE`, and only the read tables are InnoDB type, the rest are MyISAM; this also fixes bug #7879 for InnoDB type tables. (Bug #7879 [http://bugs.mysql.com/7879])
- InnoDB: Fixed a bug: 32-bit **mysqld** binaries built on HP-UX-11 did not work with `InnoDB` files greater than 2 GB in size. (Bug #6189 [http://bugs.mysql.com/6189])

- InnoDB: Fixed a bug: InnoDB failed to drop a table in the background drop queue if the table was referenced by a foreign key constraint.
- InnoDB: Fixed a bug: if we dropped a table where an `INSERT` was waiting for a lock to check a `FOREIGN KEY` constraint, then an assertion would fail in `lock_reset_all_on_table()`, since that operation assumes no waiting locks on the table or its records.
- Fixed that, when encountering a ``disk full'' or ``quota exceeded'' write error, `MyISAM` sometimes didn't sleep and retry the write, thus resulting in a corrupted table. (Bug #7714 [http://bugs.mysql.com/7714])
- Fixed that a slave could crash after replicating many `ANALYZE TABLE`, `OPTIMIZE TABLE`, or `REPAIR TABLE` statements from the master. (Bug #6461 [http://bugs.mysql.com/6461], Bug #7658 [http://bugs.mysql.com/7658])
- Fixed a bug where MySQL was allowing concurrent updates (inserts, deletes) to a table if binary logging is enabled. Changed to ensure that all updates are executed in a serialized fashion, because they are executed serialized when binlog is replayed. (Bug #7879 [http://bugs.mysql.com/7879])
- Fixed a bug that caused the slave to stop on statements that produced an error on the master. (Bug #8412 [http://bugs.mysql.com/8412])
- Documented problem with using `mysqldump` in 4.0.x to dump `TIMESTAMP(2)` and `TIMESTAMP(4)` column types. (Bug #6530 [http://bugs.mysql.com/6530])

D.3.3. Changes in release 4.0.23 (18 Dec 2004)

Note: Due to a **libtool**-related bug in the source distribution, the creation of shared `libmysqlclient` libraries was not possible (the resulting files were missing the `.so` file name extension). The file `lt-main.sh` was updated to fix this problem and the resulting source distribution was released as `mysql-4.0.23a.tar.gz`. This modification did not affect the binary packages. (Bug #7401 [http://bugs.mysql.com/7401])

Functionality added or changed:

- Added `--hex-blob` option to **mysqldump** for dumping binary string columns using hexadecimal notation.
- Added `mysql_hex_string()` C API function that hex-encodes a string.
- InnoDB: Do not periodically write `SHOW INNODB STATUS` information to a temporary file unless the configuration option `innodb_status_file=1` is set.
- InnoDB: Made the foreign key parser better aware of quotes. (Bug #6340 [http://bugs.mysql.com/6340])
- `mysqlbinlog` now prints an informative commented line (thread id, timestamp, server id, etc) before each `LOAD DATA INFILE`, like it does for other queries; unless `--short-form` is used.

Bugs fixed:

- Corrected accounts in the `mysql.user` table in Windows distributions that had been created with a `Host` value of `build` rather than `%`. (Bug #6000 [http://bugs.mysql.com/6000])
- Prevent adding `CREATE TABLE .. SELECT` query to the binary log when the insertion of new

records partially failed. (Bug #6682 [http://bugs.mysql.com/6682])

- Fixed bug which caused `FROM_UNIXTIME()` function to return wrong result if the argument was too big. (Bug #6439 [http://bugs.mysql.com/6439])
- Fixed bug which caused MySQL server to store wrong values in `TIMESTAMP` columns and give wrong results for `UNIX_TIMESTAMP()` function if it was run in time zone with leap seconds. (Bug #6387 [http://bugs.mysql.com/6387])
- InnoDB: Fixed a bug in `LOAD DATA INFILE…REPLACE` printing duplicate key error when executing the same load query several times. (Bug #5835 [http://bugs.mysql.com/5835])
- InnoDB: Refuse to open new-style tables created with MySQL 5.0.3 or later. (Bug #7089 [http://bugs.mysql.com/7089])
- InnoDB: Do not call `rewind()` when displaying `SHOW INNODB STATUS` information on `stderr`.
- InnoDB: If one used `INSERT IGNORE` to insert several rows at a time, and the first inserts were ignored because of a duplicate key collision, then InnoDB in a replication slave assigned `AUTO_INCREMENT` values 1 bigger than in the master. This broke the MySQL replication. (Bug #6287 [http://bugs.mysql.com/6287])
- InnoDB: Fix two hangs: `FOREIGN KEY` constraints treated table and database names as case-insensitive. `RENAME TABLE t TO T` would hang in an endless loop if `t` had a foreign key constraint defined on it. Fix also a hang over the dictionary mutex that would occur if one tried in `ALTER TABLE` or `RENAME TABLE` to create a foreign key constraint name that collided with another existing name. (Bug #3478 [http://bugs.mysql.com/3478])
- InnoDB: Treat character `0xA0` as space in InnoDB's `FOREIGN KEY` parser if MySQL treats it as space in the default charset. EMS MySQL Manager inserts character `0xA0` after the table name in an `ALTER`, which confused InnoDB's parser.
- Fixed a bug which caused a crash when only the slave I/O thread was stopped and restarted. (Bug #6148 [http://bugs.mysql.com/6148])
- If a connection had an open transaction but had done no updates to transactional tables (for example if had just done a `SELECT FOR UPDATE` then executed a non-transactional update, that update automatically committed the transaction (thus releasing InnoDB's row-level locks etc). (Bug #5714 [http://bugs.mysql.com/5714])
- If a connection was interrupted by a network error and did a rollback, the network error code got stored into the `BEGIN` and `ROLLBACK` binary log events; that caused superfluous slave stops. (Bug #6522 [http://bugs.mysql.com/6522])
- A sequence of `BEGIN` (or `SET AUTOCOMMIT=0`), `FLUSH TABLES WITH READ LOCK`, transactional update, `COMMIT`, `FLUSH TABLES WITH READ LOCK` could hang the connection forever and possibly the MySQL server itself. This happened for example when running the `innobackup` script several times. (Bug #6732 [http://bugs.mysql.com/6732])

D.3.4. Changes in release 4.0.22 (27 Oct 2004)

Functionality added or changed:

- The `--with-openssl` option for **configure** now accepts a path prefix as an argument. `--with-openssl-includes` and `--with-openssl-libs` are still supported, but are needed

only to override the default values. (Bug #5494 [http://bugs.mysql.com/5494])

- Added new `--without-man` option to **configure** to suppress building/installing the manual pages. (Bug #5379 [http://bugs.mysql.com/5379])

- InnoDB: New **mysqld** option `--innodb-table-locks` and session variable `innodb_table_locks` (on by default). In applications using `AUTOCOMMIT=1` and MySQL's `LOCK TABLES` command, InnoDB's internal table locks that were added in 4.0.20 can cause deadlocks. You can set `innodb_table_locks=0` in `my.cnf` to remove that problem. See Section 15.17, “Restrictions on `InnoDB` Tables”. (Bug #3299 [http://bugs.mysql.com/3299], Bug #5998 [http://bugs.mysql.com/5998])

- InnoDB: Added the startup option and settable global variable `innodb_max_purge_lag` for delaying `INSERT`, `UPDATE` and `DELETE` operations when the purge operations are lagging. The default value of this parameter is zero, meaning that there are no delays. See Section 15.13, “Implementation of Multi-Versioning”.

- InnoDB: Change error code to `HA_ERR_ROW_IS_REFERENCED` if we cannot `DROP` a parent table because it is referenced by a `FOREIGN KEY` constraint.

Bugs fixed:

- Fixed bug in server which caused connection stall when one of deprecated `libmysqlclient` functions `mysql_create_db(), mysql_rm_db()` were called and were going to return error. (Bug #6081 [http://bugs.mysql.com/6081])

- Fixed returning wrong query result from query cache if a temporary table was hiding a real table after putting results to query cache. (Bug #6084 [http://bugs.mysql.com/6084])

- Fixed `ENABLE KEYS`, which failed if `tmpdir` ran out of space. Now, a full repair is done in this case. (Bug #5625 [http://bugs.mysql.com/5625])

- Fixed an improper error message when trying to drop a table which is referenced by a `FOREIGN KEY` constraint. (Bug #5784 [http://bugs.mysql.com/5784])

- Fixed a bug that allowed `FLUSH TABLE(S)` to close `HANDLER` tables. `HANDLER` tables are now reopened after a `FLUSH TABLE(S)` the next time they are used. However, they lose their file position if this happens. (Bug #4286 [http://bugs.mysql.com/4286])

- Fixed a bug that allowed `HANDLER` tables with the same alias to be opened multiple times. `HANDLER` aliases must now be unique, even though it is syntactically correct in versions below 4.1, to qualify them with their base table's database name (e.g. test_db.handler_tbl, but this now conflicts with e.g. another_db.handler_tbl). (Bug #4335 [http://bugs.mysql.com/4335])

- Fixed crash when using MySQL 4.0 with privilege tables from MySQL 5.0.

- **mysqlimport** now reads input files locally from the client host only if the `--local` option is given. Previously, it assumed incorrectly in some cases that files were local even without `--local`. (Bug #5829 [http://bugs.mysql.com/5829])

- InnoDB: Make the check for excessive semaphore waits to tolerate glitches in the system clock (do not crash the server if the system time is adjusted while InnoDB is under load.). (Bug #5898 [http://bugs.mysql.com/5898])

- InnoDB: Fixed a bug in the InnoDB `FOREIGN KEY` parser that prevented `ALTER TABLE` of tables containing '#' in their names. (Bug #5856 [http://bugs.mysql.com/5856])

- InnoDB: Fixed problem introduced in 4.0.21 where a connection starting a transaction, doing updates, then `FLUSH TABLES WITH READ LOCK`, then `COMMIT`, would cause replication slaves to stop (complaining about error 1223). Bug surfaced when using the InnoDB `innobackup` script. (Bug #5949 [http://bugs.mysql.com/5949])

- InnoDB: If one updated a column so that its size changed, or updated it to an externally stored (`TEXT` or `BLOB`) value, then ANOTHER externally stored column would show up as 512 bytes of good data + 20 bytes of garbage in a consistent read that fetched the old version of the row. (Bug #5960 [http://bugs.mysql.com/5960])

- InnoDB: Release the dictionary latch during a long cascaded `FOREIGN KEY` operation, so that we do not starve other users doing `CREATE TABLE` or other DDL operations. This caused a notorious 'Long semaphore wait' message to be printed to the `.err` log. (Bug #5961 [http://bugs.mysql.com/5961])

- InnoDB: Let InnoDB remember row locking type (X or S) inside `LOCK TABLES`, also over plain consistent read `SELECT`s.

- InnoDB: Fixed a bug introduced in 4.0.21. An assertion failed if one used **mysqldump** with the option `-l` or `--opt`, or if one used `LOCK TABLES ... LOCAL`. (Workaround in 4.0.21: use `--quick` and `--single-transaction`. (Bug #5538 [http://bugs.mysql.com/5538])

- InnoDB: Having a column prefix index in the primary key, and the same column fully in a secondary key could cause an assertion failure in row_build_row_ref(). (Bug #5180 [http://bugs.mysql.com/5180])

- Fixed a bug which resulted in an erroneously calculated number of examined rows in `UNION`s. This value is printed in the slow query log. (Bug #5879 [http://bugs.mysql.com/5879])

- Fixed bug with crash of server on some values of `read_rnd_buffer_size` (Bug #5492 [http://bugs.mysql.com/5492])

- Fixed bug which caused truncation of values read from or into `TIMESTAMP` fields if `--new` mode was enabled. (Bug #4131 [http://bugs.mysql.com/4131])

- **mysqladmin** now returns a status of 0 even when the server denies access; such an error means the server is running. (Bug #3120 [http://bugs.mysql.com/3120])

- Fixed that if the slave SQL thread found a syntax error in a query (which should be rare, as the master parsed it successfully), it stops. (Bug #5711 [http://bugs.mysql.com/5711])

- Fixed that if a write to a MyISAM table fails because of a full disk or an exceeded disk quota, it prints a message to the error log every 10 minutes, and waits until disk becomes free. (Bug #3248 [http://bugs.mysql.com/3248])

- Fixed problem with symlinked databases on Windows being shown with `SHOW DATABASES` even if the database name doesn't match the given wildcard (Bug #5539 [http://bugs.mysql.com/5539])

D.3.5. Changes in release 4.0.21 (06 Sep 2004)

Functionality added or changed:

- Print *`version_comment`* (from `./configure --comment` during compilation) when starting the server. E.g.: `Version: '4.0.21-debug' socket: '/tmp/mysql.sock' port: 0 Official MySQL Binary`

- Made the MySQL server not react to signals `SIGHUP` and `SIGQUIT` on Mac OS X 10.3. This is needed because under this OS, the MySQL server receives lots of these signals (reported as Bug #2030 [http://bugs.mysql.com/2030]).
- On Windows, the **mysqld-nt** and **mysqld-max-nt** servers now write error messages to the Windows event log in addition to the MySQL error log.

Bugs fixed:

- Fixed an old bug in concurrent accesses to `MERGE` tables (even one `MERGE` table and `MyISAM` tables), that could've resulted in a crash or hang of the server. (Bug #2408 [http://bugs.mysql.com/2408])
- Fixed a bug that caused incorrect results from `GROUP BY` queries with expression in `HAVING` clause that refers to a `BLOB` (`TEXT`, `TINYBLOB`, etc) fields. (Bug #4358 [http://bugs.mysql.com/4358])
- Fixed a bug when memory was not released when `HEAP` table is dropped. It could only happen on Windows when a symlink file (.sym) is used and if that symlink file contained double backslashes (\\). (Bug #4973 [http://bugs.mysql.com/4973])
- Fixed a bug which prevented `TIMESTAMP(19)` fields from being created. (Bug #4491 [http://bugs.mysql.com/4491])
- Fixed a bug that caused wrong results in queries that were using index to search for `NULL` values in `BLOB` (`TINYBLOB`, `TEXT`, `TINYTEXT`, etc) columns of `MyISAM` tables. (Bug #4816 [http://bugs.mysql.com/4816])
- Fixed a bug in the function `ROUND()` reporting incorrect metadata (number of digits after the decimal point). It can be seen, for example, in `CREATE TABLE t1 SELECT ROUND(1, 34)`. (Bug #4393 [http://bugs.mysql.com/4393])
- Fixed precision loss bug in some mathematical functions such as `SQRT()` and `LOG()`. (Bug #4356 [http://bugs.mysql.com/4356])
- Fixed a long-standing problem with `LOAD DATA` with the `LOCAL` option. The problem occurs when an error happens during the `LOAD DATA` operation. Previously, the connection was broken. Now the error message is returned and connection stays open.
- Optimizer now treats `col IN (val)` the same way it does for `col = val`.
- Fixed a problem with `net_buffer_length` when building the `DBD::mysql` Perl module. (Bug #4206 [http://bugs.mysql.com/4206])
- `lower_case_table_names=2` (keep case for table names) was not honored with `ALTER TABLE` and `CREATE/DROP INDEX`. (Bug #3109 [http://bugs.mysql.com/3109])
- Fixed a crash on declaration of `DECIMAL(0,...)` column. (Bug #4046 [http://bugs.mysql.com/4046])
- Fixed a bug in `IF()` function incorrectly determining the result type if aggregate functions were involved. (Bug #3987 [http://bugs.mysql.com/3987])
- Fixed bug in privilege checking where, under some conditions, one was able to grant privileges on the database, he has no privileges on. (Bug #3933 [http://bugs.mysql.com/3933])
- Fixed crash in `MATCH ... AGAINST()` on a phrase search operator with a missing closing

double quote. (Bug #3870 [http://bugs.mysql.com/3870])

- Fixed a bug with truncation of big values (> 4294967295) of 64-bit system variables. (Bug #3754 [http://bugs.mysql.com/3754])
- If `server-id` was not set using startup options but with `SET GLOBAL`, the replication slave still complained that it was not set. (Bug #3829 [http://bugs.mysql.com/3829])
- Fixed potential memory overrun in `mysql_real_connect()` (which required a compromised DNS server and certain operating systems). (Bug #4017 [http://bugs.mysql.com/4017])
- During the installation process of the server RPM on Linux, **mysqld** was run as the `root` system user, and if you had `--log-bin=<somewhere_out_of_var_lib_mysql>` it created binary log files owned by `root` in this directory, which remained owned by `root` after the installation. This is now fixed by starting **mysqld** as the `mysql` system user instead. (Bug #4038 [http://bugs.mysql.com/4038])
- Made `DROP DATABASE` honor the value of `lower_case_table_names`. (Bug #4066 [http://bugs.mysql.com/4066])
- The slave SQL thread refused to replicate `INSERT ... SELECT` if it examined more than 4 billion rows. (Bug #3871 [http://bugs.mysql.com/3871])
- Fixed incorrect destruction of expression which led to crash of server on complex `AND`/`OR` expressions if query was ignored (either by a replication server because of `--replicate-*-table` rules, or by any MySQL server because of a syntax error). (Bug #3969 [http://bugs.mysql.com/3969], Bug #4494 [http://bugs.mysql.com/4494])
- Fixed that **mysqlbinlog --position --read-from-remote-server** had wrong `# at` lines. (Bug #4506 [http://bugs.mysql.com/4506])
- If `CREATE TEMPORARY TABLE t SELECT` failed while loading the data, the temporary table was not dropped. (Bug #4551 [http://bugs.mysql.com/4551])
- Fixed that when a multiple-table `DROP TABLE` failed to drop a table on the master server, the error code was not written to the binary log. (Bug #4553 [http://bugs.mysql.com/4553])
- When the slave SQL thread was replicating a `LOAD DATA INFILE` statement, it didn't show the statement in the output of `SHOW PROCESSLIST`. (Bug #4326 [http://bugs.mysql.com/4326])
- Fixed that `CREATE TABLE ... TYPE=HEAP ... AS SELECT...` caused replication slave to stop. (Bug #4971 [http://bugs.mysql.com/4971])
- Fixed that `disable-local-infile` option had no effect if client read it from a configuration file using `mysql_options(...,MYSQL_READ_DEFAULT,...)`. (Bug #5073 [http://bugs.mysql.com/5073])
- Fixed that **mysql-test-run** failed on the `rpl_trunc_binlog` test if running test from the installed (the target of 'make install') directory. (Bug #5050)
- Fixed an unlikely deadlock which could happen when using `KILL`. (Bug #4810)
- Fixed a crash when one connection got `KILL`ed while it was doing `START SLAVE`. (Bug #4827 [http://bugs.mysql.com/4827])
- Made `FLUSH TABLES WITH READ LOCK` block `COMMIT` if server is running with binary logging; this ensures that the binary log position is trustable when doing a full backup of tables and the binary log. (Bug #4953 [http://bugs.mysql.com/4953])

- Fixed that the counter of an `auto_increment` column was not reset by `TRUNCATE TABLE` is the table was a temporary one. (Bug #5033 [http://bugs.mysql.com/5033])
- Made database names to compare case-insensitively in fully qualified column names (`database.table.column`) when `lower_case_table_names=1`. (Bug #4792 [http://bugs.mysql.com/4792])
- Fixed that `SET CHARACTER SET` was not replicated correctly. MySQL 4.1 does not have that bug. (Bug #4500 [http://bugs.mysql.com/4500])

D.3.6. Changes in release 4.0.20 (17 May 2004)

Note: The windows packages had to be repackaged and re-released several times to resolve packaging issues (such as missing files). This did not affect the binaries included (they have not been recompiled), therefore the installation packages are of version 4.0.20d, while the binaries included still identify themselves as version 4.0.20b.

Functionality added or changed:

- From the Windows distribution, predefined accounts without passwords for remote users ("root@%", "@%") were removed (other distributions never had them).
- Phrase search in `MATCH ... AGAINST ( ... IN BOOLEAN MODE)` no longer matches partial words.

Bugs fixed:

- A crashing bug (race condition) was fixed in InnoDB diagnostic logging. It was introduced in 4.0.19. (Bug #3596 [http://bugs.mysql.com/3596])
- Fixed a bug in division `/` reporting incorrect metadata (number of digits after the decimal point). It can be seen, for example, in `CREATE TABLE t1 SELECT "0.01"/"3"`. (Bug #3612 [http://bugs.mysql.com/3612])
- Fixed a problem with non-working `DROP DATABASE` on some configurations (in particular, Linux 2.6.5 with ext3 are known to expose this bug). (Bug #3594 [http://bugs.mysql.com/3594])
- Fixed that in some replication error messages, a very long query caused the rest of the message to be invisible (truncated), by putting the query last in the message. (Bug #3357 [http://bugs.mysql.com/3357])

D.3.7. Changes in release 4.0.19 (04 May 2004)

Note: The MySQL 4.0.19 binaries were uploaded to the download mirrors on May, 10th. However, a potential crashing bug was found just before the 4.0.19 release was publicly announced and published from the 4.0 download pages at http://dev.mysql.com/.

A fix for the bug was pushed into the MySQL source tree shortly after it could be reproduced and is included in MySQL 4.0.20. Users upgrading from MySQL 4.0.18 should upgrade directly to MySQL 4.0.20 or later.

See (Bug #3596 [http://bugs.mysql.com/3596]) for details (it was reported against MySQL-4.1, but was confirmed to affect 4.0.19 as well).

Functionality added or changed:

- If length of a timestamp field is defined as 19, the timestamp is displayed as `"YYYY-MM-DD HH:MM:SS`. This is done to make it easier to use tables created in MySQL 4.1 to be used in MySQL 4.0.
- If you use `RAID_CHUNKS` with a value > 255 it is set to 255. This was made to ensure that all raid directories are always 2 hex bytes. (Bug #3182 [http://bugs.mysql.com/3182])
- Changed that the optimizer now considers the index specified in `FORCE INDEX` clause as a candidate to resolve `ORDER BY` as well.
- The `--log-warnings` server option now is enabled by default. Disable with `--skip-log-warnings`.
- Until now, in `SELECT ... UNION SELECT ... ORDER BY ...`, it was possible to qualify a column name in the `ORDER BY` clause with a table name. This is no longer possible. Column names in `ORDER BY` should refer to names established in the first `SELECT` of the `UNION`. (Bug #3064 [http://bugs.mysql.com/3064])
- Added `max_insert_delayed_threads` system variable as a synonym for `max_delayed_threads`.
- Added `query_cache_wlock_invalidate` system variable. It allows emulation of `MyISAM` table write-locking behavior, even for queries in the query cache. (Bug #2693 [http://bugs.mysql.com/2693])
- The keyword `MASTER_SERVER_ID` is not reserved anymore.
- The following is relevant mainly for Mac OS X users who use a case-insensitive filesystem. This is not relevant for Windows users as `InnoDB` in this case always stores filenames in lower case:

  You can now force `lower_case_table_names` to 0 from the command line or a configuration file. This is useful with case-insensitive filesystems when you have previously not used `lower_case_table_names=1` or `lower_case_table_names=2` and you have created `InnoDB` tables. With `lower_case_table_names=0`, `InnoDB` tables were stored in mixed case while setting lower_case_table_names to a non-zero value now forces it to lower case (to make the table names case insensitive).

  Because it's possible to crash `MyISAM` tables by referring to them with different case on a case-insensitive filesystem, we recommend that you use `lower_case_table_names` or `lower_case_table_names=2` on such filesystems.

  The easiest way to convert to use `lower_case_table_names=2` is to dump all your `InnoDB` tables with **mysqldump**, drop them and then restore them.
- Changed that the relay log is flushed to disk by the slave I/O thread every time it reads a relay log event. This reduces the risk of losing some part of the relay log in case of brutal crash.
- When a session having open temporary tables terminates, the statement automatically written to the binary log is now `DROP TEMPORARY TABLE IF EXISTS` instead of `DROP TEMPORARY TABLE`, for more robustness.
- Added option `--replicate-same-server-id`.

Bugs fixed:

- Added missing full-text variable `ft_stopword_file` to **myisamchk**.
- Don't allow stray `','` at the end of field specifications. (Bug #3481 [http://bugs.mysql.com/3481])
- `INTERVAL` now can handle big values for seconds, minutes and hours. (Bug #3498 [http://bugs.mysql.com/3498])
- Blank hostname did not work as documented for table and column privileges. Now it's works the same way as `'%'`. (Bug #3473 [http://bugs.mysql.com/3473])
- Fixed a harmless buffer overflow in `replace` utility. (Bug# 3541)
- Fixed `SOUNDEX()` to ignore non-alphabetic characters also in the beginning of the string. (Bug #3556 [http://bugs.mysql.com/3556])
- Fixed a bug in `MATCH ... AGAINST()` searches when another thread was doing concurrent inserts into the `MyISAM` table in question. The first --- full-text search --- query could return incorrect results in this case (for example, ``phantom" rows or not all matching rows, even an empty result set). The easiest way to check whether you are affected is to start **mysqld** with `--skip-concurrent-insert` switch and see if it helps.
- Fixed bug when doing `DROP DATABASE` on a directory containing non- MySQL files. Now a proper error message is returned.
- Fixed bug in `ANALYZE TABLE` on a `BDB` table inside a transaction that hangs server thread. (Bug #2342 [http://bugs.mysql.com/2342])
- Fixed a symlink vulnerability in the `mysqlbug` script. (Bug #3284 [http://bugs.mysql.com/3284])
- Fixed core dump bug in `SELECT DISTINCT` where all selected parts where constants and there were hidden columns in the created temporary table. (Bug #3203 [http://bugs.mysql.com/3203])
- Fixed core dump bug in `COUNT(DISTINCT)` when there was a lot of values and one had a big value for `max_heap_table_size`.
- Fixed problem with multiple-table-update and BDB tables. (Bug: #3098)
- Fixed memory leak when dropping database with `RAID` tables. (Bug #2882 [http://bugs.mysql.com/2882])
- Fixed core dump crash in replication during relay-log switch when the relay log went over `max_relay_log_size` and the slave thread did a `flush_io_cache()` at the same time.
- Fixed hangup bug when issuing multiple `SLAVE START` from different threads at the same time. (Bug #2921 [http://bugs.mysql.com/2921])
- Fixed bug when using `DROP DATABASE` with `lower_case_table_names=2`.
- Fixed wrong result in `UNION` when using `lower_case_table_names=2`. (Bug #2858 [http://bugs.mysql.com/2858])
- One can now kill threads that is 'stuck' in the join optimizer (can happen when there is MANY tables in the join in which case the optimizer can take really long time). (Bug #2825 [http://bugs.mysql.com/2825])
- Rollback `DELETE` and `UPDATE` statements if thread is killed. (Bug #2422 [http://bugs.mysql.com/2422])
- Ensure that all rows in an `INSERT DELAYED` statement is written at once if binary logging is en-

abled. (Bug #2491 [http://bugs.mysql.com/2491]).

- Fixed bug in query cache statistic, more accurate formula linked statistic variables mentioned in the manual.
- Fixed a bug in parallel repair (**myisamchk -p**, `myisam_repair_threads`) - sometimes repair process failed to repair a table. (Bug #1334 [http://bugs.mysql.com/1334])
- Fixed bugs with names of tables, databases and columns that end to space (Bug #2985 [http://bugs.mysql.com/2985])
- Fixed a bug in multiple-table `UPDATE` statements involving at least one constant table. Bug was exhibited in allowing non matching row to be updated. (Bug #2996 [http://bugs.mysql.com/2996]).
- Fixed all bugs in scripts for creating/upgrading system database (Bug #2874 [http://bugs.mysql.com/2874]) Added tests which guarantee against such bugs in the future.
- Fixed bug in `mysql` command-line client in interpreting quotes within comments. (Bug #539 [http://bugs.mysql.com/539])
- `--set-character-set` and `--character-sets-dir` options in **myisamchk** now work.
- Fixed a bug in **mysqlbinlog** that caused one pointer to be free'd twice in some cases.
- Fixed a bug in boolean full-text search, that sometimes could lead to false matches in queries with several levels of subexpressions using `+` operator (for example, `MATCH ... AGAINST('+(+(word1 word2)) +word3*' IN BOOLEAN MODE)`.
- Fixed Windows-specific portability bugs in `myisam_ftdump`.
- Fixed a bug in multiple-table `DELETE` that was caused by foreign key constraints. If the order of the tables established by MySQL optimizer did not match parent-child order, no rows were deleted and no error message was provided. (Bug #2799 [http://bugs.mysql.com/2799])
- Fixed a few years old bug in the range optimizer that caused a segmentation fault on some very rare queries. (Bug #2698 [http://bugs.mysql.com/2698])
- Replication: If a client connects to a slave server and issues an administrative statement for a table (for example, `OPTIMIZE TABLE` or `REPAIR TABLE`), this could sometimes stop the slave SQL thread. This does not lead to any corruption, but you must use `START SLAVE` to get replication going again. (Bug #1858 [http://bugs.mysql.com/1858]) The bug was accidentally not fixed in 4.0.17 as it was unfortunately earlier said.
- Fixed that when a `Rotate` event is found by the slave SQL thread in the middle of a transaction, the value of `Relay_Log_Pos` in `SHOW SLAVE STATUS` remains correct. (Bug #3017 [http://bugs.mysql.com/3017])
- Corrected the master's binary log position that `InnoDB` reports when it is doing a crash recovery on a slave server. (Bug #3015 [http://bugs.mysql.com/3015])
- Changed that when a `DROP TEMPORARY TABLE` statement is automatically written to the binary log when a session ends, the statement is recorded with an error code of value zero (this ensures that killing a `SELECT` on the master does not result in a superfluous error on the slave). (Bug #3063 [http://bugs.mysql.com/3063])
- Changed that when a thread handling `INSERT DELAYED` (also known as a `delayed_insert` thread) is killed, its statements are recorded with an error code of value zero (killing such a thread does not endanger replication, so we thus avoid a superfluous error on the slave). (Bug #3081 [http://bugs.mysql.com/3081])

- Fixed deadlock when two `START SLAVE` commands were run at the same time. (Bug #2921 [http://bugs.mysql.com/2921])
- Fixed that a statement never triggers a superfluous error on the slave, if it must be excluded given the `--replicate-*` options. The bug was that if the statement had been killed on the master, the slave would stop. (Bug #2983 [http://bugs.mysql.com/2983])
- The `--local-load` option of **mysqlbinlog** now requires an argument.
- Fixed a segmentation fault when running `LOAD DATA FROM MASTER` after `RESET SLAVE`. (Bug #2922 [http://bugs.mysql.com/2922])
- Fixed a rare error condition that caused the slave SQL thread spuriously to print the message `Binlog has bad magic number` and stop when it was not necessary to do so. (Bug #3401 [http://bugs.mysql.com/3401])
- Fixed the column `Exec_master_log_pos` (and its disk image in the `relay-log.info` file) to be correct if the master had version 3.23 (it was too big by 6 bytes). This bug does not exist in the 5.0 version. (Bug #3400 [http://bugs.mysql.com/3400])
- Fixed that **mysqlbinlog** does not forget to print a `USE` command under rare circumstances where the binary log contained a `LOAD DATA INFILE` command. (Bug #3415 [http://bugs.mysql.com/3415])
- Fixed a memory corruption when replicating a `LOAD DATA INFILE` when the master had version 3.23. Some smaller problems remain in this setup, See Section 6.7, “Replication Features and Known Problems”. (Bug #3422 [http://bugs.mysql.com/3422])
- Multiple-table `DELETE` statements were always replicated by the slave if there were some `--replicate-*-ignore-table` options and no `--replicate-*-do-table` options. (Bug #3461 [http://bugs.mysql.com/3461])
- Fixed a crash of the MySQL slave server when it was built with `--with-debug` and replicating itself. (Bug #3568 [http://bugs.mysql.com/3568])

D.3.8. Changes in release 4.0.18 (12 Feb 2004)

Functionality added or changed:

- Fixed processing of `LOAD DATA` by **mysqlbinlog** in remote mode. (Bug #1378 [http://bugs.mysql.com/1378])
- The **ft_dump** utility program was renamed to **myisam_ftdump**, and is included in binary distributions.
- `ENGINE` is now a synonym for the `TYPE` option for `CREATE TABLE` and `ALTER TABLE`.
- `lower_case_table_names` system variable now can take a value of `2`, to store table names in mixed case on case-insensitive filesystems. It's forced to 2 if the database directory is located on a case-insensitive filesystem.
- For replication of `MEMORY` (`HEAP`) tables: Made the master automatically write a `DELETE FROM` statement to its binary log when a `MEMORY` table is opened for the first time since master's startup. This is for the case where the slave has replicated a non-empty `MEMORY` table, then the master is shut down and restarted: the table is now empty on master; the `DELETE FROM` empties it on slave too. Note that even with this fix, between the master's restart and the first use of the table on master,

the slave still has out-of-date data in the table. But if you use the `init-file` option to populate the `MEMORY` table on the master at startup, it ensures that the failing time interval is zero. (Bug #2477 [http://bugs.mysql.com/2477])

- Optimizer is now better tuned for the case where the first used key part (of many) is a constant. (Bug #1679 [http://bugs.mysql.com/1679])
- Removed old non-working `--old-rpl-compat` server option, which was a holdover from the very first 4.0.x versions. (Bug #2428 [http://bugs.mysql.com/2428])
- Added option `--sync-frm`. It's on by default, to instruct MySQL to sync to disk each time `.frm` file is created. Use `--disable-sync-frm` to disable.

Bugs fixed:

- **mysqlhotcopy** now works on NetWare.
- `DROP DATABASE` could not drop databases with RAID tables that had more than nine `RAID_CHUNKS`. (Bug #2627 [http://bugs.mysql.com/2627])
- Fixed bug in range optimizer when using overlapping ranges. (Bug #2448 [http://bugs.mysql.com/2448])
- Limit `wait_timeout` to 2147483 on Windows (OS limit). (Bug #2400 [http://bugs.mysql.com/2400])
- Fixed bug when `--init-file` crashes MySQL if it contains a large `SELECT`. (Bug #2526 [http://bugs.mysql.com/2526])
- `SHOW KEYS` now shows `NULL` in the `Sub_part` column for `FULLTEXT` indexes.
- The signal thread's stack size was increased to enable **mysqld** to run on Debian/IA-64 with a TLS-enabled `glibc`. (Bug #2599 [http://bugs.mysql.com/2599])
- Now only the `SELECT` privilege is needed for tables that are only read in multiple-table `UPDATE` statements. (Bug #2377 [http://bugs.mysql.com/2377])
- Give proper error message if one uses `LOCK TABLES ... ; INSERT ... SELECT` and one used the same table in the `INSERT` and `SELECT` part. (Bug #2296 [http://bugs.mysql.com/2296])
- `SELECT INTO ... DUMPFILE` now deletes the generated file on error.
- Fixed foreign key reference handling to allow references to column names that contain spaces. (Bug #1725 [http://bugs.mysql.com/1725])
- Fixed problem with index reads on character columns with `BDB` tables. The symptom was that data could be returned in the wrong lettercase. (Bug #2509 [http://bugs.mysql.com/2509])
- Fixed a spurious table corruption problem that could sometimes appear on tables with indexed `TEXT` columns if these columns happened to contain values having trailing spaces. This bug was introduced in 4.0.17.
- Fixed a problem where some queries could hang if a condition like *`indexed_TEXT_column = expr`* was present and the column contained values having trailing spaces. This bug was introduced in 4.0.17.
- Fixed a bug that could cause incorrect results from a query that involved range conditions on in-

dexed TEXT columns that happened to contain values having trailing spaces. This bug was introduced in 4.0.17. (Bug #2295 [http://bugs.mysql.com/2295])

- Fixed incorrect path names in some of the manual pages. (Bug #2270 [http://bugs.mysql.com/2270])
- Fixed spurious ``table corrupted'' errors in parallel repair operations. See Section 5.2.3, “Server System Variables”.
- Fixed a crashing bug in parallel repair operations. See Section 5.2.3, “Server System Variables”.
- Fixed bug in updating MyISAM tables for BLOB values longer than 16MB. (Bug #2159 [http://bugs.mysql.com/2159])
- Fixed bug in **mysqld_safe** when running multiple instances of MySQL. (Bug #2114 [http://bugs.mysql.com/2114])
- Fixed a bug in using HANDLER statement with tables not from a current database. (Bug #2304 [http://bugs.mysql.com/2304])
- Fixed a crashing bug that occurred due to the fact that multiple-table UPDATE statements did not check that there was only one table to be updated. (Bug #2103 [http://bugs.mysql.com/2103])
- Fixed a crashing bug that occurred due to BLOB column type index size being calculated incorrectly in MIN() and MAX() optimizations. (Bug #2189 [http://bugs.mysql.com/2189])
- Fixed a bug with incorrect syntax for LOCK TABLES in **mysqldump**. (Bug #2242 [http://bugs.mysql.com/2242])
- Fixed a bug in **mysqld_safe** that caused **mysqld** to generate a warning about duplicate user=*xxx* options if this option was specified in the [mysqld] or [server] sections of my.cnf. (Bug #2163 [http://bugs.mysql.com/2163])
- INSERT DELAYED ... SELECT ... could cause table corruption because tables were not locked properly. This is now fixed by ignoring DELAYED in this context. (Bug #1983 [http://bugs.mysql.com/1983])
- Replication: Sometimes the master gets a non-fatal error during the execution of a statement that does not immediately succeed. (For example, a write to a MyISAM table may first receive ``no space left on device,'' but later complete when disk space becomes available. See Section A.4.3, “How MySQL Handles a Full Disk”.) The bug was that the master forgot to reset the error code to 0 after success, so the error code got into its binary log, thus causing the slave to issue false alarms such as ``did not get the same error as on master.'' (Bug #2083 [http://bugs.mysql.com/2083])
- Removed a misleading ``check permissions on master.info'' from a replication error message, because the cause of the problem could be something other than permissions. (Bug #2121 [http://bugs.mysql.com/2121])
- Fixed a crash when the replication slave was unable to create the first relay log. (Bug #2145 [http://bugs.mysql.com/2145])
- Replication of LOAD DATA INFILE for an empty file from a 3.23 master to a 4.0 slave caused the slave to print an error. (Bug #2452 [http://bugs.mysql.com/2452])
- When automatically forcing lower_case_table_names to 1 if the file system was case insensitive, **mysqld** could crash. This bug existed only in MySQL 4.0.17. (Bug #2481 [http://bugs.mysql.com/2481])
- Restored ability to specify default values for TIMESTAMP columns that was erroneously disabled in previous release. (Bug #2539 [http://bugs.mysql.com/2539]) Fixed SHOW CREATE TABLE to re-

flect these values. (Bug #1885 [http://bugs.mysql.com/1885]) Note that because of the auto-update feature for the first `TIMESTAMP` column in a table, it makes no sense to specify a default value for the column. Any such default is silently ignored (unless another `TIMESTAMP` column is added before this one). Also fixed the meaning of the `DEFAULT` keyword when it is used to specify the value to be inserted into a `TIMESTAMP` column other than the first. (Bug #2464 [http://bugs.mysql.com/2464])

- Fixed bug for out-of-range arguments on QNX platform that caused `UNIX_TIMESTAMP()` to produce incorrect results or that caused non-zero values to be inserted into `TIMESTAMP` columns. (Bug #2523 [http://bugs.mysql.com/2523]) Also, current time zone now is taken into account when checking if datetime values satisfy both range boundaries for `TIMESTAMP` columns. The range allowed for a `TIMESTAMP` column is time zone-dependent and equivalent to a range of `1970-01-01 00:00:01` UTC to `2037-12-31 23:59:59` UTC.
- Multiple-table `DELETE` statements were never replicated by the slave if there were any `--replicate-*-table` options. (Bug #2527 [http://bugs.mysql.com/2527])
- Changes to session counterparts of variables `query_prealloc_size`, `query_alloc_block_size`, `trans_prealloc_size`, `trans_alloc_block_size` now have an effect. (Bug #1948 [http://bugs.mysql.com/1948])
- Fixed bug in `ALTER TABLE RENAME`, when rename to the table with the same name in another database silently dropped destination table if it existed. (Bug #2628 [http://bugs.mysql.com/2628])

D.3.9. Changes in release 4.0.17 (14 Dec 2003)

Functionality added or changed:

- **mysqldump** no longer dumps data for `MERGE` tables. (Bug #1846 [http://bugs.mysql.com/1846])
- `lower_case_table_names` is now forced to 1 if the database directory is located on a case-insensitive filesystem. (Bug #1812 [http://bugs.mysql.com/1812])
- Symlink creation is now disabled on systems where `realpath()` doesn't work. (Before one could use `CREATE TABLE .. DATA DIRECTORY=..` even if `HAVE_BROKEN_REALPATH` was defined. This is now disabled to avoid problems when running `ALTER TABLE`).
- Inserting a negative `AUTO_INCREMENT` value in a `MyISAM` table no longer updates the `AUTO_INCREMENT` counter to a big unsigned value. (Bug #1366 [http://bugs.mysql.com/1366])
- Added four new modes to `WEEK(..., mode)` function. See `WEEK(date: (mode))`. (Bug #1178 [http://bugs.mysql.com/1178])
- Allow `UNION DISTINCT` syntax.
- MySQL now syncs to disk each time `.frm` file is created.
- `mysql_server_init()` now returns 1 if it can't initialize the environment. (Previously `mysql_server_init()` called `exit(1)` if it could not create a key with `pthread_key_create()`. (Bug #2062 [http://bugs.mysql.com/2062])
- Allow spaces in Windows service names.
- Changed the default Windows service name for **mysqld** from `MySql` to `MySQL`. This should not affect usage, because service names are not case sensitive.

- When you install **mysqld** as a service on Windows systems, **mysqld** reads startup options in option files from the option group with the same name as the service name. (Except when the service name is `MySQL`).

Bugs fixed:

- Sending `SIGHUP` to **mysqld** crashed the server if it was running with `--log-bin`. (Bug #2045 [http://bugs.mysql.com/2045])
- One can now configure MySQL as a Windows service as a normal user. (Bug #1802 [http://bugs.mysql.com/1802]). Thanks to Richard Hansen for fixing this.
- Database names are now compared in lowercase in `ON` clauses when `lower_case_table_names` is set. (Bug #1736 [http://bugs.mysql.com/1736])
- `IGNORE ... LINES` option to `LOAD DATA INFILE` didn't work when used with fixed length rows. (Bug #1704 [http://bugs.mysql.com/1704])
- Fixed problem with `UNIX_TIMESTAMP()` for timestamps close to 0. (Bug #1998 [http://bugs.mysql.com/1998])
- Fixed problem with character values greater than 128 in the `QUOTE()` function. (Bug #1868 [http://bugs.mysql.com/1868])
- Fixed searching of `TEXT` with endspace. (Bug #1651 [http://bugs.mysql.com/1651])
- Fixed caching bug in multiple-table updates where same table was used twice. (Bug #1711 [http://bugs.mysql.com/1711])
- Fixed directory permissions for the MySQL-server RPM documentation directory. (Bug #1672 [http://bugs.mysql.com/1672])
- Fixed server crash when updating an `ENUM` column that is set to the empty string (for example, with `REPLACE()`). (Bug #2023 [http://bugs.mysql.com/2023])
- **mysql** client program now correctly prints connection identifier returned by `mysql_thread_id()` as unsigned integer rather than as signed integer. (Bug #1951 [http://bugs.mysql.com/1951])
- `FOUND_ROWS()` could return incorrect number of rows after a query with an impossible `WHERE` condition. (Bug #1468 [http://bugs.mysql.com/1468])
- `SHOW DATABASES` no longer shows `.sym` files (on Windows) that do not point to a valid directory. (Bug #1385 [http://bugs.mysql.com/1385])
- Fixed a possible memory leak on Mac OS X when using the shared `libmysql.so` library. (from `pthread_key_create()`). (Bug #2061 [http://bugs.mysql.com/2061])
- Fixed bug in `UNION` statement with alias `*`. (Bug #1249 [http://bugs.mysql.com/1249])
- Fixed a bug in `DELETE ... ORDER BY ... LIMIT` where the rows where not deleted in the proper order. (Bug #1024 [http://bugs.mysql.com/1024], Bug #1697 [http://bugs.mysql.com/1697]).
- Fixed serious problem with multi-threaded programs on Windows that used the embedded MySQL libraries. (Locks of tables were not handled correctly between different threads).
- Code cleanup: Fixed a few code defects (potential memory leaks, null pointer dereferences, unini-

tialized variables). Thanks to Reasoning Inc. for informing us about these findings.

- Fixed a buffer overflow error that occurred with prepended '0' characters in some columns of type DECIMAL. (Bug #2128 [http://bugs.mysql.com/2128])
- Filesort was never shown in EXPLAIN if query contained an ORDER BY NULL clause. (Bug #1335 [http://bugs.mysql.com/1335])
- Fixed invalidation of whole query cache on DROP DATABASE. (Bug #1898 [http://bugs.mysql.com/1898])
- Fixed bug in range optimizer that caused wrong results for some unlikely AND/OR queries. (Bug #1828 [http://bugs.mysql.com/1828])
- Fixed a crash in ORDER BY when ordering by expression and identifier. (Bug #1945 [http://bugs.mysql.com/1945])
- Fixed a crash in an open HANDLER when an ALTER TABLE was executed in a different connection. (Bug #1826 [http://bugs.mysql.com/1826])
- Fixed a bug in trunc* operator of full-text search which sometimes caused MySQL not to find all matched rows.
- Fixed bug in prepending '0' characters to DECIMAL column values.
- Fixed optimizer bug, introduced in 4.0.16, when REF access plan was preferred to more efficient RANGE on another column.
- Fixed problem when installing a MySQL server as a Windows service using a command of the form **mysqld --install mysql --defaults-file=*path-to-file*.** (Bug #1643 [http://bugs.mysql.com/1643])
- Fixed an incorrect result from a query that uses only const tables (such as one-row tables) and non-constant expression (such as RAND()). (Bug #1271 [http://bugs.mysql.com/1271])
- Fixed bug when the optimizer did not take SQL_CALC_FOUND_ROWS into account if LIMIT clause was present. (Bug #1274 [http://bugs.mysql.com/1274])
- **mysqlbinlog** now asks for a password at the console when the -p or --password option is used with no argument. This is consistent with the way that other clients such **mysqladmin** and **mysqldump** behave. **Note**: A consequence of this change is that it is no longer possible to invoke **mysqlbinlog** as **mysqlbinlog -p pass_val** (with a space between the -p option and the following password value). (Bug #1595 [http://bugs.mysql.com/1595])
- Fixed bug accidentally introduced in 4.0.16 where the slave SQL thread deleted its replicated temporary tables when STOP SLAVE was issued.
- In a ``chain'' replication setup A->B->C, if 2 sessions on A updated temporary tables of the same name at the same time, the binary log of B became incorrect, resulting in C becoming confused. (Bug #1686 [http://bugs.mysql.com/1686])
- In a ``chain'' replication setup A->B->C, if STOP SLAVE was issued on B while it was replicating a temporary table from A, then when START SLAVE was issued on B, the binary log of B became incorrect, resulting in C becoming confused. (Bug #1240 [http://bugs.mysql.com/1240])
- When MASTER_LOG_FILE and MASTER_LOG_POS were not specified, CHANGE MASTER used the coordinates of the slave I/O thread to set up replication, which broke replication if the slave SQL thread lagged behind the slave I/O thread. This caused the slave SQL thread to lose some events. The new behavior is to use the coordinates of the slave SQL thread instead. See Section 13.6.2.1,

CHANGE MASTER TO Syntax". (Bug #1870 [http://bugs.mysql.com/1870])

- Now if integer is stored or converted to TIMESTAMP or DATETIME value checks of year, month, day, hour, minute and second ranges are performed and numbers representing illegal timestamps are converted to 0 value. This behavior is consistent with manual and with behavior of string to TIMESTAMP/DATETIME conversion. (Bug #1448 [http://bugs.mysql.com/1448])

- Fixed bug when BIT_AND() and BIT_OR() group functions returned incorrect value if SELECT used a temporary table and no rows were found. (Bug #1790 [http://bugs.mysql.com/1790]).

- BIT_AND() is now unsigned in all contexts. This means that it now returns 18446744073709551615 (= 0xffffffffffffffff) instead of -1 if there were no rows in the result.

- Fixed bug with BIT_AND() still returning signed value for an empty set in some cases. (Bug #1972 [http://bugs.mysql.com/1972])

- Fixed bug with ^ (XOR) and >> (bit shift) still returning signed value in some cases. (Bug #1993 [http://bugs.mysql.com/1993])

- Replication: a rare race condition in the slave SQL thread, which could lead to a wrong complain that the relay log is corrupted. (Bug #2011 [http://bugs.mysql.com/2011])

- Replication: in the slave SQL thread, a multiple-table UPDATE could produce a wrong complain that some record was not found in one table, if the UPDATE was preceded by a INSERT ... SELECT. (Bug #1701 [http://bugs.mysql.com/1701])

- Fixed deficiency in MySQL code which is responsible for scanning directories. This deficiency caused SHOW TABLE STATUS to be very slow when a database contained a large number of tables, even if a single particular table were specified. (Bug #1952 [http://bugs.mysql.com/1952])

D.3.10. Changes in release 4.0.16 (17 Oct 2003)

Functionality added or changed:

- Option values in option files now may be quoted. This is useful for values that contain whitespace or comment characters.

- Write memory allocation information to error log when doing **mysqladmin debug**. This works only on systems that support the mallinfo() call (like newer Linux systems).

- Added the following new system variables to allow more precise memory allocation: range_alloc_block_size, query_alloc_block_size, query_prealloc_size, transaction_alloc_block_size, and transaction_prealloc_size.

- **mysqlbinlog** now reads option files. To make this work, you must now specify --read-from-remote-server when reading binary logs from a MySQL server. (Note that using a remote server is deprecated and may disappear in future **mysqlbinlog** versions).

- Block SIGPIPE signals also for non-threaded programs. The blocking is moved from mysql_init() to mysql_server_init(), which is automatically called on the first call to mysql_init().

- Added --libs_r and --include options to **mysql_config**.

- New `> prompt for **mysql**. This prompt is similar to the '> and "> prompts, but indicates that an identifier quoted with backticks was begun on an earlier line and the closing backtick has not yet

been seen.

- Updated **mysql_install_db** to be able to use the local machine's IP address instead of the hostname when building the initial grant tables if `skip-name-resolve` has been specified. This option can be helpful on FreeBSD to avoid thread-safety problems with the FreeBSD resolver libraries. (Thanks to Jeremy Zawodny for the patch.)

- A documentation change: Added a note that when backing up a slave, it is necessary also to back up the `master.info` and `relay-log.info` files, as well as any `SQL_LOAD-*` files located in the directory specified by the `--slave-load-tmpdir` option. All these files are needed when the slave resumes replication after you restore the slave's data.

Bugs fixed:

- Fixed a spurious error `ERROR 14: Can't change size of file (Errcode: 2)` on Windows in `DELETE FROM` *`tbl_name`* without a `WHERE` clause or `TRUNCATE TABLE` *`tbl_name`*, when *`tbl_name`* is a `MyISAM` table. (Bug #1397 [http://bugs.mysql.com/1397])

- Fixed a bug that resulted in `thr_alarm queue is full` warnings after increasing the `max_connections` variable with `SET GLOBAL`. (Bug #1435 [http://bugs.mysql.com/1435])

- Made `LOCK TABLES` to work when `Lock_tables_priv` is granted on the database level and `Select_priv` is granted on the table level.

- Fixed crash of `FLUSH QUERY CACHE` on queries that use same table several times (Bug #988 [http://bugs.mysql.com/988]).

- Fixed core dump bug when setting an enum system variable (such as `SQL_WARNINGS`) to `NULL`.

- Extended the default timeout value for Windows clients from 30 seconds to 1 year. (The timeout that was added in MySQL 4.0.15 was way too short). This fixes a bug that caused `ERROR 2013: Lost connection to MySQL server during query` for queries that lasted longer than 30 seconds, if the client didn't specify a limit with `mysql_options()`. Users of 4.0.15 on Windows should upgrade to avoid this problem.

- More ``out of memory'' checking in range optimizer.

- Fixed and documented a problem when setting and using a user variable within the same `SELECT` statement. (Bug #1194 [http://bugs.mysql.com/1194]).

- Fixed bug in overrun check for `BLOB` values with compressed tables. This was a bug introduced in 4.0.14. It caused MySQL to regard some correct tables containing `BLOB` values as corrupted. (Bug #770 [http://bugs.mysql.com/770], Bug #1304 [http://bugs.mysql.com/1304], and maybe Bug #1295 [http://bugs.mysql.com/1295])

- `SHOW GRANTS` showed `USAGE` instead of the real column-level privileges when no table-level privileges were given.

- When copying a database from the master, `LOAD DATA FROM MASTER` dropped the corresponding database on the slave, thus erroneously dropping tables that had no counterpart on the master and tables that may have been excluded from replication using `--replicate-*-table` rules. Now `LOAD DATA FROM MASTER` no longer drops the database. Instead, it drops only the tables that have a counterpart on the master and that match the `--replicate-*-table` rules. `--replicate-*-db` rules can still be used to include or exclude a database as a whole from `LOAD DATA FROM MASTER`. A database also is included or excluded as a whole if there are some rules like `--replicate-wild-do-table=db1.%` or `--replicate-wild-ignore-table=db1.%`, as is the case for `CREATE DATABASE` and

DROP DATABASE in replication. (Bug #1248)

- Fixed a bug where **mysqlbinlog** crashed with a segmentation fault when used with the -h or --host option. (Bug #1258 [http://bugs.mysql.com/1258])
- Fixed a bug where **mysqlbinlog** crashed with a segmentation fault when used on a binary log containing only final events for LOAD DATA. (Bug #1340 [http://bugs.mysql.com/1340])
- **mysqlbinlog** does not reuse temporary filenames from previous runs. Previously **mysqlbinlog** failed if was used several times on the same binary log file that contained a LOAD DATA command.
- Fixed compilation problem when compiling with OpenSSL 0.9.7 with disabled old DES support (If OPENSSL_DISABLE_OLD_DES_SUPPORT option was enabled).
- Fixed a bug when two (or more) MySQL servers were running on the same machine, and they were both slaves, and at least one of them was replicating some LOAD DATA INFILE command from its master. The bug was that one slave MySQL server sometimes deleted the SQL_LOAD-* files (used for replication of LOAD DATA INFILE and located in the slave-load-tmpdir directory, which defaults to tmpdir) belonging to the other slave MySQL server of this machine, if these slaves had the same slave-load-tmpdir directory. When that happened, the other slave could not replicate LOAD DATA INFILE and complained about not being able to open some SQL_LOAD-* file. (Bug #1357 [http://bugs.mysql.com/1357])
- If LOAD DATA INFILE failed for a small file, the master forgot to write a marker (a Delete_file event) in its binary log, so the slave could not delete 2 files (SQL_LOAD-*.info and SQL_LOAD-*.data from its tmpdir. (Bug #1391 [http://bugs.mysql.com/1391])
- On Windows, the slave forgot to delete a SQL_LOAD-*.info file from tmpdir after successfully replicating a LOAD DATA INFILE command. (Bug #1392 [http://bugs.mysql.com/1392])
- When a connection terminates, MySQL writes DROP TEMPORARY TABLE statements to the binary log for all temporary tables which the connection had not explicitly dropped. MySQL forgot to use backticks to quote the database and table names in the statement. (Bug #1345 [http://bugs.mysql.com/1345])
- On some 64-bit machines (some HP-UX and Solaris machines), a slave installed with the 64-bit MySQL binary could not connect to its master (it connected to itself instead). (Bug #1256 [http://bugs.mysql.com/1256], Bug #1381 [http://bugs.mysql.com/1381])
- Code was introduced in MySQL 4.0.15 for the slave to detect that the master had died while writing a transaction to its binary log. This code reported an error in a legal situation: When the slave I/O thread was stopped while copying a transaction to the relay log, the slave SQL thread would later pretend that it found an unfinished transaction. (Bug #1475 [http://bugs.mysql.com/1475])

D.3.11. Changes in release 4.0.15 (03 Sep 2003)

IMPORTANT:

If you are using this release on Windows, you should upgrade at least your clients (any program that uses libmysql.lib) to 4.0.16 or above. This is because the 4.0.15 release had a bug in the Windows client library that causes Windows clients using the library to die with a Lost connection to MySQL server during query error for queries that take more than 30 seconds. This problem is specific to Windows; clients on other platforms are unaffected.

Functionality added or changed:

- **mysqldump** now correctly quotes all identifiers when communicating with the server. This assures that during the dump process, **mysqldump** never sends queries to the server that result in a syntax error. This problem is **not** related to the **mysqldump** program's output, which was not changed. (Bug #1148 [http://bugs.mysql.com/1148])

- Change result set metadata information so that `MIN()` and `MAX()` report that they can return `NULL` (this is true because an empty set returns `NULL`). (Bug #324 [http://bugs.mysql.com/324])

- Produce an error message on Windows if a second **mysqld** server is started on the same TCP/IP port as a running **mysqld** server.

- The **mysqld** system variables `wait_timeout`, `net_read_timeout`, and `net_write_timeout` now work on Windows. One can now also set timeouts for read and writes in Windows clients with `mysql_options()`.

- Added option `--sql-mode=NO_DIR_IN_CREATE` to make it possible for slaves to ignore `INDEX DIRECTORY` and `DATA DIRECTORY` options given to `CREATE TABLE`. When this is mode is on, `SHOW CREATE TABLE` does not show the given directories.

- `SHOW CREATE TABLE` now shows the `INDEX DIRECTORY` and `DATA DIRECTORY` options, if they were specified when the table was created.

- The `open_files_limit` system variable now shows the real open files limit.

- `MATCH ... AGAINST()` in natural language mode now treats words that are present in more than 2,000,000 rows as stopwords.

- The Mac OS X installation disk images now include an additional `MySQLStartupItem.pkg` package that enables the automatic startup of MySQL on system startup. See Section 2.5, “Installing MySQL on Mac OS X”.

- Most of the documentation included in the binary tarball distributions (`.tar.gz`) has been moved into a subdirectory `docs`. See Section 2.1.5, “Installation Layouts”.

- The manual is now included as an additional `info` file in the binary distributions. (Bug #1019 [http://bugs.mysql.com/1019])

- The binary distributions now include the embedded server library (`libmysqld.a`) by default. Due to a linking problem with non-gcc compilers, it was not included in all packages of the initial 4.0.15 release. The affected packages were rebuilt and released as 4.0.15a. See Section 1.3.1.2, “The Embedded MySQL Server”.

- MySQL can now use range optimization for `BETWEEN` with non-constant limits. (Bug #991 [http://bugs.mysql.com/991])

- Replication error messages now include the default database, so that users can check which database the failing query was run for.

- A documentation change: Added a paragraph about how the `binlog-do-db` and `binlog-ignore-db` options are tested against the database on the master (see Section 5.9.4, “The Binary Log”), and a paragraph about how `--replicate-do-db`, `--replicate-do-table` and analogous options are tested against the database and tables on the slave (see Section 6.8, “Replication Startup Options”).

- Now the slave does not replicate `SET PASSWORD` if it is configured to exclude the `mysql` database from replication (using for example `--replicate-wild-ignore-table=mysql.%`). This was the case for `GRANT` and `REVOKE` since version 4.0.13 (although there was Bug #980 [http://bugs.mysql.com/980] in 4.0.13 & 4.0.14, which has been fixed in 4.0.15).

- Rewrote the information shown in the `State` column of `SHOW PROCESSLIST` for replication threads and for `MASTER_POS_WAIT()` and added the most common states for these threads to the documentation, see Section 6.3, “Replication Implementation Details”.
- Added a test in replication to detect the case where the master died in the middle of writing a transaction to the binary log; such unfinished transactions now trigger an error message on the slave.
- A `GRANT` command that creates an anonymous user (that is, an account with an empty username) no longer requires `FLUSH PRIVILEGES` for the account to be recognized by the server. (Bug #473 [http://bugs.mysql.com/473])
- `CHANGE MASTER` now flushes `relay-log.info`. Previously this was deferred to the next run of `START SLAVE`, so if **mysqld** was shutdown on the slave after `CHANGE MASTER` without having run `START SLAVE`, the relay log's name and position were lost. At restart they were reloaded from `relay-log.info`, thus reverting to their old (incorrect) values from before `CHANGE MASTER` and leading to error messages (as the old relay log did not exist any more) and the slave threads refusing to start. (Bug #858 [http://bugs.mysql.com/858])

Bugs fixed:

- Fixed buffer overflow in password handling which could potentially be exploited by MySQL users with `ALTER` privilege on the `mysql.user` table to execute random code or to gain shell access with the UID of the **mysqld** process (thanks to Jedi/Sector One for spotting and reporting this bug).
- Fixed server crash on `FORCE INDEX` in a query that contained "Range checked for each record" in the `EXPLAIN` output. (Bug #1172 [http://bugs.mysql.com/1172])
- Fixed table/column grant handling: The proper sort order (from most specific to less specific, see Section 5.5.6, “Access Control, Stage 2: Request Verification”) was not honored. (Bug #928 [http://bugs.mysql.com/928])
- Fixed rare bug in `MYISAM` introduced in 4.0.3 where the index file header was not updated directly after an `UPDATE` of split dynamic rows. The symptom was that the table had a corrupted delete-link if **mysqld** was shut down or the table was checked directly after the update.
- Fixed `Can't unlock file` error when running **myisamchk** **--sort-index** on Windows. (Bug #1119 [http://bugs.mysql.com/1119])
- Fixed possible deadlock when changing `key_buffer_size` while the key cache was actively used. (Bug #1088 [http://bugs.mysql.com/1088])
- Fixed overflow bug in `MyISAM` and `ISAM` when a row is updated in a table with a large number of columns and at least one `BLOB/TEXT` column.
- Fixed incorrect result when doing `UNION` and `LIMIT #,#` when braces were not used around the `SELECT` parts.
- Fixed incorrect result when doing `UNION` and `ORDER BY .. LIMIT #` when one didn't use braces around the `SELECT` parts.
- Fixed problem with `SELECT SQL_CALC_FOUND_ROWS ... UNION ALL ... LIMIT #` where `FOUND_ROWS()` returned incorrect number of rows.
- Fixed unlikely stack bug when having a BIG expression of type `1+1-1+1-1...` in certain combinations. (Bug #871 [http://bugs.mysql.com/871])
- Fixed the bug that sometimes prevented a table with a `FULLTEXT` index from being marked as "ana-

lyzed".

- Fixed MySQL so that the column length (in C API) for the second column in `SHOW CREATE TABLE` is always larger than the data length. The only known application that was affected by the old behavior was Borland dbExpress, which truncated the output from the command. (Bug #1064 [http://bugs.mysql.com/1064])
- Fixed crash in comparisons of strings using the `tis620` character set. (Bug #1116 [http://bugs.mysql.com/1116])
- Fixed `ISAM` bug in `MAX()` optimization.
- **myisamchk --sort-records=N** no longer marks table as crashed if sorting failed because of an inappropriate key. (Bug #892 [http://bugs.mysql.com/892])
- Fixed a minor bug in `MyISAM` compressed table handling that sometimes made it impossible to repair compressed table in "Repair by sort" mode. "Repair with keycache" (**myisamchk --safe-recover**) worked, though. (Bug #1015 [http://bugs.mysql.com/1015])
- Fixed bug in propagating the version number to the manual included in the distribution files. (Bug #1020 [http://bugs.mysql.com/1020])
- Fixed key sorting problem (a `PRIMARY` key declared for a column that is not explicitly marked `NOT NULL` was sorted after a `UNIQUE` key for a `NOT NULL` column).
- Fixed the result of `INTERVAL` when applied to a `DATE` value. (Bug #792 [http://bugs.mysql.com/792])
- Fixed compiling of the embedded server library in the RPM spec file. (Bug #959 [http://bugs.mysql.com/959])
- Added some missing files to the RPM spec file and fixed some RPM building errors that occurred on Red Hat Linux 9. (Bug #998 [http://bugs.mysql.com/998])
- Fixed incorrect `XOR` evaluation in `WHERE` clause. (Bug #992 [http://bugs.mysql.com/992])
- Fixed bug with processing in query cache merged tables constructed from more then 255 tables. (Bug #930 [http://bugs.mysql.com/930])
- Fixed incorrect results from outer join query (for example, `LEFT JOIN`) when `ON` condition is always false, and range search in used. (Bug #926 [http://bugs.mysql.com/926])
- Fixed a bug causing incorrect results from `MATCH ... AGAINST()` in some joins. (Bug #942 [http://bugs.mysql.com/942])
- `MERGE` tables do not ignore `Using index` (from `EXPLAIN` output) anymore.
- Fixed a bug that prevented an empty table from being marked as "analyzed". (Bug #937 [http://bugs.mysql.com/937])
- Fixed **myisamchk --sort-records** crash when used on compressed table.
- Fixed slow (as compared to 3.23) `ALTER TABLE` and related commands such as `CREATE INDEX`. (Bug #712 [http://bugs.mysql.com/712])
- Fixed segmentation fault resulting from `LOAD DATA FROM MASTER` when the master was running without the `--log-bin` option. (Bug #934 [http://bugs.mysql.com/934])
- Fixed a security bug: A server compiled without SSL support still allowed connections by users who

had the REQUIRE SSL option specified for their accounts.

- Fixed a random bug: Sometimes the slave would replicate GRANT or REVOKE queries even if it was configured to exclude the mysql database from replication (for example, using --replicate-wild-ignore-table=mysql.%). (Bug #980 [http://bugs.mysql.com/980])

- The Last_Errno and Last_Error fields in the output of SHOW SLAVE STATUS are now cleared by CHANGE MASTER and when the slave SQL thread starts. (Bug #986 [http://bugs.mysql.com/986])

- A documentation mistake: It said that RESET SLAVE does not change connection information (master host, port, user, and password), whereas it does. The statement resets these to the startup options (master-host etc) if there were some. (Bug #985 [http://bugs.mysql.com/985])

- SHOW SLAVE STATUS now shows correct information (master host, port, user, and password) after RESET SLAVE (that is, it shows the new values, which are copied from the startup options if there were some). (Bug #985 [http://bugs.mysql.com/985])

- Disabled propagation of the original master's log position for events because this caused unexpected values for Exec_Master_Log_Pos and problems with MASTER_POS_WAIT() in A->B->C replication setup. (Bug #1086 [http://bugs.mysql.com/1086])

- Fixed a segfault in **mysqlbinlog** when --position=x was used with x being between a Create_file event and its fellow Append_block, Exec_load or Delete_file events. (Bug #1091 [http://bugs.mysql.com/1091])

- **mysqlbinlog** printed superfluous warnings when using --database, which caused syntax errors when piped to **mysql**. (Bug #1092 [http://bugs.mysql.com/1092])

- Made **mysqlbinlog --database** filter LOAD DATA INFILE too (previously, it filtered all queries except LOAD DATA INFILE). (Bug #1093 [http://bugs.mysql.com/1093])

- **mysqlbinlog** in some cases forgot to put a leading '#' in front of the original LOAD DATA INFILE (this command is displayed only for information, not to be run; it is later reworked to LOAD DATA LOCAL with a different filename, for execution by **mysql**). (Bug #1096 [http://bugs.mysql.com/1096])

- binlog-do-db and binlog-ignore-db incorrectly filtered LOAD DATA INFILE (it was half-written to the binary log). This resulted in a corrupted binary log, which could cause the slave to stop with an error. (Bug #1100 [http://bugs.mysql.com/1100])

- When, in a transaction, a transactional table (such as an InnoDB table) was updated, and later in the same transaction a non-transactional table (such as a MyISAM table) was updated using the updated content of the transactional table (with INSERT ... SELECT for example), the queries were written to the binary log in an incorrect order. (Bug #873 [http://bugs.mysql.com/873])

- When, in a transaction, INSERT ... SELECT updated a non-transactional table, and ROLLBACK was issued, no error was returned to the client. Now the client is warned that some changes could not be rolled back, as this was the case for normal INSERT. (Bug #1113 [http://bugs.mysql.com/1113])

- Fixed a potential bug: When STOP SLAVE was run while the slave SQL thread was in the middle of a transaction, and then CHANGE MASTER was used to point the slave to some non-transactional statement, the slave SQL thread could get confused (because it would still think, from the past, that it was in a transaction).

D.3.12. Changes in release 4.0.14 (18 Jul 2003)

Functionality added or changed:

- Added `default_week_format` system variable. The value is used as the default mode for the `WEEK()` function.
- **mysqld** now reads an additional option file group having a name corresponding to the server's release series: `[mysqld-4.0]` for 4.0.x servers, `[mysqld-4.1]` for 4.1.x servers, and so forth. This allows options to be specified on a series-specific basis.
- The `CONCAT_WS()` function no longer skips empty strings. (Bug #586 [http://bugs.mysql.com/586]).
- `InnoDB` now supports indexing a prefix of a column. This means, in particular, that `BLOB` and `TEXT` columns can be indexed in `InnoDB` tables, which was not possible before.
- A documentation change: Function `INTERVAL(NULL, ...)` returns `-1`.
- Enabled `INSERT` from `SELECT` when the table into which the records are inserted is also a table listed in the `SELECT`.
- Allow `CREATE TABLE` and `INSERT` from any `UNION`.
- The `SQL_CALC_FOUND_ROWS` option now always returns the total number of rows for any `UNION`.
- Removed `--table` option from **mysqlbinlog** to avoid repeating **mysqldump** functionality.
- Comment lines in option files can now start from the middle of a line, too (like `basedir=c:\mysql # installation directory`).
- Changed optimizer slightly to prefer index lookups over full table scans in some boundary cases.
- Added thread-specific `max_seeks_for_key` variable that can be used to force the optimizer to use keys instead of table scans even if the cardinality of the index is low.
- Added optimization that converts `LEFT JOIN` to normal join in some cases.
- A documentation change: added a paragraph about failover in replication (how to use a surviving slave as the new master, how to resume to the original setup). See Section 6.9, “Replication FAQ”.
- A documentation change: added warning notes about safe use of the `CHANGE MASTER` command. See Section 13.6.2.1, “`CHANGE MASTER TO` Syntax”.
- MySQL now issues a warning (not an error, as in 4.0.13) when it opens a table that was created with MySQL 4.1.
- Added `--nice` option to **mysqld_safe** to allow setting the niceness of the **mysqld** process. (Thanks to Christian Hammers for providing the initial patch.) (Bug #627 [http://bugs.mysql.com/627])
- Added `--read-only` option to cause **mysqld** to allow no updates except from slave threads or from users with the `SUPER` privilege. (Original patch from Markus Benning).
- `SHOW BINLOG EVENTS FROM x` where `x` is less than 4 now silently converts `x` to 4 instead of printing an error. The same change was done for `CHANGE MASTER TO MASTER_LOG_POS=x` and `CHANGE MASTER TO RELAY_LOG_POS=x`.
- **mysqld** now only adds an interrupt handler for the `SIGINT` signal if you start it with the new `--gdb` option. This is because some MySQL users encountered strange problems when they acci-

dentally sent SIGINT to **mysqld** threads.

- RESET SLAVE now clears the Last_Errno and Last_Error fields in the output of SHOW SLAVE STATUS.
- Added max_relay_log_size variable; the relay log is rotated automatically when its size exceeds max_relay_log_size. But if max_relay_log_size is 0 (the default), max_binlog_size is used (as in older versions). max_binlog_size still applies to binary logs in any case.
- FLUSH LOGS now rotates relay logs in addition to the other types of logs it rotates.

Bugs fixed:

- Comparison/sorting for latin1_de character set was rewritten. The old algorithm could not handle cases like "sä" > "ßa". See Section 5.8.1.1, “Using the German Character Set”. In rare cases it resulted in table corruption.
- Fixed a problem with the password prompt on Windows. (Bug #683 [http://bugs.mysql.com/683])
- ALTER TABLE ... UNION=(...) for MERGE table is now allowed even if some underlying MyISAM tables are read-only. (Bug #702 [http://bugs.mysql.com/702])
- Fixed a problem with CREATE TABLE t1 SELECT x'41'. (Bug #801 [http://bugs.mysql.com/801])
- Removed some incorrect lock warnings from the error log.
- Fixed memory overrun when doing REPAIR TABLE on a table with a multiple-part auto_increment key where one part was a packed CHAR.
- Fixed a probable race condition in the replication code that could potentially lead to INSERT statements not being replicated in the event of a FLUSH LOGS command or when the binary log exceeds max_binlog_size. (Bug #791 [http://bugs.mysql.com/791])
- Fixed a crashing bug in INTERVAL and GROUP BY or DISTINCT. (Bug #807 [http://bugs.mysql.com/807])
- Fixed bug in **mysqlhotcopy** so it actually aborts for unsuccessful table copying operations. Fixed another bug so that it succeeds when there are thousands of tables to copy. (Bug #812 [http://bugs.mysql.com/812])
- Fixed problem with **mysqlhotcopy** failing to read options from option files. (Bug #808 [http://bugs.mysql.com/808])
- Fixed bugs in optimizer that sometimes prevented MySQL from using FULLTEXT indexes even though it was possible (for example, in SELECT * FROM t1 WHERE MATCH a,b AGAINST("index") > 0).
- Fixed a bug with ``table is full'' in UNION operations.
- Fixed a security problem that enabled users with no privileges to obtain information on the list of existing databases by using SHOW TABLES and similar commands.
- Fixed a stack problem on UnixWare/OpenUnix.
- Fixed a configuration problem on UnixWare/OpenUNIX and OpenServer.

- Fixed a stack overflow problem in password verification.
- Fixed a problem with `max_user_connections`.
- `HANDLER` without an index now works properly when a table has deleted rows. (Bug #787 [http://bugs.mysql.com/787])
- Fixed a bug with `LOAD DATA` in **mysqlbinlog**. (Bug #670 [http://bugs.mysql.com/670])
- Fixed that `SET CHARACTER SET DEFAULT` works. (Bug #462 [http://bugs.mysql.com/462])
- Fixed `MERGE` table behavior in `ORDER BY ... DESC` queries. (Bug #515 [http://bugs.mysql.com/515])
- Fixed server crash on `PURGE MASTER LOGS` or `SHOW MASTER LOGS` when the binary log is off. (Bug #733 [http://bugs.mysql.com/733])
- Fixed password-checking problem on Windows. (Bug #464 [http://bugs.mysql.com/464])
- Fixed the bug in comparison of a `DATETIME` column and an integer constant. (Bug #504 [http://bugs.mysql.com/504])
- Fixed remote mode of **mysqlbinlog**. (Bug #672 [http://bugs.mysql.com/672])
- Fixed `ERROR 1105: Unknown error` that occurred for some `SELECT` queries, where a column that was declared as `NOT NULL` was compared with an expression that took `NULL` value.
- Changed timeout in `mysql_real_connect()` to use `poll()` instead of `select()` to work around problem with many open files in the client.
- Fixed incorrect results from `MATCH ... AGAINST` used with a `LEFT JOIN` query.
- Fixed a bug that limited the maximum value for **mysqld** variables to 4294967295 when they are specified on the command line.
- Fixed a bug that sometimes caused spurious ``Access denied'' errors in `HANDLER ... READ` statements, when a table is referenced via an alias.
- Fixed portability problem with `safe_malloc`, which caused MySQL to give "Freeing wrong aligned pointer" errors on SCO 3.2.
- `ALTER TABLE ... ENABLE/DISABLE KEYS` could cause a core dump when done after an `INSERT DELAYED` statement on the same table.
- Fixed problem with conversion of localtime to GMT where some times resulted in different (but correct) timestamps. Now MySQL should use the smallest possible timestamp value in this case. (Bug #316 [http://bugs.mysql.com/316])
- Very small query cache sizes could crash **mysqld**. (Bug #549 [http://bugs.mysql.com/549])
- Fixed a bug (accidentally introduced by us but present only in version 4.0.13) that made `INSERT ... SELECT` into an `AUTO_INCREMENT` column not replicate well. This bug is in the master, not in the slave. (Bug #490 [http://bugs.mysql.com/490])
- Fixed a bug: When an `INSERT ... SELECT` statement inserted rows into a non-transactional table, but failed at some point (for example, due to a ``Duplicate key'' error), the query was not written to the binary log. Now it is written to the binary log, with its error code, as all other queries are. About the `slave-skip-errors` option for how to handle partially completed queries in the slave, see Section 6.8, “Replication Startup Options”. (Bug #491 [http://bugs.mysql.com/491])

- `SET FOREIGN_KEY_CHECKS=0` was not replicated properly. The fix probably will not be backported to 3.23.
- On a slave, `LOAD DATA INFILE` which had no `IGNORE` or `REPLACE` clause on the master, was replicated with `IGNORE`. While this is not a problem if the master and slave data are identical (a `LOAD` that produces no duplicate conflicts on the master produces none on the slave anyway), which is true in normal operation, it is better for debugging not to silently add the `IGNORE`. That way, you can get an error message on the slave and discover that for some reason, the data on master and slave are different and investigate why. (Bug #571 [http://bugs.mysql.com/571])
- On a slave, `LOAD DATA INFILE` printed an incomplete ``Duplicate entry '%-.64s' for key %d'" message (the key name and value were not mentioned) in case of duplicate conflict (which does not happen in normal operation). (Bug #573 [http://bugs.mysql.com/573])
- When using a slave compiled with `--debug`, `CHANGE MASTER TO RELAY_LOG_POS` could cause a debug assertion failure. (Bug #576 [http://bugs.mysql.com/576])
- When doing a `LOCK TABLES WRITE` on an `InnoDB` table, commit could not happen, if the query was not written to the binary log (for example, if `--log-bin` was not used, or `binlog-ignore-db` was used). (Bug #578 [http://bugs.mysql.com/578])
- If a 3.23 master had open temporary tables that had been replicated to a 4.0 slave, and the binary log got rotated, these temporary tables were immediately dropped by the slave (which caused problems if the master used them subsequently). This bug had been fixed in 4.0.13, but in a manner which caused an unlikely inconvenience: If the 3.23 master died brutally (power failure), without having enough time to automatically write `DROP TABLE` statements to its binary log, then the 4.0.13 slave would not notice the temporary tables have to be dropped, until the slave **mysqld** server is restarted. This minor inconvenience is fixed in 3.23.57 and 4.0.14 (meaning the master must be upgraded to 3.23.57 and the slave to 4.0.14 to remove the inconvenience). (Bug #254 [http://bugs.mysql.com/254])
- If `MASTER_POS_WAIT()` was waiting, and the slave was idle, and the slave SQL thread terminated, `MASTER_POS_WAIT()` would wait forever. Now when the slave SQL thread terminates, `MASTER_POS_WAIT()` immediately returns `NULL` (``slave stopped"). (Bug #651 [http://bugs.mysql.com/651])
- After `RESET SLAVE; START SLAVE;`, the `Relay_Log_Space` value displayed by `SHOW SLAVE STATUS` was too big by four bytes. (Bug #763 [http://bugs.mysql.com/763])
- If a query was ignored on the slave (because of `--replicate-ignore-table` and other similar rules), the slave still checked if the query got the same error code (0, no error) as on the master. So if the master had an error on the query (for example, ``Duplicate entry" in a multiple-row insert), then the slave stopped and warned that the error codes didn't match. (Bug #797 [http://bugs.mysql.com/797])

D.3.13. Changes in release 4.0.13 (16 May 2003)

Functionality added or changed:

- `PRIMARY KEY` now implies `NOT NULL`. (Bug #390 [http://bugs.mysql.com/390])
- The Windows binary packages are now compiled with `--enable-local-infile` to match the Unix build configuration.
- Removed timing of tests from **mysql-test-run**. `time` does not accept all required parameters on many platforms (for example, QNX) and timing the tests is not really required (it's not a benchmark

anyway).

- SHOW MASTER STATUS and SHOW SLAVE STATUS required the SUPER privilege; now they accept REPLICATION CLIENT as well. (Bug #343 [http://bugs.mysql.com/343])
- Added multi-threaded MyISAM repair optimization and myisam_repair_threads variable to enable it. See Section 5.2.3, “Server System Variables”.
- Added innodb_max_dirty_pages_pct variable which controls amount of dirty pages allowed in InnoDB buffer pool.
- CURRENT_USER() and Access denied error messages now report the hostname exactly as it was specified in the GRANT command.
- Removed benchmark results from the source and binary distributions. They are still available in the BK source tree, though.
- InnoDB tables now support ANALYZE TABLE.
- MySQL now issues an error when it opens a table that was created with MySQL 4.1.
- Option --new now changes binary items (0xFFDF) to be treated as binary strings instead of numbers by default. This fixes some problems with character sets where it's convenient to input the string as a binary item. After this change you have to convert the binary string to INTEGER with a CAST if you want to compare two binary items with each other and know which one is bigger than the other. SELECT CAST(0xfeff AS UNSIGNED) < CAST(0xff AS UNSIGNED). This is the default behavior in MySQL 4.1. (Bug #152 [http://bugs.mysql.com/152])
- Enabled delayed_insert_timeout on Linux (most modern glibc libraries have a fixed pthread_cond_timedwait()). (Bug #211 [http://bugs.mysql.com/211])
- Don't create more insert delayed threads than given by max_delayed_threads. (Bug #211 [http://bugs.mysql.com/211])
- Changed UPDATE ... LIMIT to apply the limit to rows that were matched, whether or not they actually were changed. Previously the limit was applied as a restriction on the number of rows changed.
- Tuned optimizer to favor clustered index over table scan.
- BIT_AND() and BIT_OR() now return an unsigned 64-bit value.
- Added warnings to error log of why a secure connection failed (when running with --log-warnings).
- Deprecated options --skip-symlink and --use-symbolic-links and replaced these with --symbolic-links.
- The default option for innodb_flush_log_at_trx_commit was changed from 0 to 1 to make InnoDB tables ACID by default. See Section 15.5, “InnoDB Startup Options”.
- Added a feature to SHOW KEYS to display keys that are disabled by ALTER TABLE DISABLE KEYS command.
- When using a non-existing table type with CREATE TABLE, first try if the default table type exists before falling back to MyISAM.
- Added MEMORY as an alias for HEAP.

- Renamed function `rnd` to `my_rnd` as the name was too generic and is an exported symbol in `libmysqlclient` (thanks to Dennis Haney for the initial patch).
- Portability fix: renamed `include/dbug.h` to `include/my_dbug.h`.
- **mysqldump** no longer silently deletes the binary logs when invoked with the `--master-data` or `--first-slave` option; while this behavior was convenient for some users, others may suffer from it. Now you must explicitly ask for binary logs to be deleted by using the new `--delete-master-logs` option.
- If the slave is configured (using for example `--replicate-wild-ignore-table=mysql.%`) to exclude `mysql.user`, `mysql.host`, `mysql.db`, `mysql.tables_priv` and `mysql.columns_priv` from replication, then `GRANT` and `REVOKE` are not replicated.

Bugs fixed:

- Logged `Access denied` error message had incorrect `Using password` value. (Bug #398 [http://bugs.mysql.com/398])
- Fixed bug with `NATURAL LEFT JOIN`, `NATURAL RIGHT JOIN` and `RIGHT JOIN` when using many joined tables. The problem was that the `JOIN` method was not always associated with the tables surrounding the `JOIN` method. If you have a query that uses many `RIGHT JOIN` or `NATURAL ... JOINS` you should verify that they work as you expected after upgrading MySQL to this version. (Bug #291 [http://bugs.mysql.com/291])
- Fixed **mysql** parser not to erroneously interpret `'''` or `'"'` characters within `/* ... */` comment as beginning a quoted string.
- **mysql** command-line client no longer looks for `\*` commands inside backtick-quoted strings.
- Fixed `Unknown error` when using `UPDATE ... LIMIT`. (Bug #373 [http://bugs.mysql.com/373])
- Fixed problem with ANSI mode and `GROUP BY` with constants. (Bug #387 [http://bugs.mysql.com/387])
- Fixed bug with `UNION` and `OUTER JOIN`. (Bug #386 [http://bugs.mysql.com/386])
- Fixed bug if one used a multiple-table `UPDATE` and the query required a temporary table bigger than `tmp_table_size`. (Bug #286 [http://bugs.mysql.com/286])
- Run **mysql_install_db** with the `-IN-RPM` option for the Mac OS X installation to not fail on systems with improperly configured hostname configurations.
- `LOAD DATA INFILE` now reads `000000` as a zero date instead of `"2000-00-00"`.
- Fixed bug that caused `DELETE FROM table WHERE const_expression` always to delete the whole table (even if expression result was false). (Bug #355 [http://bugs.mysql.com/355])
- Fixed core dump bug when using `FORMAT('nan',#)`. (Bug #284 [http://bugs.mysql.com/284])
- Fixed name resolution bug with `HAVING ... COUNT(DISTINCT ...)`.
- Fixed incorrect result from truncation operator (`*`) in `MATCH ... AGAINST()` in some complex joins.

- Fixed a crash in `REPAIR ... USE_FRM` command, when used on read-only, nonexisting table or a table with a crashed index file.
- Fixed a crashing bug in **mysql** monitor program. It occurred if program was started with `--no-defaults`, with a prompt that contained the hostname and a connection to a non-existent database was requested.
- Fixed problem when comparing a key for a multi-byte character set. (Bug #152 [http://bugs.mysql.com/152])
- Fixed bug in `LEFT`, `RIGHT` and `MID` when used with multi-byte character sets and some `GROUP BY` queries. (Bug #314 [http://bugs.mysql.com/314])
- Fix problem with `ORDER BY` being discarded for some `DISTINCT` queries. (Bug #275 [http://bugs.mysql.com/275])
- Fixed that `SET SQL_BIG_SELECTS=1` works as documented (This corrects a new bug introduced in 4.0)
- Fixed some serious bugs in `UPDATE ... ORDER BY`. (Bug #241 [http://bugs.mysql.com/241])
- Fixed unlikely problem in optimizing `WHERE` clause with constant expression like in `WHERE 1 AND (a=1 AND b=1)`.
- Fixed that `SET SQL_BIG_SELECTS=1` works again.
- Introduced proper backtick quoting for db.table in `SHOW GRANTS`.
- `FULLTEXT` index stopped working after `ALTER TABLE` that converts `TEXT` column to `CHAR`. (Bug #283 [http://bugs.mysql.com/283])
- Fixed a security problem with `SELECT` and wildcarded select list, when user only had partial column `SELECT` privileges on the table.
- Mark a `MyISAM` table as "analyzed" only when all the keys are indeed analyzed.
- Only ignore world-writable `my.cnf` files that are regular files (and not, for example, named pipes or character devices).
- Fixed few smaller issues with `SET PASSWORD`.
- Fixed error message which contained deprecated text.
- Fixed a bug with two `NATURAL JOIN`s in the query.
- `SUM()` didn't return `NULL` when there was no rows in result or when all values was `NULL`.
- On Unix, symbolic link handling was not enabled by default and there was no way to turn this on.
- Added missing dashes to parameter `--open-files-limit` in **mysqld_safe**. (Bug #264 [http://bugs.mysql.com/264])
- Fixed incorrect hostname for TCP/IP connections displayed in `SHOW PROCESSLIST`.
- Fixed a bug with `NAN` in `FORMAT(...)` function ...
- Fixed a bug with improperly cached database privileges.
- Fixed a bug in `ALTER TABLE ENABLE / DISABLE KEYS` which failed to force a refresh of table data in the cache.

- Fixed bugs in replication of `LOAD DATA INFILE` for custom parameters (`ENCLOSED`, `TERMINATED` and so on) and temporary tables. (Bug #183 [http://bugs.mysql.com/183], Bug #222 [http://bugs.mysql.com/222])
- Fixed a replication bug when the master is 3.23 and the slave 4.0: the slave lost the replicated temporary tables if `FLUSH LOGS` was issued on the master. (Bug #254 [http://bugs.mysql.com/254])
- Fixed a bug when doing `LOAD DATA INFILE IGNORE`: When reading the binary log, **mysqlbinlog** and the replication code read `REPLACE` instead of `IGNORE`. This could make the slave's table become different from the master's table. (Bug #218 [http://bugs.mysql.com/218])
- Fixed a deadlock when `relay_log_space_limit` was set to a too small value. (Bug #79 [http://bugs.mysql.com/79])
- Fixed a bug in HAVING clause when an alias is used from the **select list**.
- Fixed overflow bug in `MyISAM` when a row is inserted into a table with a large number of columns and at least one `BLOB/TEXT` column. Bug was caused by incorrect calculation of the needed buffer to pack data.
- Fixed a bug when `SELECT` **@nonexistent_variable** caused the error in client - server protocol due to net_printf() being sent to the client twice.
- Fixed a bug in setting `SQL_BIG_SELECTS` option.
- Fixed a bug in `SHOW PROCESSLIST` which only displayed a localhost in the `"Host"` column. This was caused by a glitch that used only current thread information instead of information from the linked list of threads.
- Removed unnecessary Mac OS X helper files from server RPM. (Bug #144 [http://bugs.mysql.com/144])
- Allow optimization of multiple-table update for `InnoDB` tables as well.
- Fixed a bug in multiple-table updates that caused some rows to be updated several times.
- Fixed a bug in **mysqldump** when it was called with `--master-data`: the `CHANGE MASTER TO` commands appended to the SQL dump had incorrect coordinates. (Bug #159 [http://bugs.mysql.com/159])
- Fixed a bug when an updating query using `USER()` was replicated on the slave; this caused segfault on the slave. (Bug #178 [http://bugs.mysql.com/178]). `USER()` is still badly replicated on the slave (it is replicated to `""`).

D.3.14. Changes in release 4.0.12 (15 Mar 2003: Production)

Functionality added or changed:

- **mysqld** no longer reads options from world-writable config files.
- Integer values between 9223372036854775807 and 9999999999999999999 are now regarded as unsigned longlongs, not as floats. This makes these values work similar to values between 10000000000000000000 and 18446744073709551615.
- `SHOW PROCESSLIST` now includes the client TCP port after the hostname to make it easier to

know from which client the request originated.

- The `--new` option can be used to make a 4.0 server return `TIMESTAMP` as a string in `'YYYY-MM-DD HH:MM:SS'` format, the way that 4.1 servers do.

Bugs fixed:

- Fixed **mysqld** crash on extremely small values of `sort_buffer` variable.
- `INSERT INTO u SELECT ... FROM t` was written too late to the binary log if t was very frequently updated during the execution of this query. This could cause a problem with **mysqlbinlog** or replication. The master must be upgraded, not the slave. (Bug #136 [http://bugs.mysql.com/136])
- Fixed checking of random part of `WHERE` clause. (Bug #142 [http://bugs.mysql.com/142])
- Fixed a bug with multiple-table updates with `InnoDB` tables. This bug occurred as, in many cases, `InnoDB` tables cannot be updated ``on the fly," but offsets to the records have to be stored in a temporary table.
- Added missing file `mysql_secure_installation` to the `server` RPM subpackage. (Bug #141 [http://bugs.mysql.com/141])
- Fixed MySQL (and **myisamchk**) crash on artificially corrupted `.MYI` files.
- Don't allow `BACKUP TABLE` to overwrite existing files.
- Fixed a bug with multiple-table `UPDATE` statements when user had all privileges on the database where tables are located and there were any entries in `tables_priv` table, that is, `grant_option` was true.
- Fixed a bug that allowed a user with table or column grants on some table, `TRUNCATE` any table in the same database.
- Fixed deadlock when doing `LOCK TABLE` followed by `DROP TABLE` in the same thread. In this case one could still kill the thread with `KILL`.
- `LOAD DATA LOCAL INFILE` was not properly written to the binary log (hence not properly replicated). (Bug #82 [http://bugs.mysql.com/82])
- `RAND()` entries were not read correctly by **mysqlbinlog** from the binary log which caused problems when restoring a table that was inserted with `RAND()`. `INSERT INTO t1 VALUES(RAND())`. In replication this worked okay.
- `SET SQL_LOG_BIN=0` was ignored for `INSERT DELAYED` queries. (Bug #104 [http://bugs.mysql.com/104])
- `SHOW SLAVE STATUS` reported too old positions (columns `Relay_Master_Log_File` and `Exec_Master_Log_Pos`) for the last executed statement from the master, if this statement was the `COMMIT` of a transaction. The master must be upgraded for that, not the slave. (Bug #52 [http://bugs.mysql.com/52])
- `LOAD DATA INFILE` was not replicated by the slave if `replicate_*_table` was set on the slave. (Bug #86 [http://bugs.mysql.com/86])
- After `RESET SLAVE`, the coordinates displayed by `SHOW SLAVE STATUS` looked un-reset (although they were, but only internally). (Bug #70 [http://bugs.mysql.com/70])

- Fixed query cache invalidation on `LOAD DATA`.
- Fixed memory leak on `ANALYZE` procedure with error.
- Fixed a bug in handling `CHAR(0)` columns that could cause incorrect results from the query.
- Fixed rare bug with incorrect initialization of `AUTO_INCREMENT` column, as a secondary column in a multi-column key (see Section 3.6.9, "Using `AUTO_INCREMENT`"), when data was inserted with `INSERT ... SELECT` or `LOAD DATA` into an empty table.
- On Windows, `STOP SLAVE` didn't stop the slave until the slave got one new command from the master (this bug has been fixed for MySQL 4.0.11 by releasing updated 4.0.11a Windows packages, which include this individual fix on top of the 4.0.11 sources). (Bug #69 [http://bugs.mysql.com/69])
- Fixed a crash when no database was selected and `LOAD DATA` command was issued with full table name specified, including database prefix.
- Fixed a crash when shutting down replication on some platforms (for example, Mac OS X).
- Fixed a portability bug with `pthread_attr_getstacksize` on HP-UX 10.20 (Patch was also included in 4.0.11a sources).
- Fixed the `bigint` test to not fail on some platforms (for example, HP-UX and Tru64) due to different return values of the `atof()` function.
- Fixed the `rpl_rotate_logs` test to not fail on certain platforms (such as Mac OS X) due to a too-long file name (changed `slave-master-info.opt` to `.slave-mi`).

D.3.15. Changes in release 4.0.11 (20 Feb 2003)

Functionality added or changed:

- `NULL` is now sorted **LAST** if you use `ORDER BY ... DESC` (as it was before MySQL 4.0.2). This change was required to comply with the SQL standard. (The original change was made because we thought that standard SQL required `NULL` to be always sorted at the same position, but this was incorrect).
- Added `START TRANSACTION` (standard SQL syntax) as alias for `BEGIN`. This is recommended to use instead of `BEGIN` to start a transaction.
- Added `OLD_PASSWORD()` as a synonym for `PASSWORD()`.
- Allow keyword `ALL` in group functions.
- Added support for some new `INNER JOIN` and `JOIN` syntaxes. For example, `SELECT * FROM t1 INNER JOIN t2` didn't work before.
- Novell NetWare 6.0 porting effort completed, Novell patches merged into the main source tree.

Bugs fixed:

- Fixed problem with multiple-table delete and `InnoDB` tables.
- Fixed a problem with `BLOB NOT NULL` columns used with `IS NULL`.

- Re-added missing pre- and post(un)install scripts to the Linux RPM packages (they were missing after the renaming of the server subpackage).
- Fixed that table locks are not released with multiple-table updates and deletes with `InnoDB` storage engine.
- Fixed bug in updating `BLOB` columns with long strings.
- Fixed integer-wraparound when giving big integer (>= 10 digits) to function that requires an unsigned argument, like `CREATE TABLE (...) AUTO_INCREMENT=#`.
- `MIN(key_column)` could in some cases return `NULL` on a column with `NULL` and other values.
- `MIN(key_column)` and `MAX(key_column)` could in some cases return incorrect values when used in `OUTER JOIN`.
- `MIN(key_column)` and `MAX(key_column)` could return incorrect values if one of the tables was empty.
- Fixed rare crash in compressed `MyISAM` tables with blobs.
- Fixed bug in using aggregate functions as argument for `INTERVAL`, `CASE`, `FIELD`, `CONCAT_WS`, `ELT` and `MAKE_SET` functions.
- When running with `--lower-case-table-names` (default on Windows) and you had tables or databases with mixed case on disk, then executing `SHOW TABLE STATUS` followed with `DROP DATABASE` or `DROP TABLE` could fail with `Errcode 13`.

D.3.16. Changes in release 4.0.10 (29 Jan 2003)

Functionality added or changed:

- Added option `--log-error[=file_name]` to **mysqld_safe** and **mysqld**. This option forces all error messages to be put in a log file if the option `--console` is not given. On Windows `--log-error` is enabled as default, with a default name of *`host_name`*`.err` if the name is not specified.
- Changed some things from `Warning:` to `Note:` in the log files.
- The **mysqld** server should now compile on NetWare.
- Added optimization that if one does `GROUP BY ... ORDER BY NULL` then result is not sorted.
- New `--ft-stopword-file` command-line option for **mysqld** to replace/disable the built-in stopword list that is used in full-text searches. See Section 5.2.3, “Server System Variables”.
- Changed default stack size from 64KB to 192KB; This fixes a core dump problem on Red Hat 8.0 and other systems with a `glibc` that requires a stack size larger than 128K for `gethostbyaddr()` to resolve a hostname. You can fix this for earlier MySQL versions by starting **mysqld** with `--thread-stack=192K`.
- Added `mysql_waitpid` to the binary distribution and the `MySQL-client` RPM subpackage (required for **mysql-test-run**).
- Renamed the main `MySQL` RPM package to `MySQL-server`. When updating from an older version, `MySQL-server.rpm` simply replaces `MySQL.rpm`.

- If a slave is configured with `replicate_wild_do_table=db.%` or `replicate_wild_ignore_table=db.%`, these rules are applied to `CREATE/DROP DATABASE`, too.
- Added timeout value for `MASTER_POS_WAIT()`.

Bugs fixed:

- Fixed initialization of the random seed for newly created threads to give a better `rand()` distribution from the first call.
- Fixed a bug that caused **mysqld** to hang when a table was opened with the `HANDLER` command and then dropped without being closed.
- Fixed bug in logging to binary log (which affects replication) a query that inserts a `NULL` in an `AUTO_INCREMENT` column and also uses `LAST_INSERT_ID()`.
- Fixed an unlikely bug that could cause a memory overrun when using `ORDER BY constant_expression`.
- Fixed a table corruption in **myisamchk** parallel repair mode.
- Fixed bug in query cache invalidation on simple table renaming.
- Fixed bug in **mysqladmin --relative**.
- On some 64-bit systems, `show status` reported a strange number for `Open_files` and `Open_streams`.
- Fixed incorrect number of columns in `EXPLAIN` on empty table.
- Fixed bug in `LEFT JOIN` that caused zero rows to be returned in the case the `WHERE` condition was evaluated as `FALSE` after reading `const` tables. (Unlikely condition).
- `FLUSH PRIVILEGES` didn't correctly flush table/column privileges when `mysql.tables_priv` is empty.
- Fixed bug in replication when using `LOAD DATA INFILE` one a file that updated an `AUTO_INCREMENT` column with `NULL` or `0`. This bug only affected MySQL 4.0 masters (not slaves or MySQL 3.23 masters). **Note**: If you have a slave that has replicated a file with generated `AUTO_INCREMENT` columns then the slave data is corrupted and you should reinitialize the affected tables from the master.
- Fixed possible memory overrun when sending a `BLOB` value larger than 16M to the client.
- Fixed incorrect error message when setting a `NOT NULL` column to an expression that returned `NULL`.
- Fixed core dump bug in *`str`* `LIKE "%`*`other_str`*`%"` where *`str`* or *`other_str`* contained characters >= 128.
- Fixed bug: When executing on master `LOAD DATA` and `InnoDB` failed with `table full` error the binary log was corrupted.

D.3.17. Changes in release 4.0.9 (09 Jan 2003)

Functionality added or changed:

- `OPTIMIZE TABLE` for `MyISAM` tables treats all `NULL` values as different when calculating cardinality. This helps in optimizing joins between tables where one of the tables has a lot of `NULL` values in a indexed column:

  ```
  SELECT * from t1, t2 where t1.a=t2.key_with_a_lot_of_null;
  ```

- Added join operator `FORCE INDEX (key_list)`. This acts likes `USE INDEX (key_list)` but with the addition that a table scan is assumed to be VERY expensive. One bad thing with this is that it makes `FORCE` a reserved word.
- Reset internal row buffer in `MyISAM` after each query. This reduces memory in case you have a lot of big blobs in a table.

Bugs fixed:

- A security patch in 4.0.8 causes the **mysqld** server to die if the remote hostname can't be resolved. This is now fixed.
- Fixed crash when replication big `LOAD DATA INFILE` statement that caused log rotation.

D.3.18. Changes in release 4.0.8 (07 Jan 2003)

Functionality added or changed:

- Default `max_packet_length` for `libmysqld.c` is now 1024*1024*1024.
- You can now specify `max_allowed_packet` in a file read by `mysql_options(MYSQL_READ_DEFAULT_FILE)`. for clients.
- When sending a too big packet to the server with the not compressed protocol, the client now gets an error message instead of a lost connection.
- We now send big queries/result rows in bigger hunks, which should give a small speed improvement.
- Fixed some bugs with the compressed protocol for rows > 16MB.
- `InnoDB` tables now also support `ON UPDATE CASCADE` in `FOREIGN KEY` constraints. See the `InnoDB` section in the manual for the `InnoDB` changelog.

Bugs fixed:

- Fixed bug in `ALTER TABLE` with `BDB` tables.
- Fixed core dump bug in `QUOTE()` function.
- Fixed a bug in handling communication packets bigger than 16MB. Unfortunately this required a protocol change; If you upgrade the server to 4.0.8 and above and have clients that use packets >= 255*255*255 bytes (=16581375) you must also upgrade your clients to at least 4.0.8. If you don't upgrade, the clients hang when sending a big packet.
- Fixed bug when sending blobs longer than 16MB to client.

- Fixed bug in `GROUP BY` when used on BLOB column with `NULL` values.
- Fixed a bug in handling `NULL` values in `CASE` ... WHEN ...

D.3.19. Changes in release 4.0.7 (20 Dec 2002)

Functionality added or changed:

- **mysqlbug** now also reports the compiler version used for building the binaries (if the compiler supports the option `--version`).

Bugs fixed:

- Fixed compilation problems on OpenUnix and HPUX 10.20.
- Fixed some optimization problems when compiling MySQL with `-DBIG_TABLES` on a 32-bit system.
- `mysql_drop_db()` didn't check permissions properly so anyone could drop another users database. `DROP DATABASE` is checked properly.

D.3.20. Changes in release 4.0.6 (14 Dec 2002: Gamma)

Functionality added or changed:

- Added syntax support for `CHARACTER SET` *`xxx`* and `CHARSET=`*`xxx`* table options (to be able to read table dumps from 4.1).
- Fixed replication bug that caused the slave to loose its position in some cases when the replication log was rotated.
- Fixed that a slave restarts from the start of a transaction if it's killed in the middle of one.
- Moved the manual pages from `man` to `man/man1` in the binary distributions.
- The default type returned by `IFNULL(A,B)` is now set to be the more 'general' of the types of `A` and `B`. (The order is `STRING`, `REAL` or `INTEGER`).
- Moved the **mysql.server** startup script in the RPM packages from `/etc/rc.d/init.d/mysql` to `/etc/init.d/mysql` (which almost all current Linux distributions support for LSB compliance).
- Added `Qcache_lowmem_prunes` status variable (number of queries that were deleted from the cache because of low memory).
- Fixed **mysqlcheck** so it can deal with table names containing dashes.
- Bulk insert optimization (see Section 5.2.3, “Server System Variables”) is no longer used when inserting small (less than 100) number of rows.
- Optimization added for queries like `SELECT ... FROM merge_table WHERE indexed_column=constant_expr`.

- Added functions `LOCALTIME` and `LOCALTIMESTAMP` as synonyms for `NOW()`.
- `CEIL` is now an alias for `CEILING`.
- The `CURRENT_USER()` function can be used to get a `user@host` value as it was matched in the `GRANT` system. See Section 12.8.3, “Information Functions”.
- Fixed `CHECK` constraints to be compatible with standard SQL. This made `CHECK` a reserved word. (Checking of `CHECK` constraints is still not implemented).
- Added `CAST(... as CHAR)`.
- Added PostgreSQL compatible `LIMIT` syntax: `SELECT ... LIMIT` *`row_count`* `OFFSET` *`offset`*
- `mysql_change_user()` now resets the connection to the state of a fresh connect (Ie, `ROLLBACK` any active transaction, close all temporary tables, reset all user variables etc..)
- `CHANGE MASTER` and `RESET SLAVE` now require that slave threads both be stopped; these commands return an error if at least one of these two threads is running.

Bugs fixed:

- Fixed number of found rows returned in `multi table updates`
- Make `--lower-case-table-names` default on Mac OS X as the default filesystem (HFS+) is case insensitive. See Section 9.2.2, “Identifier Case Sensitivity”.
- Transactions in `AUTOCOMMIT=0` mode didn't rotate binary log.
- A fix for the bug in a `SELECT` with joined tables with `ORDER BY` and `LIMIT` clause when `filesort` had to be used. In that case `LIMIT` was applied to `filesort` of one of the tables, although it could not be. This fix also solved problems with `LEFT JOIN`.
- `mysql_server_init()` now makes a copy of all arguments. This fixes a problem when using the embedded server in C# program.
- Fixed buffer overrun in `libmysqlclient` library that allowed a malicious MySQL server to crash the client application.
- Fixed security-related bug in `mysql_change_user()` handling. All users are strongly recommended to upgrade to version 4.0.6.
- Fixed bug that prevented `--chroot` command-line option of **mysqld** from working.
- Fixed bug in phrase operator `"..."` in boolean full-text search.
- Fixed bug that caused `OPTIMIZE TABLE` to corrupt the table under some rare circumstances.
- Part rewrite of multiple-table-update to optimize it, make it safer and more bug-free.
- `LOCK TABLES` now works together with multiple-table-update and multiple-table-delete.
- `--replicate-do=`*`xxx`* didn't work for `UPDATE` commands. (Bug introduced in 4.0.0)
- Fixed shutdown problem on Mac OS X.
- Major `InnoDB` bugs in `REPLACE, AUTO_INCREMENT, INSERT INTO ... SELECT ...`

were fixed. See the `InnoDB` changelog in the `InnoDB` section of the manual.

- `RESET SLAVE` caused a crash if the slave threads were running.

D.3.21. Changes in release 4.0.5 (13 Nov 2002)

Functionality added or changed:

- Port number was added to hostname (if it is known) in `SHOW PROCESSLIST` command
- Changed handling of last argument in `WEEK()` so that you can get week number according to the ISO 8601 specification. (Old code should still work).
- Fixed that `INSERT DELAYED` threads don't hang on `Waiting for INSERT` when one sends a `SIGHUP` to **mysqld**.
- Change that `AND` works according to standard SQL when it comes to `NULL` handling. In practice, this affects only queries where you do something like `WHERE ... NOT (NULL AND 0)`.
- **mysqld** now resolves `basedir` to its full path (with `realpath()`). This enables one to use relative symlinks to the MySQL installation directory. This however causes `show variables` to report different directories on systems where there is a symbolic link in the path.
- Fixed that MySQL does not use index scan on index disabled with `IGNORE INDEX` or `USE INDEX`. to be ignored.
- Added `--use-frm` option to **mysqlcheck**. When used with `REPAIR TABLE`, it gets the table structure from the `.frm` file, so the table can be repaired even if the `.MYI` header is corrupted.
- Fixed bug in `MAX()` optimization when used with `JOIN` and `ON` expressions.
- Added support for reading of MySQL 4.1 table definition files.
- `BETWEEN` behavior changed (see Section 12.1.3, “Comparison Functions and Operators”). Now `datetime_col BETWEEN timestamp AND timestamp` should work as expected.
- One can create `TEMPORARY MERGE` tables now.
- `DELETE FROM myisam_table` now shrinks not only the `.MYD` file but also the `.MYI` file.
- When one uses the `--open-files-limit=#` option to **mysqld_safe** it's now passed on to **mysqld**.
- Changed output from `EXPLAIN` from `'where used'` to `'Using where'` to make it more in line with other output.
- Removed variable `safe_show_database` as it was no longer used.
- Updated source tree to be built using **automake** 1.5 and **libtool** 1.4.
- Fixed an inadvertently changed option (`--ignore-space`) back to the original `--ignore-spaces` in `mysqlclient`. (Both syntaxes work).
- Don't require `UPDATE` privilege when using `REPLACE`.
- Added support for `DROP TEMPORARY TABLE ...`, to be used to make replication safer.

- When transactions are enabled, all commands that update temporary tables inside a `BEGIN/COMMIT` are now stored in the binary log on `COMMIT` and not stored if one does `ROLLBACK`. This fixes some problems with non-transactional temporary tables used inside transactions.
- Allow braces in joins in all positions. Formerly, things like `SELECT * FROM (t2 LEFT JOIN t3 USING (a)), t1` worked, but not `SELECT * FROM t1, (t2 LEFT JOIN t3 USING (a))`. Note that braces are simply removed, they do not change the way the join is executed.
- `InnoDB` now supports also isolation levels `READ UNCOMMITTED` and `READ COMMITTED`. For a detailed `InnoDB` changelog, see Section D.9, "`InnoDB` Change History" in this manual.

Bugs fixed:

- Fixed bug in `MAX()` optimization when used with `JOIN` and `ON` expressions.
- Fixed that `INSERT DELAY` threads don't hang on `Waiting for INSERT` when one sends a `SIGHUP` to **mysqld**.
- Fixed that MySQL does not use an index scan on an index that has been disabled with `IGNORE INDEX` or `USE INDEX`.
- Corrected test for `root` user in **mysqld_safe**.
- Fixed error message issued when storage engine cannot do `CHECK TABLE` or `REPAIR TABLE`.
- Fixed rare core dump problem in complicated `GROUP BY` queries that didn't return any result.
- Fixed **mysqlshow** to work properly with wildcarded database names and with database names that contain underscores.
- Portability fixes to get MySQL to compile cleanly with Sun Forte 5.0.
- Fixed `MyISAM` crash when using dynamic-row tables with huge numbers of packed columns.
- Fixed query cache behavior with `BDB` transactions.
- Fixed possible floating point exception in `MATCH` relevance calculations.
- Fixed bug in full-text search `IN BOOLEAN MODE` that made `MATCH` to return incorrect relevance value in some complex joins.
- Fixed a bug that limited `MyISAM` key length to a value slightly less that 500. It is exactly 500 now.
- Fixed that `GROUP BY` on columns that may have a `NULL` value doesn't always use disk based temporary tables.
- The filename argument for the `--des-key-file` argument to **mysqld** is interpreted relative to the data directory if given as a relative pathname.
- Removed a condition that temp table with index on column that can be `NULL` has to be `MyISAM`. This was okay for 3.23, but not needed in 4.*. This resulted in slowdown in many queries since 4.0.2.
- Small code improvement in multiple-table updates.
- Fixed a newly introduced bug that caused `ORDER BY ... LIMIT` *`row_count`* to not return all rows.

- Fixed a bug in multiple-table deletes when outer join is used on an empty table, which gets first to be deleted.
- Fixed a bug in multiple-table updates when a single table is updated.
- Fixed bug that caused `REPAIR TABLE` and **myisamchk** to corrupt `FULLTEXT` indexes.
- Fixed bug with caching the `mysql` grant table database. Now queries in this database are not cached in the query cache.
- Small fix in **mysqld_safe** for some shells.
- Give error if a `MyISAM MERGE` table has more than 2 ^ 32 rows and MySQL was not compiled with `-DBIG_TABLES`.
- Fixed some `ORDER BY ... DESC` problems with `InnoDB` tables.

D.3.22. Changes in release 4.0.4 (29 Sep 2002)

- Fixed bug where `GRANT/REVOKE` failed if hostname was given in non-matching case.
- Don't give warning in `LOAD DATA INFILE` when setting a `timestamp` to a string value of `'0'`.
- Fixed bug in **myisamchk -R** mode.
- Fixed bug that caused **mysqld** to crash on `REVOKE`.
- Fixed bug in `ORDER BY` when there is a constant in the `SELECT` statement.
- One didn't get an error message if **mysqld** couldn't open the privilege tables.
- `SET PASSWORD FOR ...` closed the connection in case of errors (bug from 4.0.3).
- Increased maximum possible `max_allowed_packet` in **mysqld** to 1GB.
- Fixed bug when doing a multiple-row `INSERT` on a table with an `AUTO_INCREMENT` key which was not in the first part of the key.
- Changed `LOAD DATA INFILE` to not re-create index if the table had rows from before.
- Fixed overrun bug when calling `AES_DECRYPT()` with incorrect arguments.
- `--skip-ssl` can now be used to disable SSL in the MySQL clients, even if one is using other SSL options in an option file or previously on the command line.
- Fixed bug in `MATCH ... AGAINST( ... IN BOOLEAN MODE)` used with `ORDER BY`.
- Added `LOCK TABLES` and `CREATE TEMPORARY TABLES` privilege on the database level. You must run the **mysql_fix_privilege_tables** script on old installations to activate these.
- In `SHOW TABLE ... STATUS`, compressed tables sometimes showed up as `dynamic`.
- `SELECT @@[global|session].`*`var_name`* didn't report `global | session` in the result column name.
- Fixed problem in replication that `FLUSH LOGS` in a circular replication setup created an infinite number of binary log files. Now a `rotate-binary-log` command in the binary log does not

cause slaves to rotate logs.

- Removed `STOP EVENT` from binary log when doing `FLUSH LOGS`.
- Disable the use of `SHOW NEW MASTER FOR SLAVE` as this needs to be completely reworked in a future release.
- Fixed a bug with constant expression (for example, column of a one-row table, or column from a table, referenced by a `UNIQUE` key) appeared in `ORDER BY` part of `SELECT DISTINCT`.
- `--log-binary=a.b.c` now properly strips off `.b.c`.
- `FLUSH LOGS` removed numerical extension for all future update logs.
- `GRANT ... REQUIRE` didn't store the SSL information in the `mysql.user` table if SSL was not enabled in the server.
- `GRANT ... REQUIRE NONE` can now be used to remove SSL information.
- `AND` is now optional between `REQUIRE` options.
- `REQUIRE` option was not properly saved, which could cause strange output in `SHOW GRANTS`.
- Fixed that **mysqld --help** reports correct values for `--datadir` and `--bind-address`.
- Fixed that one can drop UDFs that didn't exist when **mysqld** was started.
- Fixed core dump problem with `SHOW VARIABLES` on some 64-bit systems (like Solaris SPARC).
- Fixed a bug in `my_getopt()`; `--set-variable` syntax didn't work for those options that didn't have a valid variable in the `my_option` struct. This affected at least the `default-table-type` option.
- Fixed a bug from 4.0.2 that caused `REPAIR TABLE` and **myisamchk --recover** to fail on tables with duplicates in a unique key.
- Fixed a bug from 4.0.3 in calculating the default data type for some functions. This affected queries of type `CREATE TABLE` *`tbl_name`* `SELECT` *`expression`*`(),...`
- Fixed bug in queries of type `SELECT * FROM table-list GROUP BY ...` and `SELECT DISTINCT * FROM ...`.
- Fixed bug with the `--slow-log` when logging an administrator command (like `FLUSH TABLES`).
- Fixed a bug that `OPTIMIZE TABLE` of locked and modified table, reported table corruption.
- Fixed a bug in `my_getopt()` in handling of special prefixes (`--skip-`, `--enable-`). `--skip-external-locking` didn't work and the bug may have affected other similar options.
- Fixed bug in checking for output file name of the `tee` option.
- Added some more optimization to use index for `SELECT ... FROM many_tables .. ORDER BY key limit #`
- Fixed problem in `SHOW OPEN TABLES` when a user didn't have access permissions to one of the opened tables.

D.3.23. Changes in release 4.0.3 (26 Aug 2002: Beta)

- Fixed problem with types of user variables. (Bug #551 [http://bugs.mysql.com/551])
- Fixed problem with `configure ... --localstatedir=...`.
- Cleaned up **mysql.server** script.
- Fixed a bug in **mysqladmin shutdown** when pid file was modified while **mysqladmin** was still waiting for the previous one to disappear. This could happen during a very quick restart and caused **mysqladmin** to hang until `shutdown_timeout` seconds had passed.
- Don't increment warnings when setting `AUTO_INCREMENT` columns to `NULL` in `LOAD DATA INFILE`.
- Fixed all boolean type variables/options to work with the old syntax, for example, all of these work: `--lower-case-table-names`, `--lower-case-table-names=1`, `-O lower-case-table-names=1`, `--set-variable=lower-case-table-names=1`
- Fixed shutdown problem (SIGTERM signal handling) on Solaris. (Bug from 4.0.2).
- `SHOW MASTER STATUS` now returns an empty set if binary log is not enabled.
- `SHOW SLAVE STATUS` now returns an empty set if slave is not initialized.
- Don't update `MyISAM` index file on update if not strictly necessary.
- Fixed bug in `SELECT DISTINCT ... FROM many_tables ORDER BY not-used-column`.
- Fixed a bug with `BIGINT` values and quoted strings.
- Added `QUOTE()` function that performs SQL quoting to produce values that can be used as data values in queries.
- Changed variable `DELAY_KEY_WRITE` to an enumeration to allow it to be set for all tables without taking down the server.
- Changed behavior of `IF(condition,column,NULL)` so that it returns the value of the column type.
- Made **safe_mysqld** a symlink to **mysqld_safe** in binary distribution.
- Fixed security bug when having an empty database name in the `user.db` table.
- Fixed some problems with `CREATE TABLE ... SELECT function()`.
- **mysqld** now has the option `--temp-pool` enabled by default as this gives better performance with some operating systems.
- Fixed problem with too many allocated alarms on slave when connecting to master many times (normally not a very critical error).
- Fixed hang in `CHANGE MASTER TO` if the slave thread died very quickly.
- Big cleanup in replication code (less logging, better error messages, etc..)
- If the `--code-file` option is specified, the server calls `setrlimit()` to set the maximum allowed core file size to unlimited, so core files can be generated.

- Fixed bug in query cache after temporary table creation.
- Added `--count=N` (`-c`) option to **mysqladmin**, to make the program do only *N* iterations. To be used with `--sleep` (`-i`). Useful in scripts.
- Fixed bug in multiple-table `UPDATE`: when updating a table, `do_select()` became confused about reading records from a cache.
- Fixed bug in multiple-table `UPDATE` when several columns were referenced from a single table
- Fixed bug in truncating nonexisting table.
- Fixed bug in `REVOKE` that caused user resources to be randomly set.
- Fixed bug in `GRANT` for the new `CREATE TEMPORARY TABLE` privilege.
- Fixed bug in multiple-table `DELETE` when tables are re-ordered in the table initialization method and ref_lengths are of different sizes.
- Fixed two bugs in `SELECT DISTINCT` with large tables.
- Fixed bug in query cache initialization with very small query cache size.
- Allow `DEFAULT` with `INSERT` statement.
- The startup parameters `myisam_max_sort_file_size` and `myisam_max_extra_sort_file_size` are now given in bytes, not megabytes.
- External system locking of `MyISAM`/`ISAM` files is now turned off by default. One can turn this on with `--external-locking`. (For most users this is never needed).
- Fixed core dump bug with `INSERT ... SET db_name.tbl_name.col_name=''`.
- Fixed client hangup bug when using some SQL commands with incorrect syntax.
- Fixed a timing bug in `DROP DATABASE`
- New `SET [GLOBAL | SESSION]` syntax to change thread-specific and global system variables at runtime.
- Added variable `slave_compressed_protocol`.
- Renamed variable `query_cache_startup_type` to `query_cache_type`, `myisam_bulk_insert_tree_size` to `bulk_insert_buffer_size`, `record_buffer` to `read_buffer_size` and `record_rnd_buffer` to `read_rnd_buffer_size`.
- Renamed some SQL variables, but old names still work until 5.0. See Section 2.10.3, “Upgrading from Version 3.23 to 4.0”.
- Renamed `--skip-locking` to `--skip-external-locking`.
- Removed unused variable `query_buffer_size`.
- Fixed a bug that made the pager option in the **mysql** client non-functional.
- Added full `AUTO_INCREMENT` support to `MERGE` tables.
- Extended `LOG()` function to accept an optional arbitrary base parameter. See Section 12.4.2, “Mathematical Functions”.

- Added `LOG2()` function (useful for finding out how many bits a number would require for storage).
- Added `LN()` natural logarithm function for compatibility with other databases. It is synonymous with `LOG(X)`.

D.3.24. Changes in release 4.0.2 (01 Jul 2002)

- Cleaned up `NULL` handling for default values in `DESCRIBE tbl_name`.
- Fixed `TRUNCATE()` to round up negative values to the nearest integer.
- Changed `--chroot=path` option to execute `chroot()` immediately after all options have been parsed.
- Don't allow database names that contain '\'.
- `lower_case_table_names` now also applies to database names.
- Added `XOR` operator (logical and bitwise `XOR`) with `^` as a synonym for bitwise `XOR`.
- Added function `IS_FREE_LOCK("lock_name")`. Based on code contributed by Hartmut Holzgraefe `<hartmut@six.de>`.
- Removed `mysql_ssl_clear()` from C API, as it was not needed.
- `DECIMAL` and `NUMERIC` types can now read exponential numbers.
- Added `SHA1()` function to calculate 160 bit hash value as described in RFC 3174 (Secure Hash Algorithm). This function can be considered a cryptographically more secure equivalent of `MD5()`. See Section 12.8.2, “Encryption Functions”.
- Added `AES_ENCRYPT()` and `AES_DECRYPT()` functions to perform encryption according to AES standard (Rijndael). See Section 12.8.2, “Encryption Functions”.
- Added `--single-transaction` option to **mysqldump**, allowing a consistent dump of `InnoDB` tables. See Section 8.8, “The **mysqldump** Database Backup Program”.
- Fixed bug in `innodb_log_group_home_dir` in `SHOW VARIABLES`.
- Fixed a bug in optimizer with merge tables when non-unique values are used in summing up (causing crashes).
- Fixed a bug in optimizer when a range specified makes index grouping impossible (causing crashes).
- Fixed a rare bug when `FULLTEXT` index is present and no tables are used.
- Added privileges `CREATE TEMPORARY TABLES`, `EXECUTE`, `LOCK TABLES`, `REPLICATION CLIENT`, `REPLICATION SLAVE`, `SHOW DATABASES` and `SUPER`. To use these, you must have run the **mysql_fix_privilege_tables** script after upgrading.
- Fixed query cache align data bug.
- Fixed mutex bug in replication when reading from master fails.
- Added missing mutex in `TRUNCATE TABLE`. This fixes some core dump/hangup problems when using `TRUNCATE TABLE`.

- Fixed bug in multiple-table `DELETE` when optimizer uses only indexes.
- Fixed that `ALTER TABLE` *`tbl_name`* `RENAME` *`new_tbl_name`* is as fast as `RENAME TABLE`.
- Fixed bug in `GROUP BY` with two or more columns, where at least one column can contain `NULL` values.
- Use `Turbo Boyer-Moore` algorithm to speed up `LIKE "%keyword%"` searches.
- Fixed bug in `DROP DATABASE` with symlink.
- Fixed crash in `REPAIR ... USE_FRM`.
- Fixed bug in `EXPLAIN` with `LIMIT offset != 0`.
- Fixed bug in phrase operator `"..."` in boolean full-text search.
- Fixed bug that caused duplicated rows when using truncation operator `*` in boolean full-text search.
- Fixed bug in truncation operator of boolean full-text search (incorrect results when there are only `+word*`s in the query).
- Fixed bug in boolean full-text search that caused a crash when an identical `MATCH` expression that did not use an index appeared twice.
- Query cache is now automatically disabled in **mysqldump**.
- Fixed problem on Windows 98 that made sending of results very slow.
- Boolean full-text search weighting scheme changed to something more reasonable.
- Fixed bug in boolean full-text search that caused MySQL to ignore queries of `ft_min_word_len` characters.
- Boolean full-text search now supports ``phrase searches."
- New configure option `--without-query-cache`.
- Memory allocation strategy for ``root memory" changed. Block size now grows with number of allocated blocks.
- `INET_NTOA()` now returns `NULL` if you give it an argument that is too large (greater than the value corresponding to `255.255.255.255`).
- Fix `SQL_CALC_FOUND_ROWS` to work with `UNION`. It works only if the first `SELECT` has this option and if there is global `LIMIT` for the entire statement. For the moment, this requires using parentheses for individual `SELECT` queries within the statement.
- Fixed bug in `SQL_CALC_FOUND_ROWS` and `LIMIT`.
- Don't give an error for `CREATE TABLE ...(... VARCHAR(0))`.
- Fixed `SIGINT` and `SIGQUIT` problems in `mysql.cc` on Linux with some `glibc` versions.
- Fixed bug in `convert.cc`, which is caused by having an incorrect `net_store_length()` linked in the `CONVERT::store()` method.
- `DOUBLE` and `FLOAT` columns now honor the `UNSIGNED` flag on storage.

- InnoDB now retains foreign key constraints through ALTER TABLE and CREATE/DROP INDEX.
- InnoDB now allows foreign key constraints to be added through the ALTER TABLE syntax.
- InnoDB tables can now be set to automatically grow in size (autoextend).
- Added --ignore-lines=n option to **mysqlimport**. This has the same effect as the IGNORE n LINES clause for LOAD DATA.
- Fixed bug in UNION with last offset being transposed to total result set.
- REPAIR ... USE_FRM added.
- Fixed that DEFAULT_SELECT_LIMIT is always imposed on UNION result set.
- Fixed that some SELECT options can appear only in the first SELECT.
- Fixed bug with LIMIT with UNION, where last select is in the braces.
- Fixed that full-text works fine with UNION operations.
- Fixed bug with indexless boolean full-text search.
- Fixed bug that sometimes appeared when full-text search was used with const tables.
- Fixed incorrect error value when doing a SELECT with an empty HEAP table.
- Use ORDER BY column DESC now sorts NULL values first. (In other words, NULL values sort first in all cases, whether or not DESC is specified.) This is changed back in 4.0.10.
- Fixed bug in WHERE key_name='constant' ORDER BY key_name DESC.
- Fixed bug in SELECT DISTINCT ... ORDER BY DESC optimization.
- Fixed bug in ... HAVING 'GROUP_FUNCTION'(*xxx*) IS [NOT] NULL.
- Fixed bug in truncation operator for boolean full-text search.
- Allow value of --user=# option for **mysqld** to be specified as a numeric user ID.
- Fixed a bug where SQL_CALC_ROWS returned an incorrect value when used with one table and ORDER BY and with InnoDB tables.
- Fixed that SELECT 0 LIMIT 0 doesn't hang thread.
- Fixed some problems with USE/IGNORE INDEX when using many keys with the same start column.
- Don't use table scan with BerkeleyDB and InnoDB tables when we can use an index that covers the whole row.
- Optimized InnoDB sort-buffer handling to take less memory.
- Fixed bug in multiple-table DELETE and InnoDB tables.
- Fixed problem with TRUNCATE and InnoDB tables that produced the error Can't execute the given command because you have active locked tables or an active transaction.

- Added NO_UNSIGNED_SUBTRACTION to the set of flags that may be specified with the --sql-mode option for **mysqld**. It disables unsigned arithmetic rules when it comes to subtraction. (This makes MySQL 4.0 behave more like 3.23 with UNSIGNED columns).
- The result returned for all bit functions (|, <<, ...) is now of type unsigned integer.
- Added detection of nan values in MyISAM to make it possible to repair tables with nan in float or double columns.
- Fixed new bug in **myisamchk** where it didn't correctly update number of ``parts'' in the MyISAM index file.
- Changed to use **autoconf** 2.52 (from **autoconf** 2.13).
- Fixed optimization problem where the MySQL Server was in ``preparing'' state for a long time when selecting from an empty table which had contained a lot of rows.
- Fixed bug in complicated join with const tables. This fix also improves performance a bit when referring to another table from a const table.
- First pre-version of multiple-table UPDATE statement.
- Fixed bug in multiple-table DELETE.
- Fixed bug in SELECT CONCAT(argument_list) ... GROUP BY 1.
- INSERT ... SELECT did a full rollback in case of an error. Fixed so that we only roll back the last statement in the current transaction.
- Fixed bug with empty expression for boolean full-text search.
- Fixed core dump bug in updating full-text key from/to NULL.
- ODBC compatibility: Added BIT_LENGTH() function.
- Fixed core dump bug in GROUP BY BINARY column.
- Added support for NULL keys in HEAP tables.
- Use index for ORDER BY in queries of type: SELECT * FROM t WHERE key_part1=1 ORDER BY key_part1 DESC, key_part2 DESC
- Fixed bug in FLUSH QUERY CACHE.
- Added CAST() and CONVERT() functions. The CAST and CONVERT functions are nearly identical and mainly useful when you want to create a column with a specific type in a CREATE ... SELECT statement. For more information, read Section 12.7, “Cast Functions and Operators”.
- CREATE ... SELECT on DATE and TIME functions now create columns of the expected type.
- Changed order in which keys are created in tables.
- Added new columns Null and Index_type to SHOW INDEX output.
- Added --no-beep and --prompt options to **mysql** command-line client.
- New feature: management of user resources.

```
GRANT ... WITH MAX_QUERIES_PER_HOUR N1
               MAX_UPDATES_PER_HOUR N2
```

```
MAX_CONNECTIONS_PER_HOUR N3;
```

See Section 5.6.4, “Limiting Account Resources”.

- Added `mysql_secure_installation` to the `scripts/` directory.

D.3.25. Changes in release 4.0.1 (23 Dec 2001)

- Added `system` command to **mysql**.
- Fixed bug when `HANDLER` was used with some unsupported table type.
- **mysqldump** now puts `ALTER TABLE tbl_name DISABLE KEYS` and `ALTER TABLE tbl_name ENABLE KEYS` in the sql dump.
- Added **mysql_fix_extensions** script.
- Fixed stack overrun problem with `LOAD DATA FROM MASTER` on OSF/1.
- Fixed shutdown problem on HP-UX.
- Added `DES_ENCRYPT()` and `DES_DECRYPT()` functions.
- Added `FLUSH DES_KEY_FILE` statement.
- Added `--des-key-file` option to **mysqld**.
- `HEX(str)` now returns the characters in `str` converted to hexadecimal.
- Fixed problem with `GRANT` when using `lower_case_table_names=1`.
- Changed `SELECT ... IN SHARE MODE` to `SELECT ... LOCK IN SHARE MODE` (as in MySQL 3.23).
- A new query cache to cache results from identical `SELECT` queries.
- Fixed core dump bug on 64-bit machines when it got an incorrect communication packet.
- `MATCH ... AGAINST(... IN BOOLEAN MODE)` can now work without `FULLTEXT` index.
- Fixed slave to replicate from 3.23 master.
- Miscellaneous replication fixes/cleanup.
- Got shutdown to work on Mac OS X.
- Added `myisam/ft_dump` utility for low-level inspection of `FULLTEXT` indexes.
- Fixed bug in `DELETE ... WHERE ... MATCH ...`.
- Added support for `MATCH ... AGAINST(... IN BOOLEAN MODE)`. **Note**: You must rebuild your tables with `ALTER TABLE tbl_name TYPE=MyISAM` to be able to use boolean full-text search.
- `LOCATE()` and `INSTR()` are now case sensitive if either argument is a binary string.

- Changed RAND() initialization so that RAND(N) and RAND(N+1) are more distinct.
- Fixed core dump bug in UPDATE ... ORDER BY.
- In 3.23, INSERT INTO ... SELECT always had IGNORE enabled. Now MySQL stops (and possibly rolls back) by default in case of an error unless you specify IGNORE.
- Ignore DATA DIRECTORY and INDEX DIRECTORY directives on Windows.
- Added boolean full-text search code. It should be considered early alpha.
- Extended MODIFY and CHANGE in ALTER TABLE to accept the FIRST and AFTER keywords.
- Indexes are now used with ORDER BY on a whole InnoDB table.

D.3.26. Changes in release 4.0.0 (Oct 2001: Alpha)

- Added --xml option to **mysql** for producing XML output.
- Added full-text variables ft_min_word_len, ft_max_word_len, and ft_max_word_len_for_sort system variables.
- Added full-text variables ft_min_word_len, ft_max_word_len, and ft_max_word_len_for_sort variables to **myisamchk**.
- Added documentation for libmysqld, the embedded MySQL server library. Also added example programs (a **mysql** client and **mysqltest** test program) which use libmysqld.
- Removed all Gemini hooks from MySQL server.
- Removed my_thread_init() and my_thread_end() from mysql_com.h, and added mysql_thread_init() and mysql_thread_end() to mysql.h.
- Support for communication packets > 16MB. In 4.0.1 we extended MyISAM to be able to handle these.
- Secure connections (with SSL).
- Unsigned BIGINT constants now work. MIN() and MAX() now handle signed and unsigned BIGINT numbers correctly.
- New character set latin1_de which provides correct German sorting.
- STRCMP() now uses the current character set when doing comparisons, which means that the default comparison behavior now is case insensitive.
- TRUNCATE TABLE and DELETE FROM *tbl_name* are now separate functions. One bonus is that DELETE FROM *tbl_name* now returns the number of deleted rows, rather than zero.
- DROP DATABASE now executes a DROP TABLE on all tables in the database, which fixes a problem with InnoDB tables.
- Added support for UNION.
- Added support for multiple-table DELETE operations.
- A new HANDLER interface to MyISAM tables.

- Added support for `INSERT` on `MERGE` tables. Patch from Benjamin Pflugmann.
- Changed `WEEK(date,0)` to match the calendar in the USA.
- `COUNT(DISTINCT)` is about 30% faster.
- Speed up all internal list handling.
- Speed up `IS NULL`, `ISNULL()` and some other internal primitives.
- Full-text index creation now is much faster.
- Tree-like cache to speed up bulk inserts and `myisam_bulk_insert_tree_size` variable.
- Searching on packed (`CHAR`/`VARCHAR`) keys is now much faster.
- Optimized queries of type: `SELECT DISTINCT * FROM tbl_name ORDER by key_part1 LIMIT row_count`.
- `SHOW CREATE TABLE` now shows all table attributes.
- `ORDER BY ... DESC` can now use keys.
- `LOAD DATA FROM MASTER` ``automatically'' sets up a slave.
- Renamed **safe_mysqld** to **mysqld_safe** to make this name more in line with other MySQL scripts/commands.
- Added support for symbolic links to `MyISAM` tables. Symlink handling is now enabled by default for Windows.
- Added `SQL_CALC_FOUND_ROWS` and `FOUND_ROWS()`. This makes it possible to know how many rows a query would have returned without a `LIMIT` clause.
- Changed output format of `SHOW OPEN TABLES`.
- Allow `SELECT expression LIMIT ....`
- Added the `IDENTITY` variable as a synonym for the `LAST_INSERT_ID` variable (like Sybase).
- Added `ORDER BY` syntax to `UPDATE` and `DELETE`.
- `SHOW INDEXES` is now a synonym for `SHOW INDEX`.
- Added `ALTER TABLE tbl_name DISABLE KEYS` and `ALTER TABLE tbl_name ENABLE KEYS` commands.
- Allow use of `IN` as a synonym for `FROM` in `SHOW` commands.
- Implemented ``repair by sort'' for `FULLTEXT` indexes. `REPAIR TABLE`, `ALTER TABLE`, and `OPTIMIZE TABLE` for tables with `FULLTEXT` indexes are now up to 100 times faster.
- Allow standard SQL syntax `X'hexadecimal-number'`.
- Cleaned up global lock handling for `FLUSH TABLES WITH READ LOCK`.
- Fixed problem with `DATETIME = constant` in `WHERE` optimization.
- Added `--master-data` and `--no-autocommit` options to **mysqldump**. (Thanks to Brian Aker for this.)

- Added script `mysql_explain_log.sh` to distribution. (Thanks to mobile.de).

D.4. Changes in release 3.23.x (Recent; still supported)

Please note that since release 4.0 is now production level, only critical fixes are done in the 3.23 release series. You are recommended to upgrade when possible, to take advantage of all speed and feature improvements in 4.0. See Section 2.10.3, “Upgrading from Version 3.23 to 4.0”.

The 3.23 release has several major features that are not present in previous versions. We have added three new table types:

- `MyISAM`

  A new `ISAM` library which is tuned for SQL and supports large files.

- `InnoDB`

  A transaction-safe storage engine that supports row level locking, and many Oracle-like features.

- `BerkeleyDB` or `BDB`

  Uses the Berkeley DB library from Sleepycat Software to implement transaction-safe tables.

Note that only `MyISAM` is available in the standard binary distribution.

The 3.23 release also includes support for database replication between a master and many slaves, full-text indexing, and much more.

All new features are being developed in the 4.1.x and 5.0.x versions. Only serious bugfixes are added to 3.23.

D.4.1. Changes in release 3.23.59 (not released yet)

- Fixed an old bug in concurrent accesses to `MERGE` tables (even one `MERGE` table and `MyISAM` tables), that could've resulted in a crash or hang of the server. (Bug #2408 [http://bugs.mysql.com/2408])

- Fixed incorrect destruction of expression which led to crash of server on complex `AND`/`OR` expressions if query was ignored (either by a replication server because of `--replicate-*-table` rules, or by any MySQL server because of a syntax error). (Bug #3969 [http://bugs.mysql.com/3969], Bug #4494 [http://bugs.mysql.com/4494])

- Fixed problem with parsing complex queries on 64-bit architectures. (Bug #4204 [http://bugs.mysql.com/4204])

- Fixed a symlink vulnerability in the `mysqlbug` script (vulnerability ID CAN-2004-0381). (Bug #3284 [http://bugs.mysql.com/3284])

- Fixed bug in privilege checking of `ALTER TABLE RENAME`. (Bug #3270 [http://bugs.mysql.com/3270])

- Fixed bugs in `ACOS()`, `ASIN()` (Bug #2338 [http://bugs.mysql.com/2338]) and in `FLOOR()` (Bug #3051 [http://bugs.mysql.com/3051]). The cause of the problem is an overly strong optimization done by **gcc** in this case.

- Fixed bug in `INSERT ... SELECT` statements where, if a `NOT NULL` column is assigned a value of `NULL`, the following columns in the row might be assigned a value of zero. (Bug #2012 [http://bugs.mysql.com/2012])

- If a query was ignored on the slave (because of `--replicate-ignore-table` and other similar rules), the slave still checked if the query got the same error code (0, no error) as on the master. So if the master had an error on the query (for example, ``Duplicate entry'' in a multiple-row insert), then the slave stopped and warned that the error codes didn't match. This is a backport of the fix for MySQL 4.0. (Bug #797 [http://bugs.mysql.com/797])

- **mysqlbinlog** now asks for a password at console when the `-p/--password` option is used with no argument. This is how the other clients (**mysqladmin**, **mysqldump**..) behave. Note that one now has to use **mysqlbinlog -p<my_password>**; **mysqlbinlog -p <my_password>** does not work anymore (in other words, put no space after `-p`). (Bug #1595 [http://bugs.mysql.com/1595])

- On some 64-bit machines (some HP-UX and Solaris machines), a slave installed with the 64-bit MySQL binary could not connect to its master (it connected to itself instead). (Bug #1256 [http://bugs.mysql.com/1256], Bug #1381 [http://bugs.mysql.com/1381])

- Fixed a Windows-specific bug present since MySQL 3.23.57 and 3.23.58 that caused Windows slaves to crash when they started replication if a `master.info` file existed. (Bug #1720 [http://bugs.mysql.com/1720])

- Fixed bug in `ALTER TABLE RENAME`, when rename to the table with the same name in another database silently dropped destination table if it existed. (Bug #2628 [http://bugs.mysql.com/2628])

- Fixed potential memory overrun in `mysql_real_connect()` (which required a compromised DNS server and certain operating systems). (Bug #4017 [http://bugs.mysql.com/4017])

D.4.2. Changes in release 3.23.58 (11 Sep 2003)

- Fixed buffer overflow in password handling which could potentially be exploited by MySQL users with `ALTER` privilege on the `mysql.user` table to execute random code or to gain shell access with the UID of the **mysqld** process (thanks to Jedi/Sector One for spotting and reporting this bug).

- **mysqldump** now correctly quotes all identifiers when communicating with the server. This assures that during the dump process, **mysqldump** never sends queries to the server that result in a syntax error. This problem is **not** related to the **mysqldump** program's output, which was not changed. (Bug #1148 [http://bugs.mysql.com/1148])

- Fixed table/column grant handling: The proper sort order (from most specific to less specific, see Section 5.5.6, “Access Control, Stage 2: Request Verification”) was not honored. (Bug #928 [http://bugs.mysql.com/928])

- Fixed overflow bug in `MyISAM` and `ISAM` when a row is updated in a table with a large number of columns and at least one `BLOB/TEXT` column.

- Fixed MySQL so that field length (in C API) for the second column in `SHOW CREATE TABLE` is always larger than the data length. The only known application that was affected by the old behavior was Borland dbExpress, which truncated the output from the command. (Bug #1064 [http://bugs.mysql.com/1064])

- Fixed ISAM bug in MAX() optimization.
- Fixed Unknown error when doing ORDER BY on reference table which was used with NULL value on NOT NULL column. (Bug #479 [http://bugs.mysql.com/479])

D.4.3. Changes in release 3.23.57 (06 Jun 2003)

- Fixed problem in alarm handling that could cause problems when getting a packet that is too large.
- Fixed problem when installing MySQL as a service on Windows when two arguments were specified to **mysqld** (option file group name and service name).
- Fixed kill pid-of-mysqld to work on Mac OS X.
- SHOW TABLE STATUS displayed incorrect Row_format value for tables that have been compressed with **myisampack**. (Bug #427 [http://bugs.mysql.com/427])
- SHOW VARIABLES LIKE 'innodb_data_file_path' displayed only the name of the first data file. (Bug #468 [http://bugs.mysql.com/468])
- Fixed security problem where **mysqld** didn't allow one to UPDATE rows in a table even if one had a global UPDATE privilege and a database SELECT privilege.
- Fixed a security problem with SELECT and wildcarded select list, when user only had partial column SELECT privileges on the table.
- Fixed unlikely problem in optimizing WHERE clause with a constant expression such as in WHERE 1 AND (a=1 AND b=1).
- Fixed problem on IA-64 with timestamps that caused **mysqlbinlog** to fail.
- The default option for innodb_flush_log_at_trx_commit was changed from 0 to 1 to make InnoDB tables ACID by default. See Section 15.5, “InnoDB Startup Options”.
- Fixed problem with too many allocated alarms on slave when connecting to master many times (normally not a very critical error).
- Fixed a bug in replication of temporary tables. (Bug #183 [http://bugs.mysql.com/183])
- Fixed 64-bit bug that affected at least AMD hammer systems.
- Fixed a bug when doing LOAD DATA INFILE IGNORE: When reading the binary log, **mysqlbinlog** and the replication code read REPLACE instead of IGNORE. This could make the slave's table become different from the master's table. (Bug #218 [http://bugs.mysql.com/218])
- Fixed overflow bug in MyISAM when a row is inserted into a table with a large number of columns and at least one BLOB/TEXT column. Bug was caused by incorrect calculation of the needed buffer to pack data.
- The binary log was not locked during TRUNCATE *tbl_name* or DELETE FROM *tbl_name* statements, which could cause an INSERT to *tbl_name* to be written to the log before the TRUNCATE or DELETE statements.
- Fixed rare bug in UPDATE of InnoDB tables where one row could be updated multiple times.
- Produce an error for empty table and column names.

- Changed `PROCEDURE ANALYSE()` to report `DATE` instead of `NEWDATE`.
- Changed `PROCEDURE ANALYSE(#)` to restrict the number of values in an `ENUM` column to # also for string values.
- **mysqldump** no longer silently deletes the binary logs when invoked with the `--master-data` or `--first-slave` option; while this behavior was convenient for some users, others may suffer from it. Now you must explicitly ask for binary logs to be deleted by using the new `--delete-master-logs` option.
- Fixed a bug in **mysqldump** when it was invoked with the `--master-data` option: The `CHANGE MASTER TO` statements that were appended to the SQL dump had incorrect coordinates. (Bug #159 [http://bugs.mysql.com/159])

D.4.4. Changes in release 3.23.56 (13 Mar 2003)

- Fixed **mysqld** crash on extremely small values of `sort_buffer` variable.
- Fixed a bug in privilege system for `GRANT UPDATE` on the column level.
- Fixed a rare bug when using a date in `HAVING` with `GROUP BY`.
- Fixed checking of random part of `WHERE` clause. (Bug #142 [http://bugs.mysql.com/142])
- Fixed MySQL (and **myisamchk**) crash on artificially corrupted `.MYI` files.
- Security enhancement: **mysqld** no longer reads options from world-writable config files.
- Security enhancement: **mysqld** and **safe_mysqld** now use only the first `--user` option specified on the command line. (Normally this comes from `/etc/my.cnf`)
- Security enhancement: Don't allow `BACKUP TABLE` to overwrite existing files.
- Fixed unlikely deadlock bug when one thread did a `LOCK TABLE` and another thread did a `DROP TABLE`. In this case one could do a `KILL` on one of the threads to resolve the deadlock.
- `LOAD DATA INFILE` was not replicated by slave if `replicate_*_table` was set on the slave.
- Fixed a bug in handling `CHAR(0)` columns that could cause incorrect results from the query.
- Fixed a bug in `SHOW VARIABLES` on 64-bit platforms. The bug was caused by incorrect declaration of variable `server_id`.
- The Comment column in `SHOW TABLE STATUS` now reports that it can contain `NULL` values (which is the case for a crashed `.frm` file).
- Fixed the `rpl_rotate_logs` test to not fail on certain platforms (such as Mac OS X) due to a too-long file name (changed `slave-master-info.opt` to `.slave-mi`).
- Fixed a problem with `BLOB NOT NULL` columns used with `IS NULL`.
- Fixed bug in `MAX()` optimization in `MERGE` tables.
- Better `RAND()` initialization for new connections.
- Fixed bug with connect timeout. This bug was manifested on OS's with `poll()` system call, which resulted in timeout the value specified as it was executed in both `select()` and `poll()`.

- Fixed bug in `SELECT * FROM table WHERE datetime1 IS NULL OR datetime2 IS NULL`.
- Fixed bug in using aggregate functions as argument for `INTERVAL`, `CASE`, `FIELD`, `CONCAT_WS`, `ELT` and `MAKE_SET` functions.
- When running with `--lower-case-table-names=1` (default on Windows) and you had tables or databases with mixed case on disk, then executing `SHOW TABLE STATUS` followed with `DROP DATABASE` or `DROP TABLE` could fail with `Errcode 13`.
- Fixed bug in logging to binary log (which affects replication) a query that inserts a `NULL` in an `auto_increment` field and also uses `LAST_INSERT_ID()`.
- Fixed bug in **mysqladmin --relative**.
- On some 64-bit systems, `show status` reported a strange number for `Open_files` and `Open_streams`.

D.4.5. Changes in release 3.23.55 (23 Jan 2003)

- Fixed double `free`'d pointer bug in `mysql_change_user()` handling, that enabled a specially hacked version of MySQL client to crash **mysqld**. **Note** that you must log in to the server by using a valid user account to be able to exploit this bug.
- Fixed bug with the `--slow-log` when logging an administrator command (like `FLUSH TABLES`).
- Fixed bug in `GROUP BY` when used on BLOB column with `NULL` values.
- Fixed a bug in handling `NULL` values in `CASE ... WHEN ...`.
- Bugfix for `--chroot` (see Section D.4.6, “Changes in release 3.23.54 (05 Dec 2002)”) is reverted. Unfortunately, there is no way to make it to work, without introducing backward-incompatible changes in `my.cnf`. Those who need `--chroot` functionality, should upgrade to MySQL 4.0. (The fix in the 4.0 branch did not break backward-compatibility).
- Make `--lower-case-table-names` default on Mac OS X as the default filesystem (HFS+) is case insensitive.
- Fixed a bug in `scripts/mysqld_safe.sh` in `NOHUP_NICENESS` testing.
- Transactions in `AUTOCOMMIT=0` mode didn't rotate binary log.
- Fixed a bug in `scripts/make_binary_distribution` that resulted in a remaining `@HOSTNAME@` variable instead of replacing it with the correct path to the `hostname` binary.
- Fixed a very unlikely bug that could cause `SHOW PROCESSLIST` to core dump in pthread_mutex_unlock() if a new thread was connecting.
- Forbid `SLAVE STOP` if the thread executing the query has locked tables. This removes a possible deadlock situation.

D.4.6. Changes in release 3.23.54 (05 Dec 2002)

- Fixed a bug, that allowed to crash **mysqld** with a specially crafted packet.
- Fixed a rare crash (double `free`'d pointer) when altering a temporary table.
- Fixed buffer overrun in `libmysqlclient` library that allowed malicious MySQL server to crash the client application.
- Fixed security-related bug in `mysql_change_user()` handling. All users are strongly recommended to upgrade to the version 3.23.54.
- Fixed bug that prevented `--chroot` command-line option of **mysqld** from working.
- Fixed bug that made `OPTIMIZE TABLE` to corrupt the table under some rare circumstances.
- Fixed **mysqlcheck** so it can deal with table names containing dashes.
- Fixed shutdown problem on Mac OS X.
- Fixed bug with comparing an indexed `NULL` field with `<=> NULL`.
- Fixed bug that caused `IGNORE INDEX` and `USE INDEX` sometimes to be ignored.
- Fixed rare core dump problem in complicated `GROUP BY` queries that didn't return any result.
- Fixed a bug where `MATCH ... AGAINST () >=0` was treated as if it was `>`.
- Fixed core dump in `SHOW PROCESSLIST` when running with an active slave (unlikely timing bug).
- Make it possible to use multiple MySQL servers on Windows (code backported from 4.0.2).
- One can create `TEMPORARY MERGE` tables now.
- Fixed that `--core-file` works on Linux (at least on kernel 2.4.18).
- Fixed a problem with `BDB` and `ALTER TABLE`.
- Fixed reference to freed memory when doing complicated `GROUP BY ... ORDER BY` queries. Symptom was that **mysqld** died in function `send_fields`.
- Allocate heap rows in smaller blocks to get better memory usage.
- Fixed memory allocation bug when storing `BLOB` values in internal temporary tables used for some (unlikely) `GROUP BY` queries.
- Fixed a bug in key optimizing handling where the expression `WHERE` *`col_name`* `=` *`key_col_name`* was calculated as true for `NULL` values.
- Fixed core dump bug when doing `LEFT JOIN ... WHERE` *`key_columnn`*`=NULL`.
- Fixed `MyISAM` crash when using dynamic-row tables with huge numbers of packed fields.
- Updated source tree to be built using **automake** 1.5 and **libtool** 1.4.

D.4.7. Changes in release 3.23.53 (09 Oct 2002)

- Fixed crash when `SHOW INNODB STATUS` was used and `skip-innodb` was defined.

- Fixed possible memory corruption bug in binary log file handling when slave rotated the logs (only affected 3.23, not 4.0).
- Fixed problem in `LOCK TABLES` on Windows when one connects to a database that contains uppercase letters.
- Fixed that `--skip-show-database` doesn't reset the `--port` option.
- Small fix in **safe_mysqld** for some shells.
- Fixed that `FLUSH STATUS` doesn't reset `delayed_insert_threads`.
- Fixed core dump bug when using the `BINARY` cast on a `NULL` value.
- Fixed race condition when someone did a `GRANT` at the same time a new user logged in or did a `USE database`.
- Fixed bug in `ALTER TABLE` and `RENAME TABLE` when running with `-O lower_case_table_names=1` (typically on Windows) when giving the table name in uppercase.
- Fixed that `-O lower_case_table_names=1` also converts database names to lowercase.
- Fixed unlikely core dump with `SELECT ... ORDER BY ... LIMIT`.
- Changed `AND/OR` to report that they can return NULL. This fixes a bug in `GROUP BY` on `AND/OR` expressions that return `NULL`.
- Fixed a bug that `OPTIMIZE TABLE` of locked and modified `MyISAM` table, reported table corruption.
- Fixed a `BDB`-related `ALTER TABLE` bug with dropping a column and shutting down immediately thereafter.
- Fixed problem with `configure ... --localstatedir=...`.
- Fixed problem with `UNSIGNED BIGINT` on AIX (again).
- Fixed bug in pthread_mutex_trylock() on HPUX 11.0.
- Multi-threaded stress tests for `InnoDB`.

D.4.8. Changes in release 3.23.52 (14 Aug 2002)

- Wrap `BEGIN`/`COMMIT` around transaction in the binary log. This makes replication honor transactions.
- Fixed security bug when having an empty database name in the `user.db` table.
- Changed initialization of `RAND()` to make it less predicatable.
- Fixed problem with `GROUP BY` on result with expression that created a `BLOB` field.
- Fixed problem with `GROUP BY` on columns that have `NULL` values. To solve this we now create an `MyISAM` temporary table when doing a `GROUP BY` on a possible `NULL` item. From MySQL 4.0.5 we can use in memory `HEAP` tables for this case.

- Fixed problem with privilege tables when downgrading from 4.0.2 to 3.23.
- Fixed thread bug in `SLAVE START`, `SLAVE STOP` and automatic repair of `MyISAM` tables that could cause table cache to be corrupted.
- Fixed possible thread related key-cache-corruption problem with `OPTIMIZE TABLE` and `REPAIR TABLE`.
- Added name of 'administrator command' logs.
- Fixed bug with creating an auto-increment value on second part of a `UNIQUE()` key where first part could contain `NULL` values.
- Don't write slave-timeout reconnects to the error log.
- Fixed bug with slave net read timeouting
- Fixed a core-dump bug with `MERGE` tables and `MAX()` function.
- Fixed bug in `ALTER TABLE` with `BDB` tables.
- Fixed bug when logging `LOAD DATA INFILE` to binary log with no active database.
- Fixed a bug in range optimizer (causing crashes).
- Fixed possible problem in replication when doing `DROP DATABASE` on a database with `InnoDB` tables.
- Fixed `mysql_info()` to return 0 for `Duplicates` value when using `INSERT DELAYED IGNORE`.
- Added `-DHAVE_BROKEN_REALPATH` to the Mac OS X (darwin) compile options in `configure.in` to fix a failure under high load.

D.4.9. Changes in release 3.23.51 (31 May 2002)

- Fix bug with closing tags missing slash for **mysqldump** XML output.
- Remove endspace from `ENUM` values. (This fixed a problem with `SHOW CREATE TABLE`.)
- Fixed bug in `CONCAT_WS()` that cut the result.
- Changed name of server variables `Com_show_master_stat` to `Com_show_master_status` and `Com_show_slave_stat` to `Com_show_slave_status`.
- Changed handling of `gethostbyname()` to make the client library thread-safe even if `gethostbyname_r` doesn't exist.
- Fixed core-dump problem when giving a wrong password string to `GRANT`.
- Fixed bug in `DROP DATABASE` with symlinked directory.
- Fixed optimization problem with `DATETIME` and value outside `DATETIME` range.
- Removed Sleepycat's `BDB` doc files from the source tree, as they're not needed (MySQL covers `BDB` in its own documentation).

- Fixed MIT-pthreads to compile with `glibc` 2.2 (needed for **make dist**).
- Fixed the `FLOAT(X+1,X)` is not converted to `FLOAT(X+2,X)`. (This also affected `DECIMAL`, `DOUBLE` and `REAL` types)
- Fixed the result from `IF()` is case in-sensitive if the second and third arguments are case sensitive.
- Fixed core dump problem on OSF/1 in `gethostbyname_r`.
- Fixed that underflowed decimal fields are not zero filled.
- If we get an overflow when inserting `'+11111'` for `DECIMAL(5,0) UNSIGNED` columns, we just drop the sign.
- Fixed optimization bug with `ISNULL(`*`expression_which_cannot_be_null`*`)` and `ISNULL(`*`constant_expression`*`)`.
- Fixed host lookup bug in the `glibc` library that we used with the 3.23.50 Linux-x86 binaries.

D.4.10. Changes in release 3.23.50 (21 Apr 2002)

- Fixed buffer overflow problem if someone specified a too-long `datadir` parameter to **mysqld**.
- Add missing `<row>` tags for **mysqldump** XML output.
- Fixed problem with `crash-me` and **gcc** 3.0.4.
- Fixed that `@@unknown_variable` doesn't hang server.
- Added `@@VERSION` as a synonym for `VERSION()`.
- `SHOW VARIABLES LIKE '`*`xxx`*`'` is now case-insensitive.
- Fixed timeout for `GET_LOCK()` on HP-UX with DCE threads.
- Fixed memory allocation bug in the glibc library used to build Linux binaries, which caused **mysqld** to die in `free()`.
- Fixed `SIGINT` and `SIGQUIT` problems in **mysql**.
- Fixed bug in character table converts when used with big (larger than 64KB) strings.
- `InnoDB` now retains foreign key constraints through `ALTER TABLE` and `CREATE/DROP INDEX`.
- `InnoDB` now allows foreign key constraints to be added through the `ALTER TABLE` syntax.
- `InnoDB` tables can now be set to automatically grow in size (autoextend).
- Our Linux RPMS and binaries are now compiled with **gcc** 3.0.4, which should make them a bit faster.
- Fixed some buffer overflow problems when reading startup parameters.
- Because of problems on shutdown we have now disabled named pipes on Windows by default. One can enable named pipes by starting **mysqld** with `--enable-named-pipe`.

- Fixed bug when using `WHERE key_column = 'J' or key_column='j'`.
- Fixed core-dump bug when using `--log-bin` with `LOAD DATA INFILE` without an active database.
- Fixed bug in `RENAME TABLE` when used with `lower_case_table_names=1` (default on Windows).
- Fixed unlikely core-dump bug when using `DROP TABLE` on a table that was in use by a thread that also used queries on only temporary tables.
- Fixed problem with `SHOW CREATE TABLE` and `PRIMARY KEY` when using 32 indexes.
- Fixed that one can use `SET PASSWORD` for the anonymous user.
- Fixed core dump bug when reading client groups from option files using `mysql_options()`.
- Memory leak (16 bytes per every **corrupted** table) closed.
- Fixed binary builds to use `--enable-local-infile`.
- Update source to work with new version of **bison**.
- Updated shell scripts to now agree with new POSIX standard.
- Fixed bug where `DATE_FORMAT()` returned empty string when used with `GROUP BY`.

D.4.11. Changes in release 3.23.49 (14 Feb 2002)

- For a `MERGE` table, `DELETE FROM merge_table` used without a `WHERE` clause no longer clears the mapping for the table by emptying the `.MRG` file. Instead, it deletes records from the mapped tables.
- Don't give warning for a statement that is only a comment; this is needed for **mysqldump --disable-keys** to work.
- Fixed unlikely caching bug when doing a join without keys. In this case, the last used field for a table always returned `NULL`.
- Added options to make `LOAD DATA LOCAL INFILE` more secure.
- MySQL binary release 3.23.48 for Linux contained a new `glibc` library, which has serious problems under high load and Red Hat 7.2. The 3.23.49 binary release doesn't have this problem.
- Fixed shutdown problem on NT.

D.4.12. Changes in release 3.23.48 (07 Feb 2002)

- Added `--xml` option to **mysqldump** for producing XML output.
- Changed to use **autoconf** 2.52 (from **autoconf** 2.13)
- Fixed bug in complicated join with `const` tables.

- Added internal safety checks for `InnoDB`.
- Some `InnoDB` variables were always shown in `SHOW VARIABLES` as `OFF` on high-byte-first systems (like SPARC).
- Fixed problem with one thread using an `InnoDB` table and another thread doing an `ALTER TABLE` on the same table. Before that, **mysqld** could crash with an assertion failure in `row0row.c`, line 474.
- Tuned the `InnoDB` SQL optimizer to favor index searches more often over table scans.
- Fixed a performance problem with `InnoDB` tables when several large `SELECT` queries are run concurrently on a multiprocessor Linux computer. Large CPU-bound `SELECT` queries now also generally run faster on all platforms.
- If MySQL binary logging is used, `InnoDB` now prints after crash recovery the latest MySQL binary log name and the offset `InnoDB` was able to recover to. This is useful, for example, when resynchronizing a master and a slave database in replication.
- Added better error messages to help in installation problems of `InnoDB` tables.
- It is now possible to recover MySQL temporary tables that have become orphaned inside the `InnoDB` tablespace.
- `InnoDB` now prevents a `FOREIGN KEY` declaration where the signedness is not the same in the referencing and referenced integer columns.
- Calling `SHOW CREATE TABLE` or `SHOW TABLE STATUS` could cause memory corruption and make **mysqld** crash. Especially at risk was **mysqldump**, because it frequently calls `SHOW CREATE TABLE`.
- If inserts to several tables containing an `AUTO_INCREMENT` column were wrapped inside one `LOCK TABLES`, `InnoDB` asserted in `lock0lock.c`.
- In 3.23.47 we allowed several `NULL` values in a `UNIQUE` secondary index for an `InnoDB` table. But `CHECK TABLE` was not relaxed: it reports the table as corrupt. `CHECK TABLE` no longer complains in this situation.
- `SHOW GRANTS` now shows `REFERENCES` instead of `REFERENCE`.

D.4.13. Changes in release 3.23.47 (27 Dec 2001)

- Fixed bug when using the following construct: `SELECT ... WHERE` *`key=@var_name`* `OR` *`key=@var_name2`*
- Restrict `InnoDB` keys to 500 bytes.
- `InnoDB` now supports `NULL` in keys.
- Fixed shutdown problem on HP-UX. (Introduced in 3.23.46)
- Fixed core dump bug in replication when using `SELECT RELEASE_LOCK()`.
- Added new statement: `DO` *`expr`*`[,`*`expr`*`]...`
- Added `slave-skip-errors` option.

- Added statistics variables for all MySQL commands. (`SHOW STATUS` is now much longer.)
- Fixed default values for `InnoDB` tables.
- Fixed that `GROUP BY expr DESC` works.
- Fixed bug when using `t1 LEFT JOIN t2 ON t2.key=constant`.
- **mysql_config** now also works with binary (relocated) distributions.

D.4.14. Changes in release 3.23.46 (29 Nov 2001)

- Fixed problem with aliased temporary table replication.
- `InnoDB` and `BDB` tables now use index when doing an `ORDER BY` on the whole table.
- Fixed bug where one got an empty set instead of a DEADLOCK error when using `BDB` tables.
- One can now kill `ANALYZE TABLE`, `REPAIR TABLE`, and `OPTIMIZE TABLE` when the thread is waiting to get a lock on the table.
- Fixed race condition in `ANALYZE TABLE`.
- Fixed bug when joining with caching (unlikely to happen).
- Fixed race condition when using the binary log and `INSERT DELAYED` which could cause the binary log to have rows that were not yet written to `MyISAM` tables.
- Changed caching of binary log to make replication slightly faster.
- Fixed bug in replication on Mac OS X.

D.4.15. Changes in release 3.23.45 (22 Nov 2001)

- `(UPDATE|DELETE) ...WHERE MATCH` bugfix.
- shutdown should now work on Darwin (Mac OS X).
- Fixed core dump when repairing corrupted packed `MyISAM` files.
- `--core-file` now works on Solaris.
- Fix a bug which could cause `InnoDB` to complain if it cannot find free blocks from the buffer cache during recovery.
- Fixed bug in `InnoDB` insert buffer B-tree handling that could cause crashes.
- Fixed bug in `InnoDB` lock timeout handling.
- Fixed core dump bug in `ALTER TABLE` on a `TEMPORARY InnoDB` table.
- Fixed bug in `OPTIMIZE TABLE` that reset index cardinality if it was up to date.
- Fixed problem with `t1 LEFT_JOIN t2 ... WHERE t2.date_column IS NULL` when date_column was declared as `NOT NULL`.

- Fixed bug with BDB tables and keys on BLOB columns.
- Fixed bug in MERGE tables on OS with 32-bit file pointers.
- Fixed bug in TIME_TO_SEC() when using negative values.

D.4.16. Changes in release 3.23.44 (31 Oct 2001)

- Fixed Rows_examined count in slow query log.
- Fixed bug when using a reference to an AVG() column in HAVING.
- Fixed that date functions that require correct dates, like DAYOFYEAR(column), return NULL for 0000-00-00 dates.
- Fixed bug in const-propagation when comparing columns of different types. (SELECT * FROM date_col="2001-01-01" and date_col=time_col)
- Fixed bug that caused error message Can't write, because of unique constraint with some GROUP BY queries.
- Fixed problem with sjis character strings used within quoted table names.
- Fixed core dump when using CREATE ... FULLTEXT keys with other storage engines than MyISAM.
- Don't use signal() on Windows because this appears to not be 100% reliable.
- Fixed bug when doing WHERE col_name=NULL on an indexed column that had NULL values.
- Fixed bug when doing LEFT JOIN ... ON (col_name = constant) WHERE col_name = constant.
- When using replications, aborted queries that contained % could cause a core dump.
- TCP_NODELAY was not used on some systems. (Speed problem.)
- Applied portability fixes for OS/2. (Patch by Yuri Dario.)

The following changes are for InnoDB tables:

- Add missing InnoDB variables to SHOW VARIABLES.
- Foreign key checking is now done for InnoDB tables.
- DROP DATABASE now works also for InnoDB tables.
- InnoDB now supports data files and raw disk partitions bigger than 4GB on those operating systems that have big files.
- InnoDB calculates better table cardinality estimates for the MySQL optimizer.
- Accent characters in the default character set latin1 are ordered according to the MySQL ordering.

Note: If you are using `latin1` and have inserted characters whose code is greater than 127 into an indexed `CHAR` column, you should run `CHECK TABLE` on your table when you upgrade to 3.23.44, and drop and reimport the table if `CHECK TABLE` reports an error!

- A new `my.cnf` parameter, `innodb_thread_concurrency`, helps in performance tuning in heavily concurrent environments.
- A new `my.cnf` parameter, `innodb_fast_shutdown`, speeds up server shutdown.
- A new `my.cnf` parameter, `innodb_force_recovery`, helps to save your data in case the disk image of the database becomes corrupt.
- `innodb_monitor` has been improved and a new `innodb_table_monitor` added.
- Increased maximum key length from 500 to 7000 bytes.
- Fixed a bug in replication of `AUTO_INCREMENT` columns with multiple-line inserts.
- Fixed a bug when the case of letters changes in an update of an indexed secondary column.
- Fixed a hang when there are more than 24 data files.
- Fixed a crash when `MAX(col)` is selected from an empty table, and `col` is not the first column in a multi-column index.
- Fixed a bug in purge which could cause crashes.

D.4.17. Changes in release 3.23.43 (04 Oct 2001)

- Fixed a bug in `INSERT DELAYED` and `FLUSH TABLES` introduced in 3.23.42.
- Fixed unlikely bug, which returned non-matching rows, in `SELECT` with many tables and multi-column indexes and 'range' type.
- Fixed an unlikely core dump bug when doing `EXPLAIN SELECT` when using many tables and `ORDER BY`.
- Fixed bug in `LOAD DATA FROM MASTER` when using table with `CHECKSUM=1`.
- Added unique error message when a DEADLOCK occurs during a transaction with `BDB` tables.
- Fixed problem with `BDB` tables and `UNIQUE` columns defined as `NULL`.
- Fixed problem with **myisampack** when using pre-space filled `CHAR` columns.
- Applied patch from Yuri Dario for OS/2.
- Fixed bug in `--safe-user-create`.

D.4.18. Changes in release 3.23.42 (08 Sep 2001)

- Fixed problem when using `LOCK TABLES` and `BDB` tables.

- Fixed problem with `REPAIR TABLE` on `MyISAM` tables with row lengths in the range from 65517 to 65520 bytes.
- Fixed rare hang when doing **mysqladmin shutdown** when there was a lot of activity in other threads.
- Fixed problem with `INSERT DELAYED` where delayed thread could be hanging on `upgrading locks` for no apparent reason.
- Fixed problem with **myisampack** and `BLOB`.
- Fixed problem when one edited `.MRG` tables by hand. (Patch from Benjamin Pflugmann).
- Enforce that all tables in a `MERGE` table come from the same database.
- Fixed bug with `LOAD DATA INFILE` and transactional tables.
- Fix bug when using `INSERT DELAYED` with wrong column definition.
- Fixed core dump during `REPAIR TABLE` of some particularly broken tables.
- Fixed bug in `InnoDB` and `AUTO_INCREMENT` columns.
- Fixed bug in `InnoDB` and `RENAME TABLE` columns.
- Fixed critical bug in `InnoDB` and `BLOB` columns. If you have used `BLOB` columns larger than 8000 bytes in an `InnoDB` table, it is necessary to dump the table with **mysqldump**, drop it and restore it from the dump.
- Applied large patch for OS/2 from Yuri Dario.
- Fixed problem with `InnoDB` when one could get the error `Can't execute the given command...` even when no transaction was active.
- Applied some minor fixes that concern Gemini.
- Use real arithmetic operations even in integer context if not all arguments are integers. (Fixes uncommon bug in some integer contexts).
- Don't force everything to lowercase on Windows. (To fix problem with Windows and `ALTER TABLE`.) Now `--lower_case_table_names` also works on Unix.
- Fixed that automatic rollback is done when thread end doesn't lock other threads.

D.4.19. Changes in release 3.23.41 (11 Aug 2001)

- Added `--sql-mode=`*`value`*`[,`*`value`*`[,`*`value`*`]]` option to **mysqld**. See Section 5.2.1, “**mysqld** Command-Line Options”.
- Fixed possible problem with `shutdown` on Solaris where the `.pid` file wasn't deleted.
- `InnoDB` now supports < 4GB rows. The former limit was 8000 bytes.
- The `doublewrite` file flush method is used in `InnoDB`. It reduces the need for Unix `fsync()` calls to a fraction and improves performance on most Unix flavors.
- You can now use the `InnoDB` Monitor to print a lot of `InnoDB` state information, including locks,

to the standard output. This is useful in performance tuning.

- Several bugs which could cause hangs in `InnoDB` have been fixed.
- Split `record_buffer` to `record_buffer` and `record_rnd_buffer`. To make things compatible to previous MySQL versions, if `record_rnd_buffer` is not set, then it takes the value of `record_buffer`.
- Fixed optimizing bug in `ORDER BY` where some `ORDER BY` parts where wrongly removed.
- Fixed overflow bug with `ALTER TABLE` and `MERGE` tables.
- Added prototypes for `my_thread_init()` and `my_thread_end()` to `mysql_com.h`
- Added `--safe-user-create` option to **mysqld**.
- Fixed bug in `SELECT DISTINCT ... HAVING` that caused error message `Can't find record in #...`

D.4.20. Changes in release 3.23.40 (18 Jul 2001)

- Fixed problem with `--low-priority-updates` and `INSERT` statements.
- Fixed bug in slave thread when under some rare circumstances it could get 22 bytes ahead on the offset in the master.
- Added `slave_net_timeout` for replication.
- Fixed problem with `UPDATE` and `BDB` tables.
- Fixed hard bug in `BDB` tables when using key parts.
- Fixed problem when using `GRANT FILE ON database.* ...`; previously we added the `DROP` privilege for the database.
- Fixed `DELETE FROM tbl_name ... LIMIT 0` and `UPDATE FROM tbl_name ... LIMIT 0`, which acted as though the `LIMIT` clause was not present (they deleted or updated all selected rows).
- `CHECK TABLE` now checks whether an `AUTO_INCREMENT` column contains the value 0.
- Sending a `SIGHUP` to **mysqld** now only flushes the logs, but does not reset the replication.
- Fixed parser to allow floats of type `1.0e1` (no sign after `e`).
- Option `--force` to **myisamchk** now also updates states.
- Added option `--warnings` to **mysqld**. Now **mysqld** prints the error `Aborted connection` only if this option is used.
- Fixed problem with `SHOW CREATE TABLE` when you didn't have a `PRIMARY KEY`.
- Properly fixed the rename of `innodb_unix_file_flush_method` variable to `innodb_flush_method`.
- Fixed bug when converting `BIGINT UNSIGNED` to `DOUBLE`. This caused a problem when doing comparisons with `BIGINT` values outside of the signed range.

- Fixed bug in BDB tables when querying empty tables.
- Fixed a bug when using COUNT(DISTINCT) with LEFT JOIN and there weren't any matching rows.
- Removed all documentation referring to the GEMINI table type. GEMINI is not released under an Open Source license.

D.4.21. Changes in release 3.23.39 (12 Jun 2001)

- The AUTO_INCREMENT sequence wasn't reset when dropping and adding an AUTO_INCREMENT column.
- CREATE ... SELECT now creates non-unique indexes delayed.
- Fixed problem where LOCK TABLES *tbl_name* READ followed by FLUSH TABLES put an exclusive lock on the table.
- REAL @variable values were represented with only 2 digits when converted to strings.
- Fixed problem that client ``hung'' when LOAD TABLE FROM MASTER failed.
- **myisamchk --fast --force** no longer repairs tables that only had the open count wrong.
- Added functions to handle symbolic links to make life easier in 4.0.
- We are now using the -lcma thread library on HP-UX 10.20 so that MySQL is more stable on HP-UX.
- Fixed problem with IF() and number of decimals in the result.
- Fixed date-part extraction functions to work with dates where day and/or month is 0.
- Extended argument length in option files from 256 to 512 chars.
- Fixed problem with shutdown when INSERT DELAYED was waiting for a LOCK TABLE.
- Fixed core dump bug in InnoDB when tablespace was full.
- Fixed problem with MERGE tables and big tables (larger than 4GB) when using ORDER BY.

D.4.22. Changes in release 3.23.38 (09 May 2001)

- Fixed a bug when SELECT from MERGE table sometimes results in incorrectly ordered rows.
- Fixed a bug in REPLACE() when using the ujis character set.
- Applied Sleepycat BDB patches 3.2.9.1 and 3.2.9.2.
- Added --skip-stack-trace option to **mysqld**.
- CREATE TEMPORARY now works with InnoDB tables.
- InnoDB now promotes sub keys to whole keys.

- Added option CONCURRENT to LOAD DATA.
- Better error message when slave max_allowed_packet is too low to read a very long log event from the master.
- Fixed bug when too many rows were removed when using SELECT DISTINCT ... HAVING.
- SHOW CREATE TABLE now returns TEMPORARY for temporary tables.
- Added Rows_examined to slow query log.
- Fixed problems with function returning empty string when used together with a group function and a WHERE that didn't match any rows.
- New program **mysqlcheck**.
- Added database name to output for administrative commands like CHECK TABLE, REPAIR TABLE, OPTIMIZE TABLE.
- Lots of portability fixes for InnoDB.
- Changed optimizer so that queries like SELECT * FROM *tbl_name*, *tbl_name2* ... ORDER BY *key_part1* LIMIT *row_count* use an index on *key_part1* instead of filesort.
- Fixed bug when doing LOCK TABLE *to_table* WRITE,...; INSERT INTO *to_table*... SELECT ... when *to_table* was empty.
- Fixed bug with LOCK TABLE and BDB tables.

D.4.23. Changes in release 3.23.37 (17 Apr 2001)

- Fixed a bug when using MATCH() in HAVING clause.
- Fixed a bug when using HEAP tables with LIKE.
- Added --mysql-version option to **safe_mysqld**
- Changed INNOBASE to InnoDB (because the INNOBASE name was in use). All **configure** options and **mysqld** start options now use innodb instead of innobase. This means that before upgrading to this version, you have to change any configuration files where you have used innobase options!
- Fixed bug when using indexes on CHAR(255) NULL columns.
- Slave threads now start even if master-host is not set, as long as server-id is set and valid master.info is present.
- Partial updates (terminated with kill) are now logged with a special error code to the binary log. Slave refuses to execute them if the error code indicates the update was terminated abnormally, and has to be recovered with SET SQL_SLAVE_SKIP_COUNTER=1; SLAVE START after a manual sanity check/correction of data integrity.
- Fixed bug that erroneously logged a drop of internal temporary table on thread termination to the binary log --- this bug affected replication.
- Fixed a bug in REGEXP on 64-bit machines.

- UPDATE and DELETE with WHERE unique_key_part IS NULL didn't update/delete all rows.
- Disabled INSERT DELAYED for tables that support transactions.
- Fixed bug when using date functions on TEXT/BLOB column with wrong date format.
- UDFs now also work on Windows. (Patch by Ralph Mason.)
- Fixed bug in ALTER TABLE and LOAD DATA INFILE that disabled key-sorting. These commands should now be faster in most cases.
- Fixed performance bug where reopened tables (tables that had been waiting for FLUSH or REPAIR TABLE) would not use indexes for the next query.
- Fixed problem with ALTER TABLE to InnoDB tables on FreeBSD.
- Added **mysqld** variables myisam_max_sort_file_size and myisam_max_extra_sort_file_size.
- Initialize signals early to avoid problem with signals in InnoDB.
- Applied patch for the tis620 character set to make comparisons case-independent and to fix a bug in LIKE for this character set. **Note**: All tables that use the tis620 character set must be fixed with **myisamchk -r** or REPAIR TABLE!
- Added --skip-safemalloc option to **mysqld**.

D.4.24. Changes in release 3.23.36 (27 Mar 2001)

- Fixed a bug that allowed use of database names containing a '.' character. This fixes a serious security issue when **mysqld** is run as root.
- Fixed bug when thread creation failed (could happen when doing a **lot** of connections in a short time).
- Fixed some problems with FLUSH TABLES and TEMPORARY tables. (Problem with freeing the key cache and error Can't reopen table....)
- Fixed a problem in InnoDB with other character sets than latin1 and another problem when using many columns.
- Fixed bug that caused a core dump when using a very complex query involving DISTINCT and summary functions.
- Added SET TRANSACTION ISOLATION LEVEL ...
- Added SELECT ... FOR UPDATE.
- Fixed bug where the number of affected rows was not returned when MySQL was compiled without transaction support.
- Fixed a bug in UPDATE where keys weren't always used to find the rows to be updated.
- Fixed a bug in CONCAT_WS() where it returned incorrect results.
- Changed CREATE ... SELECT and INSERT ... SELECT to not allow concurrent inserts as

this could make the binary log hard to repeat. (Concurrent inserts are enabled if you are not using the binary or update log.)

- Changed some macros to be able to use fast mutex with `glibc` 2.2.

D.4.25. Changes in release 3.23.35 (15 Mar 2001)

- Fixed newly introduced bug in `ORDER BY`.
- Fixed wrong define `CLIENT_TRANSACTIONS`.
- Fixed bug in `SHOW VARIABLES` when using `INNOBASE` tables.
- Setting and using user variables in `SELECT DISTINCT` didn't work.
- Tuned `SHOW ANALYZE` for small tables.
- Fixed handling of arguments in the benchmark script `run-all-tests`.

D.4.26. Changes in release 3.23.34a (11 Mar 2001)

- Added extra files to the distribution to allow `INNOBASE` support to be compiled.

D.4.27. Changes in release 3.23.34 (10 Mar 2001)

- Added the `INNOBASE` storage engine and the `BDB` storage engine to the MySQL source distribution.
- Updated the documentation about `GEMINI` tables.
- Fixed a bug in `INSERT DELAYED` that caused threads to hang when inserting `NULL` into an `AUTO_INCREMENT` column.
- Fixed a bug in `CHECK TABLE` / `REPAIR TABLE` that could cause a thread to hang.
- Fixed problem that `REPLACE` would not replace a row that conflicts with an `AUTO_INCREMENT` generated key.
- **mysqld** now only sets `CLIENT_TRANSACTIONS` in `mysql->server_capabilities` if the server supports a transaction-safe storage engine.
- Fixed `LOAD DATA INFILE` to allow numeric values to be read into `ENUM` and `SET` columns.
- Improved error diagnostic for slave thread exit.
- Fixed bug in `ALTER TABLE ... ORDER BY`.
- Added `max_user_connections` variable to **mysqld**.
- Limit query length for replication by `max_allowed_packet`, not the arbitrary limit of 4MB.
- Allow space around = in argument to `--set-variable`.

- Fixed problem in automatic repair that could leave some threads in state `Waiting for table`.
- `SHOW CREATE TABLE` now displays the `UNION=()` for `MERGE` tables.
- `ALTER TABLE` now remembers the old `UNION=()` definition.
- Fixed bug when replicating timestamps.
- Fixed bug in bidirectional replication.
- Fixed bug in the `BDB` storage engine that occurred when using an index on multiple-part key where a key part may be `NULL`.
- Fixed `MAX()` optimization on sub-key for `BDB` tables.
- Fixed problem where garbage results were returned when using `BDB` tables and `BLOB` or `TEXT` fields when joining many tables.
- Fixed a problem with `BDB` tables and `TEXT` columns.
- Fixed bug when using a `BLOB` key where a const row wasn't found.
- Fixed that **mysqlbinlog** writes the timestamp value for each query. This ensures that one gets same values for date functions like `NOW()` when using **mysqlbinlog** to pipe the queries to another server.
- Allow `--skip-gemini`, `--skip-bdb`, and `--skip-innodb` options to be specified when invoking **mysqld**, even if these storage engines are not compiled in to **mysqld**.
- You can now use `ASC` and `DESC` with `GROUP BY` columns to specify a sort order.
- Fixed a deadlock in the `SET` code, when one ran `SET @foo=bar`, where `bar` is a column reference, an error was not properly generated.

D.4.28. Changes in release 3.23.33 (09 Feb 2001)

- Fixed DNS lookups not to use the same mutex as the hostname cache. This enables known hosts to be quickly resolved even if a DNS lookup takes a long time.
- Added `--character-sets-dir` option to **myisampack**.
- Removed warnings when running `REPAIR TABLE ... EXTENDED`.
- Fixed a bug that caused a core dump when using `GROUP BY` on an alias, where the alias was the same as an existing column name.
- Added `SEQUENCE()` as an example user-defined function.
- Changed **mysql_install_db** to use `BINARY` for `CHAR` columns in the privilege tables.
- Changed `TRUNCATE` *`tbl_name`* to `TRUNCATE TABLE` *`tbl_name`* to use the same syntax as Oracle. Until 4.0 we also allow `TRUNCATE` *`tbl_name`* to not crash old code.
- Fixed ``no found rows'' bug in `MyISAM` tables when a `BLOB` was first part of a multiple-part key.
- Fixed bug where `CASE` didn't work with `GROUP BY`.
- Added `--sort-recover` option to **myisamchk**.

- **myisamchk -S** and `OPTIMIZE TABLE` now work on Windows.
- Fixed bug when using `DISTINCT` on results from functions that referred to a group function, like:

  ```
  SELECT a, DISTINCT SEC_TO_TIME(SUM(a))
  FROM tbl_name GROUP BY a, b;
  ```

- Fixed buffer overrun in `libmysqlclient` library. Fixed bug in handling `STOP` event after `ROTATE` event in replication.
- Fixed another buffer overrun in `DROP DATABASE`.
- Added `Table_locks_immediate` and `Table_locks_waited` status variables.
- Fixed bug in replication that broke slave server start with existing `master.info`. This fixes a bug introduced in 3.23.32.
- Added `SET SQL_SLAVE_SKIP_COUNTER=n` command to recover from replication glitches without a full database copy.
- Added `max_binlog_size` variable; the binary log is rotated automatically when the size crosses the limit.
- Added `Last_Error`, `Last_Errno`, and `Slave_skip_counter` variables to `SHOW SLAVE STATUS`.
- Fixed bug in `MASTER_POS_WAIT()` function.
- Execute core dump handler on `SIGILL`, and `SIGBUS` in addition to `SIGSEGV`.
- On x86 Linux, print the current query and thread (connection) id, if available, in the core dump handler.
- Fixed several timing bugs in the test suite.
- Extended **mysqltest** to take care of the timing issues in the test suite.
- `ALTER TABLE` can now be used to change the definition for a `MERGE` table.
- Fixed creation of `MERGE` tables on Windows.
- Portability fixes for OpenBSD and OS/2.
- Added `--temp-pool` option to **mysqld**. Using this option causes most temporary files created to use a small set of names, rather than a unique name for each new file. This is to work around a problem in the Linux kernel dealing with creating a bunch of new files with different names. With the old behavior, Linux seems to "leak" memory, as it's being allocated to the directory entry cache instead of the disk cache.

D.4.29. Changes in release 3.23.32 (22 Jan 2001)

- Changed code to get around compiler bug in Compaq C++ on OSF/1, that broke `BACKUP TABLE`, `RESTORE TABLE`, `CHECK TABLE`, `REPAIR TABLE`, and `ANALYZE TABLE`.
- Added option `FULL` to `SHOW COLUMNS`. Now we show the privilege list for the columns only if this option is given.

- Fixed bug in `SHOW LOGS` when there weren't any `BDB` logs.
- Fixed a timing problem in replication that could delay sending an update to the client until a new update was done.
- Don't convert field names when using `mysql_list_fields()`. This is to keep this code compatible with `SHOW FIELDS`.
- `MERGE` tables didn't work on Windows.
- Fixed problem with `SET PASSWORD=...` on Windows.
- Added missing `my_config.h` to RPM distribution.
- `TRIM("foo" from "foo")` didn't return an empty string.
- Added `--with-version-suffix` option to **configure**.
- Fixed core dump when client aborted connection without `mysql_close()`.
- Fixed a bug in `RESTORE TABLE` when trying to restore from a non-existent directory.
- Fixed a bug which caused a core dump on the slave when replicating `SET PASSWORD`.
- Added `MASTER_POS_WAIT()` function.

D.4.30. Changes in release 3.23.31 (17 Jan 2001: Production)

- The test suite now tests all reachable `BDB` interface code. During testing we found and fixed many errors in the interface code.
- Using `HAVING` on an empty table could produce one result row when it shouldn't.
- Fixed the MySQL RPM so it no longer depends on Perl5.
- Fixed some problems with `HEAP` tables on Windows.
- `SHOW TABLE STATUS` didn't show correct average row length for tables larger than 4GB.
- `CHECK TABLE ... EXTENDED` didn't check row links for fixed size tables.
- Added option `MEDIUM` to `CHECK TABLE`.
- Fixed problem when using `DECIMAL()` keys on negative numbers.
- `HOUR()` (and some other `TIME` functions) on a `CHAR` column always returned `NULL`.
- Fixed security bug in something (please upgrade if you are using an earlier MySQL 3.23 version).
- Fixed buffer overflow bug when writing a certain error message.
- Added usage of `setrlimit()` on Linux to get `-O --open_files_limit=#` to work on Linux.
- Added `bdb_version` variable to **mysqld**.

- Fixed bug when using expression of type:

  ```
  SELECT ... FROM t1 LEFT JOIN t2 ON (t1.a=t2.a) WHERE t1.a=t2.a
  ```

  In this case the test in the `WHERE` clause was wrongly optimized away.
- Fixed bug in `MyISAM` when deleting keys with possible `NULL` values, but the first key-column was not a prefix-compressed text column.
- Fixed **mysql.server** to read the `[mysql.server]` option file group rather than the `[mysql_server]` group.
- Fixed **safe_mysqld** and **mysql.server** to also read the `server` option section.
- Added `Threads_created` status variable to **mysqld**.

D.4.31. Changes in release 3.23.30 (04 Jan 2001)

- Added `SHOW OPEN TABLES` command.
- Fixed that `myisamdump` works against old **mysqld** servers.
- Fixed **myisamchk -k#** so that it works again.
- Fixed a problem with replication when the binary log file went over 2G on 32-bit systems.
- `LOCK TABLES` now automatically starts a new transaction.
- Changed `BDB` tables to not use internal subtransactions and reuse open files to get more speed.
- Added `--mysqld=#` option to **safe_mysqld**.
- Allow hex constants in the `--fields-*-by` and `--lines-terminated-by` options to **mysqldump** and **mysqlimport**. By Paul DuBois.
- Added `--safe-show-database` option to **mysqld**.
- Added `have_bdb`, `have_gemini`, `have_innobase`, `have_raid` and `have_openssl` to `SHOW VARIABLES` to make it easy to test for supported extensions.
- Added `--open-files-limit` option to **mysqld**.
- Changed `--open-files` option to `--open-files-limit` in **safe_mysqld**.
- Fixed a bug where some rows were not found with `HEAP` tables that had many keys.
- Fixed that `--bdb-no-sync` works.
- Changed `--bdb-recover` to `--bdb-no-recover` as recover should be on by default.
- Changed the default number of `BDB` locks to 10000.
- Fixed a bug from 3.23.29 when allocating the shared structure needed for `BDB` tables.
- Changed **mysqld_multi.sh** to use configure variables. Patch by Christopher McCrory.
- Added fixing of include files for Solaris 2.8.

- Fixed bug with `--skip-networking` on Debian Linux.
- Fixed problem that some temporary files where reported as having the name `UNOPENED` in error messages.
- Fixed bug when running two simultaneous `SHOW LOGS` queries.

D.4.32. Changes in release 3.23.29 (16 Dec 2000)

- Configure updates for Tru64, large file support, and better TCP wrapper support. By Albert Chin-A-Young.
- Fixed bug in `<=>` operator.
- Fixed bug in `REPLACE` with `BDB` tables.
- `LPAD()` and `RPAD()` shortens the result string if it's longer than the length argument.
- Added `SHOW LOGS` command.
- Remove unused `BDB` logs on shutdown.
- When creating a table, put `PRIMARY` keys first, followed by `UNIQUE` keys.
- Fixed a bug in `UPDATE` involving multiple-part keys where you specified all key parts both in the update and the `WHERE` part. In this case MySQL could try to update a record that didn't match the whole `WHERE` part.
- Changed drop table to first drop the tables and then the `.frm` file.
- Fixed a bug in the hostname cache which caused **mysqld** to report the hostname as `''` in some error messages.
- Fixed a bug with `HEAP` type tables; the variable `max_heap_table_size` wasn't used. Now either `MAX_ROWS` or `max_heap_table_size` can be used to limit the size of a `HEAP` type table.
- Changed the default `server-id` value to 1 for masters and 2 for slaves to make it easier to use the binary log.
- Renamed `bdb_lock_max` variable to `bdb_max_lock`.
- Added support for `AUTO_INCREMENT` on sub-fields for `BDB` tables.
- Added `ANALYZE TABLE` of `BDB` tables.
- In `BDB` tables, we now store the number of rows; this helps to optimize queries when we need an approximation of the number of rows.
- If we get an error in a multiple-row statement, we now only roll back the last statement, not the entire transaction.
- If you do a `ROLLBACK` when you have updated a non-transactional table you get an error as a warning.
- Added `--bdb-shared-data` option to **mysqld**.
- Added `Slave_open_temp_tables` status variable to **mysqld**

- Added `binlog_cache_size` and `max_binlog_cache_size` variables to **mysqld**.
- `DROP TABLE`, `RENAME TABLE`, `CREATE INDEX` and `DROP INDEX` are now transaction end-points.
- If you do a `DROP DATABASE` on a symbolically linked database, both the link and the original database are deleted.
- Fixed `DROP DATABASE` to work on OS/2.
- Fixed bug when doing a `SELECT DISTINCT ... table1 LEFT JOIN table2 ...` when `table2` was empty.
- Added `--abort-slave-event-count` and `--disconnect-slave-event-count` options to **mysqld** for debugging and testing of replication.
- Fixed replication of temporary tables. Handles everything except slave server restart.
- `SHOW KEYS` now shows whether key is `FULLTEXT`.
- New script **mysqld_multi**. See Section 5.1.5, “The **mysqld_multi** Program for Managing Multiple MySQL Servers”.
- Added new script, `mysql-multi.server.sh`. Thanks to Tim Bunce `<Tim.Bunce@ig.co.uk>` for modifying **mysql.server** to easily handle hosts running many **mysqld** processes.
- **safe_mysqld**, **mysql.server**, and **mysql_install_db** have been modified to use `mysql_print_defaults` instead of various hacks to read the `my.cnf` files. In addition, the handling of various paths has been made more consistent with how **mysqld** handles them by default.
- Automatically remove Berkeley DB transaction logs that no longer are in use.
- Fixed bug with several `FULLTEXT` indexes in one table.
- Added a warning if number of rows changes on `REPAIR TABLE`/`OPTIMIZE TABLE`.
- Applied patches for OS/2 by `Yuri Dario`.
- `FLUSH TABLES` *`tbl_name`* didn't always flush the index tree to disk properly.
- `--bootstrap` is now run in a separate thread. This fixes a problem that caused **mysql_install_db** to core dump on some Linux machines.
- Changed `mi_create()` to use less stack space.
- Fixed bug with optimizer trying to over-optimize `MATCH()` when used with `UNIQUE` key.
- Changed `crash-me` and the MySQL benchmarks to also work with FrontBase.
- Allow `RESTRICT` and `CASCADE` after `DROP TABLE` to make porting easier.
- Reset status variable which could cause problem if one used `--slow-log`.
- Added `connect_timeout` variable to **mysql** and **mysqladmin**.
- Added `connect-timeout` as an alias for `timeout` for option files read by `mysql_options()`.

D.4.33. Changes in release 3.23.28 (22 Nov 2000: Gamma)

- Added new options `--pager[=...]`, `--no-pager`, `--tee=...` and `--no-tee` to the **mysql** client. The new corresponding interactive commands are `pager`, `nopager`, `tee` and `notee`. See Section 8.3, "**mysql**, the Command-Line Tool", **mysql --help** and the interactive help for more information.
- Fixed crash when automatic repair of `MyISAM` table failed.
- Fixed a major performance bug in the table locking code when a lot of `SELECT`, `UPDATE` and `INSERT` statements constantly were running. The symptom was that the `UPDATE` and `INSERT` queries were locked for a long time while new `SELECT` statements were executed before the updates.
- When reading `options_files` with `mysql_options()` the `return-found-rows` option was ignored.
- You can now specify `interactive-timeout` in the option file that is read by `mysql_options()`. This makes it possible to force programs that run for a long time (like **mysqlhotcopy**) to use the `interactive_timeout` time instead of the `wait_timeout` time.
- Added to the slow query log the time and the username for each logged query. If you are using `--log-long-format` then also queries that do not use an index are logged, even if the query takes less than `long_query_time` seconds.
- Fixed a problem in `LEFT JOIN` which caused all columns in a reference table to be `NULL`.
- Fixed a problem when using `NATURAL JOIN` without keys.
- Fixed a bug when using a multiple-part keys where the first part was of type `TEXT` or `BLOB`.
- `DROP` of temporary tables wasn't stored in the update/binary log.
- Fixed a bug where `SELECT DISTINCT * ... LIMIT` *`row_count`* only returned one row.
- Fixed a bug in the assembler code in `strstr()` for SPARC and cleaned up the `global.h` header file to avoid a problem with bad aliasing with the compiler submitted with Red Hat 7.0. (Reported by Trond Eivind Glomsrød)
- The `--skip-networking` option now works properly on NT.
- Fixed a long outstanding bug in the `ISAM` tables when a row with a length of more than 65KB was shortened by a single byte.
- Fixed a bug in `MyISAM` when running multiple updating processes on the same table.
- Allow one to use `FLUSH TABLE` *`tbl_name`*.
- Added `--replicate-ignore-table`, `--replicate-do-table`, `--replicate-wild-ignore-table`, and `--replicate-wild-do-table` options to **mysqld**.
- Changed all log files to use our own `IO_CACHE` mechanism instead of `FILE` to avoid OS problems when there are many files open.
- Added `--open-files` and `--timezone` options to **safe_mysqld**.
- Fixed a fatal bug in `CREATE TEMPORARY TABLE ... SELECT ....`

- Fixed a problem with `CREATE TABLE ... SELECT NULL`.
- Added variables `large_file_support`,`net_read_timeout`, `net_write_timeout` and `query_buffer_size` to `SHOW VARIABLES`.
- Added status variables `Created_tmp_files` and `Sort_merge_passes` to `SHOW STATUS`.
- Fixed a bug where we didn't allow an index name after the `FOREIGN KEY` definition.
- Added `TRUNCATE` *`tbl_name`* as a synonym for `DELETE FROM` *`tbl_name`*.
- Fixed a bug in a `BDB` key compare function when comparing part keys.
- Added `bdb_lock_max` variable to **mysqld**.
- Added more tests to the benchmark suite.
- Fixed an overflow bug in the client code when using overly long database names.
- `mysql_connect()` now aborts on Linux if the server doesn't answer in `timeout` seconds.
- `SLAVE START` did not work if you started with `--skip-slave-start` and had not explicitly run `CHANGE MASTER TO`.
- Fixed the output of `SHOW MASTER STATUS` to be consistent with `SHOW SLAVE STATUS`. (It now has no directory in the log name.)
- Added `PURGE MASTER LOGS TO`.
- Added `SHOW MASTER LOGS` statement to display a list of binary log files.
- Added `--safemalloc-mem-limit` option to **mysqld** to simulate memory shortage when compiled with the `--with-debug=full` option.
- Fixed several core dumps in out-of-memory conditions.
- `SHOW SLAVE STATUS` was using an uninitialized mutex if the slave had not been started yet.
- Fixed bug in `ELT()` and `MAKE_SET()` when the query used a temporary table.
- `CHANGE MASTER TO` without specifying `MASTER_LOG_POS` would set it to 0 instead of 4 and hit the magic number in the master binary log.
- `ALTER TABLE ... ORDER BY ...` syntax added. This creates the new table with the rows in a specific order.

D.4.34. Changes in release 3.23.27 (24 Oct 2000)

- Fixed a bug where the automatic repair of `MyISAM` tables sometimes failed when the data file was corrupt.
- Fixed a bug in `SHOW CREATE` when using `AUTO_INCREMENT` columns.
- Changed `BDB` tables to use new compare function in Berkeley DB 3.2.3.
- You can now use Unix socket files with MIT-pthreads.

- Added the `latin5` (turkish) character set.
- Small portability fixes.

D.4.35. Changes in release 3.23.26 (18 Oct 2000)

- Renamed `FLUSH MASTER` and `FLUSH SLAVE` to `RESET MASTER` and `RESET SLAVE`.
- Fixed `<>` to work properly with `NULL`.
- Fixed a problem with `SUBSTRING_INDEX()` and `REPLACE()`. (Patch by Alexander Igonitchev)
- Fix `CREATE TEMPORARY TABLE IF NOT EXISTS` not to produce an error if the table exists.
- If you don't create a `PRIMARY KEY` in a `BDB` table, a hidden `PRIMARY KEY` is created.
- Added read-only-key optimization to `BDB` tables.
- `LEFT JOIN` in some cases preferred a full table scan when there was no `WHERE` clause.
- When using `--log-slow-queries`, don't count the time waiting for a lock.
- Fixed bug in lock code on Windows which could cause the key cache to report that the key file was crashed even if it was okay.
- Automatic repair of `MyISAM` tables if you start **mysqld** with `--myisam-recover`.
- Removed the `TYPE=` keyword from `CHECK TABLE` and `REPAIR TABLE`. Allow `CHECK TABLE` options to be combined. (You can still use `TYPE=`, but this usage is deprecated.)
- Fixed mutex bug in the binary replication log --- long update queries could be read only in part by the slave if it did it at the wrong time, which was not fatal, but resulted in a performance-degrading reconnect and a scary message in the error log.
- Changed the format of the binary log --- added magic number, server version, binary log version. Added the server ID and query error code for each query event.
- Replication thread from the slave now kills all the stale threads from the same server.
- Long replication usernames were not being handled properly.
- Added `--replicate-rewrite-db` option to **mysqld**.
- Added `--skip-slave-start` option to **mysqld**.
- Updates that generated an error code (such as `INSERT INTO foo(some_key) values (1),(1)`) erroneously terminated the slave thread.
- Added optimization of queries where `DISTINCT` is used only on columns from some of the tables.
- Allow floating-point numbers where there is no sign after the exponent (like `1e1`).
- `SHOW GRANTS` didn't always show all column grants.
- Added `--default-extra-file=#` option to all MySQL clients.
- Columns referenced in `INSERT` statements now are initialized properly.

- UPDATE didn't always work when used with a range on a timestamp that was part of the key that was used to find rows.
- Fixed a bug in FULLTEXT index when inserting a NULL column.
- Changed to use mkstemp() instead of tempnam(). Based on a patch from John Jones.

D.4.36. Changes in release 3.23.25 (29 Sep 2000)

- Fixed that databasename works as second argument to **mysqlhotcopy**.
- The values for the UMASK and UMASK_DIR environment variables now can be specified in octal by beginning the value with a zero.
- Added RIGHT JOIN. This makes RIGHT a reserved word.
- Added @@IDENTITY as a synonym for LAST_INSERT_ID(). (This is for MSSQL compatibility.)
- Fixed a bug in **myisamchk** and REPAIR TABLE when using FULLTEXT index.
- LOAD DATA INFILE now works with FIFOs. (Patch by Toni L. Harbaugh-Blackford.)
- FLUSH LOGS broke replication if you specified a log name with an explicit extension as the value of the log-bin option.
- Fixed a bug in MyISAM with packed multiple-part keys.
- Fixed crash when using CHECK TABLE on Windows.
- Fixed a bug where FULLTEXT index always used the koi8_ukr character set.
- Fixed privilege checking for CHECK TABLE.
- The MyISAM repair/reindex code didn't use the --tmpdir option for its temporary files.
- Added BACKUP TABLE and RESTORE TABLE.
- Fixed core dump on CHANGE MASTER TO when the slave did not have the master to start with.
- Fixed incorrect Time in the processlist for Connect of the slave thread.
- The slave now logs when it connects to the master.
- Fixed a core dump bug when doing FLUSH MASTER if you didn't specify a filename argument to --log-bin.
- Added missing ha_berkeley.x files to the MySQL Windows distribution.
- Fixed some mutex bugs in the log code that could cause thread blocks if new log files couldn't be created.
- Added lock time and number of selected processed rows to slow query log.
- Added --memlock option to **mysqld** to lock **mysqld** in memory on systems with the mlockall() call (as in Solaris).

- HEAP tables didn't use keys properly. (Bug from 3.23.23.)
- Added better support for MERGE tables (keys, mapping, creation, documentation...). See Section 14.2, “The MERGE Storage Engine”.
- Fixed bug in **mysqldump** from 3.23 which caused some CHAR columns not to be quoted.
- Merged analyze, check, optimize and repair code.
- OPTIMIZE TABLE is now mapped to REPAIR TABLE with statistics and sorting of the index tree. This means that for the moment it only works on MyISAM tables.
- Added a pre-alloced block to root_malloc to get fewer mallocs.
- Added a lot of new statistics variables.
- Fixed ORDER BY bug with BDB tables.
- Removed warning that **mysqld** couldn't remove the .pid file under Windows.
- Changed --log-isam to log MyISAM tables instead of isam tables.
- Fixed CHECK TABLE to work on Windows.
- Added file mutexes to make pwrite() safe on Windows.

D.4.37. Changes in release 3.23.24 (08 Sep 2000)

- Added Created_tmp_disk_tables variable to **mysqld**.
- To make it possible to reliably dump and restore tables with TIMESTAMP(*X*) columns, MySQL now reports columns with *X* other than 14 or 8 to be strings.
- Changed sort order for latin1 as it was before MySQL 3.23.23. Any table that was created or modified with 3.23.22 must be repaired if it has CHAR columns that may contain characters with ASCII values greater than 128!
- Fixed small memory leak introduced from 3.23.22 when creating a temporary table.
- Fixed problem with BDB tables and reading on a unique (not primary) key.
- Restored the win1251 character set (it's now only marked deprecated).

D.4.38. Changes in release 3.23.23 (01 Sep 2000)

- Changed sort order for 'German'; all tables created with 'German' sortorder must be repaired with REPAIR TABLE or **myisamchk** before use!
- Added --core-file option to **mysqld** to get a core file on Linux if **mysqld** dies on the SIGSEGV signal.
- MySQL client **mysql** now starts with option --no-named-commands (-g) by default. This option can be disabled with --enable-named-commands (-G). This may cause incompatibility problems in some cases, for example, in SQL scripts that use named commands without a semicolon,

etc.! Long format commands still work from the first line.

- Fixed a problem when using many pending `DROP TABLE` statements at the same time.
- Optimizer didn't use keys properly when using `LEFT JOIN` on an empty table.
- Added shorter help text when invoking **mysqld** with incorrect options.
- Fixed non-fatal `free()` bug in **mysqlimport**.
- Fixed bug in `MyISAM` index handling of `DECIMAL`/`NUMERIC` keys.
- Fixed a bug in concurrent insert in `MyISAM` tables. In some contexts, usage of `MIN(key_part)` or `MAX(key_part)` returned an empty set.
- Updated **mysqlhotcopy** to use the new `FLUSH TABLES table_list` syntax. Only tables which are being backed up are flushed now.
- Changed behavior of `--enable-thread-safe-client` so that both non-threaded (`-lmysqlclient`) and threaded (`-lmysqlclient_r`) libraries are built. Users who linked against a threaded `-lmysqlclient` need to link against `-lmysqlclient_r` now.
- Added atomic `RENAME TABLE` command.
- Don't count `NULL` values in `COUNT(DISTINCT ...)`.
- Changed `ALTER TABLE`, `LOAD DATA INFILE` on empty tables and `INSERT ... SELECT ...` on empty tables to create non-unique indexes in a separate batch with sorting. This makes these statements much faster when you have many indexes.
- `ALTER TABLE` now logs the first used insert_id correctly.
- Fixed crash when adding a default value to a `BLOB` column.
- Fixed a bug with `DATE_ADD/DATE_SUB` where it returned a datetime instead of a date.
- Fixed a problem with the thread cache which made some threads show up as `***DEAD***` in `SHOW PROCESSLIST`.
- Fixed a lock in our thr_rwlock code, which could make selects that run at the same time as concurrent inserts crash. This affects only systems that don't have the `pthread_rwlock_rdlock` code.
- When deleting rows with a non-unique key in a `HEAP` table, all rows weren't always deleted.
- Fixed bug in range optimizer for `HEAP` tables for searches on a part index.
- Fixed `SELECT` on part keys to work with `BDB` tables.
- Fixed `INSERT INTO bdb_table ... SELECT` to work with `BDB` tables.
- `CHECK TABLE` now updates key statistics for the table.
- `ANALYZE TABLE` now only updates tables that have been changed since the last `ANALYZE TABLE`. Note that this is a new feature and tables are not marked to be analyzed until they are updated in any way with 3.23.23 or newer. For older tables, you have to do `CHECK TABLE` to update the key distribution.
- Fixed some minor privilege problems with `CHECK TABLE`, `ANALYZE TABLE`, `REPAIR TABLE` and `SHOW CREATE` commands.

- Added `CHANGE MASTER TO` statement.
- Added `FAST`, `QUICK` `EXTENDED` check types to `CHECK TABLES`.
- Changed **myisamchk** so that `--fast` and `--check-only-changed` are also honored with `--sort-index` and `--analyze`.
- Fixed fatal bug in `LOAD TABLE FROM MASTER` that did not lock the table during index re-build.
- `LOAD DATA INFILE` broke replication if the database was excluded from replication.
- More variables in `SHOW SLAVE STATUS` and `SHOW MASTER STATUS`.
- `SLAVE STOP` now does not return until the slave thread actually exits.
- Full-text search via the `MATCH()` function and `FULLTEXT` index type (for `MyISAM` files). This makes `FULLTEXT` a reserved word.

D.4.39. Changes in release 3.23.22 (31 Jul 2000)

- Fixed that `lex_hash.h` is created properly for each MySQL distribution.
- Fixed that `MASTER` and `COLLECTION` are not reserved words.
- The log generated by `--slow-query-log` didn't contain the whole queries.
- Fixed that open transactions in `BDB` tables are rolled back if the connection is closed unexpectedly.
- Added workaround for a bug in **gcc** 2.96 (intel) and **gcc** 2.9 (IA-64) in `gen_lex_hash.c`.
- Fixed memory leak in the client library when using `host=` in the `my.cnf` file.
- Optimized functions that manipulate the hours/minutes/seconds.
- Fixed bug when comparing the result of `DATE_ADD()`/`DATE_SUB()` against a number.
- Changed the meaning of `-F, --fast` for **myisamchk**. Added `-C, --check-only-changed` option to **myisamchk**.
- Added `ANALYZE` *`tbl_name`* to update key statistics for tables.
- Changed binary items `0x...` to be regarded as integers by default.
- Fix for SCO and `SHOW PROCESSLIST`.
- Added `auto-rehash` on reconnect for the **mysql** client.
- Fixed a newly introduced bug in `MyISAM`, where the index file couldn't get bigger than 64MB.
- Added `SHOW MASTER STATUS` and `SHOW SLAVE STATUS`.

D.4.40. Changes in release 3.23.21 (04 Jul 2000)

- Added `mysql_character_set_name()` function to the MySQL C API.

- Made the update log ASCII 0 safe.
- Added the **mysql_config** script.
- Fixed problem when using < or > with a char column that was only partly indexed.
- One would get a core dump if the log file was not readable by the MySQL user.
- Changed **mysqladmin** to use `CREATE DATABASE` and `DROP DATABASE` statements instead of the old deprecated API calls.
- Fixed `chown` warning in **safe_mysqld**.
- Fixed a bug in `ORDER BY` that was introduced in 3.23.19.
- Only optimize the `DELETE FROM tbl_name` to do a drop+create of the table if we are in `AUTO-COMMIT` mode (needed for `BDB` tables).
- Added extra checks to avoid index corruption when the `ISAM`/`MyISAM` index files get full during an `INSERT`/`UPDATE`.
- **myisamchk** didn't correctly update row checksum when used with `-ro` (this only gave a warning in subsequent runs).
- Fixed bug in `REPAIR TABLE` so that it works with tables without indexes.
- Fixed buffer overrun in `DROP DATABASE`.
- `LOAD TABLE FROM MASTER` is sufficiently bug-free to announce it as a feature.
- `MATCH` and `AGAINST` are now reserved words.

D.4.41. Changes in release 3.23.20 (28 Jun 2000: Beta)

- Fixed bug in 3.23.19; `DELETE FROM tbl_name` removed the `.frm` file.
- Added `SHOW CREATE TABLE`.

D.4.42. Changes in release 3.23.19

- Changed copyright for all files to GPL for the server code and utilities and to LGPL for the client libraries. See http://www.fsf.org/licenses/.
- Fixed bug where all rows matching weren't updated on a `MyISAM` table when doing update based on key on a table with many keys and some key changed values.
- The Linux MySQL RPMs and binaries are now statically linked with a linuxthread version that has faster mutex handling when used with MySQL.
- `ORDER BY` can now use `REF` keys to find subsets of the rows that need to be sorted.
- Changed name of **print_defaults** program to **my_print_defaults** to avoid name confusion.
- Fixed `NULLIF()` to work as required by standard SQL.

- Added `net_read_timeout` and `net_write_timeout` as startup parameters to **mysqld**.
- Fixed bug that destroyed index when doing **myisamchk --sort-records** on a table with prefix compressed index.
- Added **pack_isam** and **myisampack** to the standard MySQL distribution.
- Added the syntax `BEGIN WORK` (the same as `BEGIN`).
- Fixed core dump bug when using `ORDER BY` on a `CONV()` expression.
- Added `LOAD TABLE FROM MASTER`.
- Added `FLUSH MASTER` and `FLUSH SLAVE`.
- Fixed big/little endian problem in the replication.

D.4.43. Changes in release 3.23.18 (11 Jun 2000)

- Fixed a problem from 3.23.17 when choosing character set on the client side.
- Added `FLUSH TABLES WITH READ LOCK` to make a global lock suitable for making a copy of MySQL data files.
- `CREATE TABLE ... SELECT ... PROCEDURE` now works.
- Internal temporary tables now use compressed index when using `GROUP BY` on `VARCHAR/CHAR` columns.
- Fixed a problem when locking the same table with both a `READ` and a `WRITE` lock.
- Fixed problem with **myisamchk** and `RAID` tables.

D.4.44. Changes in release 3.23.17 (07 Jun 2000)

- Fixed a bug in `FIND_IN_SET()` when the first argument was `NULL`.
- Added table locks to Berkeley DB.
- Fixed a bug with `LEFT JOIN` and `ORDER BY` where the first table had only one matching row.
- Added 4 sample `my.cnf` example files in the `support-files` directory.
- Fixed `duplicated key` problem when doing big `GROUP BY` operations. (This bug was probably introduced in 3.23.15.)
- Changed syntax for `INNER JOIN` to match standard SQL.
- Added `NATURAL JOIN` syntax.
- A lot of fixes in the `BDB` interface.
- Added handling of `--no-defaults` and `--defaults-file` to **safe_mysqld.sh** and **mysql_install_db.sh**.

- Fixed bug in reading compressed tables with many threads.
- Fixed that `USE INDEX` works with `PRIMARY` keys.
- Added `BEGIN` statement to start a transaction in `AUTOCOMMIT` mode.
- Added support for symbolic links for Windows.
- Changed protocol to let client know if the server is in `AUTOCOMMIT` mode and if there is a pending transaction. If there is a pending transaction, the client library gives an error before reconnecting to the server to let the client know that the server did a rollback. The protocol is still backward-compatible with old clients.
- `KILL` now works on a thread that is locked on a 'write' to a dead client.
- Fixed memory leak in the replication slave thread.
- Added new `log-slave-updates` option to **mysqld**, to allow daisy-chaining the slaves.
- Fixed compile error on FreeBSD and other systems where `pthread_t` is not the same as `int`.
- Fixed master shutdown aborting the slave thread.
- Fixed a race condition in `INSERT DELAYED` code when doing `ALTER TABLE`.
- Added deadlock detection sanity checks to `INSERT DELAYED`.

D.4.45. Changes in release 3.23.16 (16 May 2000)

- Added `SLAVE START` and `SLAVE STOP` statements.
- Added `TYPE=QUICK` option to `CHECK TABLE` and to `REPAIR TABLE`.
- Fixed bug in `REPAIR TABLE` when the table was in use by other threads.
- Added a thread cache to make it possible to debug MySQL with **gdb** when one does a lot of reconnects. This also improves systems where you can't use persistent connections.
- Lots of fixes in the Berkeley DB interface.
- `UPDATE IGNORE` does not abort if an update results in a `DUPLICATE_KEY` error.
- Put `CREATE TEMPORARY TABLE` commands in the update log.
- Fixed bug in handling of masked IP numbers in the privilege tables.
- Fixed bug with `delay_key_write` tables and `CHECK TABLE`.
- Added `--replicate-do-db` and `--replicate-ignore-db` options to **mysqld**, to restrict which databases get replicated.
- Added `SQL_LOG_BIN` option.

D.4.46. Changes in release 3.23.15 (08 May 2000)

- To start **mysqld** as `root`, you must now use the `--user=root` option.
- Added interface to Berkeley DB. (This is not yet functional; play with it at your own risk!)
- Replication between master and slaves.
- Fixed bug that other threads could steal a lock when a thread had a lock on a table and did a `FLUSH TABLES` command.
- Added the `slow_launch_time` variable and the `Slow_launch_threads` status variable to **mysqld**. These can be examined with **mysqladmin variables** and **mysqladmin extended-status**.
- Added functions `INET_NTOA()` and `INET_ATON()`.
- The default type of `IF()` now depends on the second and third arguments and not only on the second argument.
- Fixed case when **myisamchk** could go into a loop when trying to repair a crashed table.
- Don't write `INSERT DELAYED` to update log if `SQL_LOG_UPDATE=0`.
- Fixed problem with `REPLACE` on `HEAP` tables.
- Added possible character sets and time zone to `SHOW VARIABLES` output.
- Fixed bug in locking code that could result in locking problems with concurrent inserts under high load.
- Fixed a problem with `DELETE` of many rows on a table with compressed keys where MySQL scanned the index to find the rows.
- Fixed problem with `CHECK TABLE` on table with deleted keyblocks.
- Fixed a bug in reconnect (at the client side) where it didn't free memory properly in some contexts.
- Fixed problems in update log when using `LAST_INSERT_ID()` to update a table with an `AUTO_INCREMENT` key.
- Added `NULLIF()` function.
- Fixed bug when using `LOAD DATA INFILE` on a table with `BLOB/TEXT` columns.
- Optimized `MyISAM` to be faster when inserting keys in sorted order.
- `EXPLAIN SELECT ...` now also prints out whether MySQL needs to create a temporary table or use file sorting when resolving the `SELECT`.
- Added optimization to skip `ORDER BY` parts where the part is a constant expression in the `WHERE` part. Indexes can now be used even if the `ORDER BY` doesn't match the index exactly, as long as all the unused index parts and all the extra `ORDER BY` columns are constants in the `WHERE` clause. See Section 7.4.5, “How MySQL Uses Indexes”.
- `UPDATE` and `DELETE` on a whole unique key in the `WHERE` part are now faster than before.
- Changed `RAID_CHUNKSIZE` to be in 1024-byte increments.
- Fixed core dump in `LOAD_FILE(NULL)`.

D.4.47. Changes in release 3.23.14 (09 Apr 2000)

- Added **mysqlbinlog** program for displaying binary log files in text format.
- Added `mysql_real_escape_string()` function to the MySQL C API.
- Fixed a bug in `CONCAT()` where one of the arguments was a function that returned a modified argument.
- Fixed a critical bug in **myisamchk**, where it updated the header in the index file when one only checked the table. This confused the **mysqld** daemon if it updated the same table at the same time. Now the status in the index file is only updated if one uses `--update-state`. With older **myisamchk** versions you should use `--read-only` when only checking tables, if there is the slightest chance that the **mysqld** server is working on the table at the same time!
- Fixed that `DROP TABLE` is logged in the update log.
- Fixed problem when searching on `DECIMAL()` key field where the column data contained leading zeros.
- Fix bug in **myisamchk** when the `AUTO_INCREMENT` column isn't the first key.
- Allow `DATETIME` in ISO8601 format: 2000-03-12T12:00:00
- Dynamic character sets. A **mysqld** binary can now handle many different character sets (you can choose which when starting **mysqld**).
- Added `REPAIR TABLE` statement.
- Added `mysql_thread_safe()` function to the MySQL C API.
- Added the `UMASK_DIR` environment variable.
- Added `CONNECTION_ID()` function to return the client connection thread ID.
- When using `=` on `BLOB` or `VARCHAR BINARY` keys, where only a part of the column was indexed, the whole column of the result row wasn't compared.
- Fix for `sjis` character set and `ORDER BY`.
- When running in ANSI mode, don't allow columns to be used that aren't in the `GROUP BY` part.

D.4.48. Changes in release 3.23.13 (14 Mar 2000)

- Fixed problem when doing locks on the same table more than 2 times in the same `LOCK TABLE` command; this fixed the problem one got when running the test-ATIS test with `--fast` or `--check-only-changed`.
- Added `SQL_BUFFER_RESULT` option to `SELECT`.
- Removed endspace from double/float numbers in results from temporary tables.
- Added `CHECK TABLE` command.
- Added changes for `MyISAM` in 3.23.12 that didn't get into the source distribution because of CVS problems.

- Fixed bug so that **mysqladmin shutdown** waits for the local server to close down.
- Fixed a possible endless loop when calculating timestamp.
- Added **print_defaults** program to the `.rpm` files. Removed **mysqlbug** from the client `.rpm` file.

D.4.49. Changes in release 3.23.12 (07 Mar 2000)

- Fixed bug in `MyISAM` involving `REPLACE ... SELECT ...` which could give a corrupted table.
- Fixed bug in **myisamchk** where it incorrectly reset the `AUTO_INCREMENT` value.
- LOTS of patches for Linux Alpha. MySQL now appears to be relatively stable on Alpha.
- Changed `DISTINCT` on `HEAP` temporary tables to use hashed keys to quickly find duplicated rows. This mostly concerns queries of type `SELECT DISTINCT ... GROUP BY ...`. This fixes a problem where not all duplicates were removed in queries of the above type. In addition, the new code is MUCH faster.
- Added patches to make MySQL compile on Mac OS X.
- Added `IF NOT EXISTS` clause to `CREATE DATABASE`.
- Added `--all-databases` and `--databases` options to **mysqldump** to allow dumping of many databases at the same time.
- Fixed bug in compressed `DECIMAL()` index in `MyISAM` tables.
- Fixed bug when storing 0 into a timestamp.
- When doing **mysqladmin shutdown** on a local connection, **mysqladmin** now waits until the PID file is gone before terminating.
- Fixed core dump with some `COUNT(DISTINCT ...)` queries.
- Fixed that **myisamchk** works properly with RAID tables.
- Fixed problem with `LEFT JOIN` and *`key_col`* `IS NULL`.
- Fixed bug in `net_clear()` which could give the error `Aborted connection` in the MySQL clients.
- Added options `USE INDEX (`*`key_list`*`)` and `IGNORE INDEX (`*`key_list`*`)` as parameters in `SELECT`.
- `DELETE` and `RENAME` should now work on `RAID` tables.

D.4.50. Changes in release 3.23.11 (16 Feb 2000)

- Added `HIGH_PRIORITY` option to `INSERT`. This overrides the effect of the `--low-priority-updates` server option and does not perform concurrent inserts.
- Allow the `ALTER TABLE` *`tbl_name`* `ADD (field_list)` syntax.

- Fixed problem with optimizer that could sometimes use incorrect keys.
- Fixed that `GRANT/REVOKE ALL PRIVILEGES` doesn't affect `GRANT OPTION`.
- Removed extra `')'` from the output of `SHOW GRANTS`.
- Fixed problem when storing numbers in timestamps.
- Fix problem with time zones that have half hour offsets.
- Allow the syntax `UNIQUE INDEX` in `CREATE` statements.
- **mysqlhotcopy** - fast online hot-backup utility for local MySQL databases. By Tim Bunce.
- New more secure **mysqlaccess**. Thanks to Steve Harvey for this.
- Added `--i-am-a-dummy` and `--safe-updates` options to **mysql**.
- Added `select_limit` and `max_join_size` variables to **mysql**.
- Added `SQL_MAX_JOIN_SIZE` and `SQL_SAFE_UPDATES` options.
- Added `READ LOCAL` lock that doesn't lock the table for concurrent inserts. (This is used by **mysqldump**.)
- Changed that `LOCK TABLES ... READ` no longer allows concurrent inserts.
- Added `--skip-delay-key-write` option to **mysqld**.
- Fixed security problem in the protocol regarding password checking.
- `_rowid` can now be used as an alias for an integer type unique indexed column.
- Added back blocking of `SIGPIPE` when compiling with `--thread-safe-clients` to make things safe for old clients.

D.4.51. Changes in release 3.23.10 (30 Jan 2000)

- Fixed bug in 3.23.9 where memory wasn't properly freed when using `LOCK TABLES`.

D.4.52. Changes in release 3.23.9 (29 Jan 2000)

- Fixed problem that affected queries that did arithmetic on group functions.
- Fixed problem with timestamps and `INSERT DELAYED`.
- Fixed that `date_col BETWEEN const_date AND const_date` works.
- Fixed problem when only changing a 0 to `NULL` in a table with `BLOB/TEXT` columns.
- Fixed bug in range optimizer when using many key parts and or on the middle key parts: `WHERE K1=1 and K3=2 and (K2=2 and K4=4 or K2=3 and K4=5)`
- Added `source` command to **mysql** to allow reading of batch files inside the **mysql** client. Original patch by Matthew Vanecek.

- Fixed critical problem with the `WITH GRANT OPTION` option.
- Don't give an unnecessary `GRANT` error when using tables from many databases in the same query.
- Added VIO wrapper (needed for SSL support; by Andrei Errapart and Tõnu Samuel).
- Fixed optimizer problem on `SELECT` when using many overlapping indexes. MySQL should now be able to choose keys even better when there are many keys to choose from.
- Changed optimizer to prefer a range key instead of a ref key when the range key can uses more columns than the ref key (which only can use columns with `=`). For example, the following type of queries should now be faster: `SELECT * from key_part_1=const and key_part_2 > const2`
- Fixed bug that a change of all `VARCHAR` columns to `CHAR` columns didn't change row type from dynamic to fixed.
- Disabled floating-point exceptions for FreeBSD to fix core dump when doing `SELECT FLOOR(POW(2,63))`.
- Renamed **mysqld** startup option from `--delay-key-write` to `--delay-key-write-for-all-tables`.
- Added `read-next-on-key` to `HEAP` tables. This should fix all problems with `HEAP` tables when using non-`UNIQUE` keys.
- Added option to print default arguments to all clients.
- Added `--log-slow-queries` option to **mysqld** to log all queries that take a long time to a separate log file with a time indicating how long the query took.
- Fixed core dump when doing `WHERE key_col=RAND(...)`.
- Fixed optimization bug in `SELECT ... LEFT JOIN ... key_col IS NULL`, when `key_col` could contain `NULL` values.
- Fixed problem with 8-bit characters as separators in `LOAD DATA INFILE`.

D.4.53. Changes in release 3.23.8 (02 Jan 2000)

- Fixed problem when handling indexfiles larger than 8GB.
- Added latest patches to MIT-pthreads for NetBSD.
- Fixed problem with time zones that are < GMT - 11.
- Fixed a bug when deleting packed keys in `NISAM`.
- Fixed problem with `ISAM` when doing some `ORDER BY ... DESC` queries.
- Fixed bug when doing a join on a text key which didn't cover the whole key.
- Option `--delay-key-write` didn't enable delayed key writing.
- Fixed update of `TEXT` column which involved only case changes.
- Fixed that `INSERT DELAYED` doesn't update timestamps that are given.

- Added function YEARWEEK() and options x, X, v and V to DATE_FORMAT().
- Fixed problem with MAX(indexed_column) and HEAP tables.
- Fixed problem with BLOB NULL keys and LIKE "prefix%".
- Fixed problem with MyISAM and fixed-length rows < 5 bytes.
- Fixed problem that could cause MySQL to touch freed memory when doing very complicated GROUP BY queries.
- Fixed core dump if you got a crashed table where an ENUM field value was too big.

D.4.54. Changes in release 3.23.7 (10 Dec 1999)

- Fixed workaround under Linux to avoid problems with pthread_mutex_timedwait(), which is used with INSERT DELAYED. See Section 2.12.1, “Linux Notes”.
- Fixed that one get a 'disk full' error message if one gets disk full when doing sorting (instead of waiting until we got more disk space).
- Fixed a bug in MyISAM with keys > 250 characters.
- In MyISAM one can now do an INSERT at the same time as other threads are reading from the table.
- Added max_write_lock_count variable to **mysqld** to force a READ lock after a certain number of WRITE locks.
- Inverted flag delay_key_write on show variables.
- Renamed concurrency variable to thread_concurrency.
- The following functions are now multi-byte-safe: LOCATE(*substr*,*str*), POSITION(*substr* IN *str*), LOCATE(*substr*,*str*,*pos*), INSTR(*str*,*substr*), LEFT(*str*,*len*), RIGHT(*str*,*len*), SUBSTRING(*str*,*pos*,*len*), SUBSTRING(*str* FROM *pos* FOR *len*), MID(*str*,*pos*,*len*), SUBSTRING(*str*,*pos*), SUBSTRING(*str* FROM *pos*), SUBSTRING_INDEX(*str*,*delim*,*count*), RTRIM(*str*), TRIM([[BOTH | TRAILING] [*remstr*] FROM] *str*), REPLACE(*str*,*from_str*,*to_str*), REVERSE(*str*), INSERT(*str*,*pos*,*len*,*newstr*), LCASE(*str*), LOWER(*str*), UCASE(*str*) and UPPER(*str*); patch by Wei He.
- Fix core dump when releasing a lock from a non-existent table.
- Remove locks on tables before starting to remove duplicates.
- Added option FULL to SHOW PROCESSLIST.
- Added option --verbose to **mysqladmin**.
- Fixed problem when automatically converting HEAP to MyISAM.
- Fixed bug in HEAP tables when doing insert + delete + insert + scan the table.
- Fixed bugs on Alpha with REPLACE() and LOAD DATA INFILE.
- Added interactive_timeout variable to **mysqld**.

- Changed the argument to `mysql_data_seek()` from `ulong` to `ulonglong`.

D.4.55. Changes in release 3.23.6 (15 Nov 1999)

- Added `-O lower_case_table_names={0|1}` option to **mysqld** to allow users to force table names to lowercase.
- Added `SELECT ... INTO DUMPFILE`.
- Added `--ansi` option to **mysqld** to make some functions standard SQL compatible.
- Temporary table names now start with `#sql`.
- Added quoting of identifiers with `` ` `` (`"` in `--ansi` mode).
- Changed to use `snprintf()` when printing floats to avoid some buffer overflows on FreeBSD.
- Made `FLOOR()` overflow safe on FreeBSD.
- Added `--quote-names` option to **mysqldump**.
- Fixed bug that one could make a part of a `PRIMARY KEY NOT NULL`.
- Fixed `encrypt()` to be thread-safe and not reuse buffer.
- Added `mysql_odbc_escape_string()` function to support big5 characters in MyODBC.
- Rewrote the storage engine to use classes. This introduces a lot of new code, but make table handling faster and better.
- Added patch by Sasha for user-defined variables.
- Changed that `FLOAT` and `DOUBLE` (without any length modifiers) no longer are fixed decimal point numbers.
- Changed the meaning of `FLOAT(X)`: Now this is the same as `FLOAT` if X <= 24 and a `DOUBLE` if 24 < X <= 53.
- `DECIMAL(X)` is now an alias for `DECIMAL(X,0)` and `DECIMAL` is now an alias for `DECIMAL(10,0)`. The same goes for `NUMERIC`.
- Added option `ROW_FORMAT={DEFAULT | DYNAMIC | FIXED | COMPRESSED}` to `CREATE_TABLE`.
- `DELETE FROM tbl_name` didn't work on temporary tables.
- Changed function `CHAR_LENGTH()` to be multi-byte character safe.
- Added function `ORD(string)`.

D.4.56. Changes in release 3.23.5 (20 Oct 1999)

- Fixed some Y2K problems in the new date handling in 3.23.

- Fixed problem with `SELECT DISTINCT ... ORDER BY RAND()`.
- Added patches by Sergei A. Golubchik for text searching on the `MyISAM` level.
- Fixed cache overflow problem when using full joins without keys.
- Fixed some configure issues.
- Some small changes to make parsing faster.
- Adding a column after the last field with `ALTER TABLE` didn't work.
- Fixed problem when using an `AUTO_INCREMENT` column in two keys
- With `MyISAM`, you now can have an `AUTO_INCREMENT` column as a key sub part: `CREATE TABLE foo (a INT NOT NULL AUTO_INCREMENT, b CHAR(5), PRIMARY KEY (b,a))`
- Fixed bug in `MyISAM` with packed char keys that could be `NULL`.
- `AS` on field name with `CREATE TABLE` *`tbl_name`* `SELECT ...` didn't work.
- Allow use of `NATIONAL` and `NCHAR` when defining character columns. This is the same as not using `BINARY`.
- Don't allow `NULL` columns in a `PRIMARY KEY` (only in `UNIQUE` keys).
- Clear `LAST_INSERT_ID()` if one uses this in ODBC: `WHERE auto_increment_column IS NULL`. This seems to fix some problems with Access.
- `SET SQL_AUTO_IS_NULL=0|1` now turns on/off the handling of searching for the last inserted row with `WHERE auto_increment_column IS NULL`.
- Added new variable `concurrency` to **mysqld** for Solaris.
- Added `--relative` option to **mysqladmin** to make `extended-status` more useful to monitor changes.
- Fixed bug when using `COUNT(DISTINCT ...)` on an empty table.
- Added support for the Chinese character set GBK.
- Fixed problem with `LOAD DATA INFILE` and `BLOB` columns.
- Added bit operator ~ (negation).
- Fixed problem with user-defined functions.

D.4.57. Changes in release 3.23.4 (28 Sep 1999)

- Inserting a `DATETIME` into a `TIME` column no longer try to store 'days' in it.
- Fixed problem with storage of float/double on little endian machines. (This affected `SUM()`.)
- Added connect timeout on TCP/IP connections.
- Fixed problem with `LIKE "%"` on an index that may have `NULL` values.

- `REVOKE ALL PRIVILEGES` didn't revoke all privileges.
- Allow creation of temporary tables with same name as the original table.
- When granting an account a `GRANT` option for a database, the account couldn't grant privileges to other users.
- New statement: `SHOW GRANTS FOR user` (by Sinisa).
- New `date_add` syntax: `date/datetime + INTERVAL # interval_type`. By Joshua Chamas.
- Fixed privilege check for `LOAD DATA REPLACE`.
- Automatic fixing of broken include files on Solaris 2.7
- Some configure issues to fix problems with big filesystem detection.
- `REGEXP` is now case-insensitive if you use non-binary strings.

D.4.58. Changes in release 3.23.3 (13 Sep 1999)

- Added patches for MIT-pthreads on NetBSD.
- Fixed range bug in `MyISAM`.
- `ASC` is now the default again for `ORDER BY`.
- Added `LIMIT` to `UPDATE`.
- Added `mysql_change_user()` function to the MySQL C API.
- Added character set to `SHOW VARIABLES`.
- Added support of `--[whitespace]` comments.
- Allow `INSERT INTO tbl_name VALUES ()`, that is, you may now specify an empty value list to insert a row in which each column is set to its default value.
- Changed `SUBSTRING(text FROM pos)` to conform to standard SQL. (Before this construct returned the rightmost `pos` characters.)
- `SUM()` with `GROUP BY` returned 0 on some systems.
- Changed output for `SHOW TABLE STATUS`.
- Added `DELAY_KEY_WRITE` option to `CREATE TABLE`.
- Allow `AUTO_INCREMENT` on any key part.
- Fixed problem with `YEAR(NOW())` and `YEAR(CURDATE())`.
- Added `CASE` construct.
- New `COALESCE()` function.

D.4.59. Changes in release 3.23.2 (09 Aug 1999)

- Fixed range optimizer bug: `SELECT * FROM tbl_name WHERE key_part1 >= const AND (key_part2 = const OR key_part2 = const)`. The bug was that some rows could be duplicated in the result.
- Running **myisamchk** without `-a` updated the index distribution incorrectly.
- `SET SQL_LOW_PRIORITY_UPDATES=1` was causing a parse error.
- You can now update index columns that are used in the `WHERE` clause. `UPDATE tbl_name SET KEY=KEY+1 WHERE KEY > 100`
- Date handling should now be a bit faster.
- Added handling of fuzzy dates (dates where day or month is 0), such as `'1999-01-00'`.
- Fixed optimization of `SELECT ... WHERE key_part1=const1 AND key_part_2=const2 AND key_part1=const4 AND key_part2=const4`; indextype should be `range` instead of `ref`.
- Fixed **egcs** 1.1.2 optimizer bug (when using `BLOB` values) on Linux Alpha.
- Fixed problem with `LOCK TABLES` combined with `DELETE FROM table`.
- `MyISAM` tables now allow keys on `NULL` and `BLOB/TEXT` columns.
- The following join is now much faster: `SELECT ... FROM t1 LEFT JOIN t2 ON ... WHERE t2.not_null_column IS NULL`.
- `ORDER BY` and `GROUP BY` can be done on functions.
- Changed handling of 'const_item' to allow handling of `ORDER BY RAND()`.
- Indexes are now used for `WHERE key_column = function`.
- Indexes are now used for `WHERE key_column = col_name` even if the columns are not identically packed.
- Indexes are now used for `WHERE col_name IS NULL`.
- Changed heap tables to be stored in low_byte_first order (to make it easy to convert to `MyISAM` tables)
- Automatic change of `HEAP` temporary tables to `MyISAM` tables in case of ``table is full'' errors.
- Added `--init-file=file_name` option to **mysqld**.
- Added `COUNT(DISTINCT value, [value, ...])`.
- `CREATE TEMPORARY TABLE` now creates a temporary table, in its own namespace, that is automatically deleted if connection is dropped.
- New reserved words (required for `CASE`): `CASE, THEN, WHEN, ELSE and END`.
- New functions `EXPORT_SET()` and `MD5()`.
- Support for the GB2312 Chinese character set.

D.4.60. Changes in release 3.23.1 (08 Jul 1999)

- Fixed some compilation problems.

D.4.61. Changes in release 3.23.0 (05 Jul 1999: Alpha)

- A new storage engine library (`MyISAM`) with a lot of new features. See Section 14.1, “The `MyISAM` Storage Engine”.
- You can create in-memory `HEAP` tables which are extremely fast for lookups.
- Support for big files (63-bit) on OSs that support big files.
- New function `LOAD_FILE(filename)` to get the contents of a file as a string value.
- New `<=>` operator that acts as `=` but returns TRUE if both arguments are `NULL`. This is useful for comparing changes between tables.
- Added the ODBC 3.0 `EXTRACT(interval FROM datetime)` function.
- Columns defined as `FLOAT(X)` are not rounded on storage and may be in scientific notation (1.0 E+10) when retrieved.
- `REPLACE` is now faster than before.
- Changed `LIKE` character comparison to behave as `=`; This means that `'e' LIKE 'é'` is now true. (If the line doesn't display correctly, the latter 'e' is a French 'e' with an acute accent above.)
- `SHOW TABLE STATUS` returns a lot of information about the tables.
- Added `LIKE` to the `SHOW STATUS` command.
- Added `Privileges` column to `SHOW COLUMNS`.
- Added `Packed` and `Comment` columns to `SHOW INDEX`.
- Added comments to tables (with `CREATE TABLE ... COMMENT 'xxx'`).
- Added `UNIQUE`, as in `CREATE TABLE tbl_name (col INT NOT NULL UNIQUE)`
- New create syntax: `CREATE TABLE tbl_name SELECT ...`
- New create syntax: `CREATE TABLE IF NOT EXISTS ...`
- Allow creation of `CHAR(0)` columns.
- `DATE_FORMAT()` now requires '`%`' before any format character.
- `DELAYED` is now a reserved word (sorry about that :( ).
- An example procedure is added: `analyse`, file: `sql_analyse.c`. This describes the data in your query. Try the following:

  ```
  SELECT ... FROM ...
  WHERE ... PROCEDURE ANALYSE([max_elements,[max_memory]])
  ```

This procedure is extremely useful when you want to check the data in your table!

- BINARY cast to force a string to be compared in case-sensitive fashion.
- Added --skip-show-database option to **mysqld**.
- Check whether a row has changed in an UPDATE now also works with BLOB/TEXT columns.
- Added the INNER join syntax. **Note**: This made INNER a reserved word!
- Added support for netmasks to the hostname in the MySQL grant tables. You can specify a netmask using the IP/NETMASK syntax.
- If you compare a NOT NULL DATE/DATETIME column with IS NULL, this is changed to a compare against 0 to satisfy some ODBC applications. (By <shreeve@uci.edu>.)
- NULL IN (...) now returns NULL instead of 0. This ensures that null_column NOT IN (...) doesn't match NULL values.
- Fix storage of floating-point values in TIME columns.
- Changed parsing of TIME strings to be more strict. Now the fractional second part is detected (and currently skipped). The following formats are supported:
  - [[DAYS] [H]H:]MM:]SS[.fraction]
  - [[[[[H]H]H]H]MM]SS[.fraction]
- Detect (and ignore) fractional second part from DATETIME.
- Added the LOW_PRIORITY attribute to LOAD DATA INFILE.
- The default index name now uses the same case as the column name on which the index name is based.
- Changed default number of connections to 100.
- Use bigger buffers when using LOAD DATA INFILE.
- DECIMAL(x,y) now works according to standard SQL.
- Added aggregate user-defined functions. Thanks to Andreas F. Bobak (<bobak@relog.ch>) for this!
- LAST_INSERT_ID() is now updated for INSERT INTO ... SELECT.
- Some small changes to the join table optimizer to make some joins faster.
- SELECT DISTINCT is much faster; it uses the new UNIQUE functionality in MyISAM. One difference compared to MySQL 3.22 is that the output of DISTINCT is no longer sorted.
- All C client API macros are now functions to make shared libraries more reliable. Because of this, you can no longer call mysql_num_fields() on a MYSQL object, you must use mysql_field_count() instead.
- Added use of LIBWRAP; patch by Henning P. Schmiedehausen.
- Don't allow AUTO_INCREMENT for other than numerical columns.

- Using AUTO_INCREMENT now automatically makes the column NOT NULL.
- Show NULL as the default value for AUTO_INCREMENT columns.
- Added SQL_BIG_RESULT; SQL_SMALL_RESULT is now default.
- Added a shared library RPM. This enhancement was contributed by David Fox (<dsfox@cogsci.ucsd.edu>).
- Added --enable-large-files and --disable-large-files options to **configure**. See configure.in for some systems where this is automatically turned off because of broken implementations.
- Upgraded readline to 4.0.
- New CREATE TABLE options: PACK_KEYS and CHECKSUM.
- Added --default-table-type option to **mysqld**.

D.5. Changes in release 3.22.x (Old; discontinued)

The 3.22 version has faster and safer connect code than version 3.21, as well as a lot of new nice enhancements. As there aren't really any major changes, upgrading from 3.21 to 3.22 should be very easy and painless. See Section 2.10.5, “Upgrading from Version 3.21 to 3.22”.

D.5.1. Changes in release 3.22.35

- Fixed problem with STD().
- Merged changes from the newest ISAM library from 3.23.
- Fixed problem with INSERT DELAYED.
- Fixed a bug core dump when using a LEFT JOIN/STRAIGHT_JOIN on a table with only one row.

D.5.2. Changes in release 3.22.34

- Fixed problem with GROUP BY on TINYBLOB columns; this caused bugzilla to not show rows in some queries.
- Had to do total recompile of the Windows binary version as VC++ didn't compile all relevant files for 3.22.33 :(

D.5.3. Changes in release 3.22.33

- Fixed problems in Windows when locking tables with LOCK TABLE.
- Quicker kill of SELECT DISTINCT queries.

D.5.4. Changes in release 3.22.32 (14 Feb 2000)

- Fixed problem when storing numbers in timestamps.
- Fix problem with time zones that have half hour offsets.
- Added **mysqlhotcopy**, a fast online hot-backup utility for local MySQL databases. By Tim Bunce.
- New more secure **mysqlaccess**. Thanks to Steve Harvey for this.
- Fixed security problem in the protocol regarding password checking.
- Fixed problem that affected queries that did arithmetic on `GROUP` functions.
- Fixed a bug in the `ISAM` code when deleting rows on tables with packed indexes.

D.5.5. Changes in release 3.22.31

- A few small fixes for the Windows version.

D.5.6. Changes in release 3.22.30 (11 Jan 2000)

- Fixed optimizer problem on `SELECT` when using many overlapping indexes.
- Disabled floating-point exceptions for FreeBSD to fix core dump when doing `SELECT FLOOR(POW(2,63))`.
- Added print of default arguments options to all clients.
- Fixed critical problem with the `WITH GRANT OPTION` option.
- Fixed non-critical Y2K problem when writing short date to log files.

D.5.7. Changes in release 3.22.29 (02 Jan 2000)

- Upgraded the configure and include files to match the latest 3.23 version. This should increase portability and make it easier to build shared libraries.
- Added latest patches to MIT-pthreads for NetBSD.
- Fixed problem with time zones that are < GMT -11.
- Fixed a bug when deleting packed keys in NISAM.
- Fixed problem that could cause MySQL to touch freed memory when doing very complicated `GROUP BY` queries.
- Fixed core dump if you got a crashed table where an `ENUM` field value was too big.
- Added `mysqlshutdown.exe` and `mysqlwatch.exe` to the Windows distribution.

- Fixed problem when doing `ORDER BY` on a reference key.
- Fixed that `INSERT DELAYED` doesn't update timestamps that are given.

D.5.8. Changes in release 3.22.28 (20 Oct 1999)

- Fixed problem with `LEFT JOIN` and `COUNT()` on a column which was declared `NULL` + and it had a `DEFAULT` value.
- Fixed core dump problem when using `CONCAT()` in a `WHERE` clause.
- Fixed problem with `AVG()` and `STD()` with `NULL` values.

D.5.9. Changes in release 3.22.27 (05 Oct 1999)

- Fixed prototype in `my_ctype.h` when using other character sets.
- Some configure issues to fix problems with big filesystem detection.
- Fixed problem when sorting on big `BLOB` columns.
- `ROUND()` now works on Windows.

D.5.10. Changes in release 3.22.26 (16 Sep 1999)

- Fixed core dump with empty `BLOB/TEXT` column argument to `REVERSE()`.
- Extended `/*! */` with version numbers.
- Changed `SUBSTRING(text FROM pos)` to conform to standard SQL. (Before this construct returned the rightmost 'pos' characters.)
- Fixed problem with `LOCK TABLES` combined with `DELETE FROM table`
- Fixed problem that `INSERT ... SELECT` didn't use `BIG_TABLES`.
- `SET SQL_LOW_PRIORITY_UPDATES=#` didn't work.
- Password wasn't updated correctly if privileges didn't change on: `GRANT ... IDENTIFIED BY`
- Fixed range optimizer bug in `SELECT * FROM` *`tbl_name`* `WHERE` *`key_part1 >= const`* `AND (`*`key_part2 = const`* `OR` *`key_part2 = const`*`)`.
- Fixed bug in compression key handling in `ISAM`.

D.5.11. Changes in release 3.22.25 (07 Jun 1999)

- Fixed some small problems with the installation.

D.5.12. Changes in release 3.22.24 (05 Jul 1999)

- `DATA` is no longer a reserved word.
- Fixed optimizer bug with tables with only one row.
- Fixed bug when using `LOCK TABLES tbl_name READ; FLUSH TABLES;`
- Applied some patches for HP-UX.
- **isamchk** should now work on Windows.
- Changed `configure` to not use big file handling on Linux as this crashes some Red Hat 6.0 systems

D.5.13. Changes in release 3.22.23 (08 Jun 1999)

- Upgraded to use Autoconf 2.13, Automake 1.4 and **libtool** 1.3.2.
- Better support for SCO in **configure**.
- Added option `--defaults-file=file_name` to option file handling to force use of only one specific option file.
- Extended `CREATE` syntax to ignore MySQL 3.23 keywords.
- Fixed deadlock problem when using `INSERT DELAYED` on a table locked with `LOCK TABLES`.
- Fixed deadlock problem when using `DROP TABLE` on a table that was locked by another thread.
- Add logging of `GRANT/REVOKE` commands in the update log.
- Fixed **isamchk** to detect a new error condition.
- Fixed bug in `NATURAL LEFT JOIN`.

D.5.14. Changes in release 3.22.22 (30 Apr 1999)

- Fixed problem in the C API when you called `mysql_close()` directly after `mysql_init()`.
- Better client error message when you can't open socket.
- Fixed `delayed_insert_thread` counting when you couldn't create a new delayed_insert thread.
- Fixed bug in `CONCAT()` with many arguments.
- Added patches for DEC 3.2 and SCO.
- Fixed path-bug when installing MySQL as a service on NT.
- MySQL on Windows is now compiled with VC++ 6.0 instead of with VC++ 5.0.
- New installation setup for MySQL on Windows.

D.5.15. Changes in release 3.22.21 (04 Apr 1999)

- Fixed problem with `DELETE FROM TABLE` when table was locked by another thread.
- Fixed bug in `LEFT JOIN` involving empty tables.
- Changed the `mysql.db` column from `CHAR(32)` to `CHAR(60)`.
- `MODIFY` and `DELAYED` are no longer reserved words.
- Fixed a bug when storing days in a `TIME` column.
- Fixed a problem with `Host '...' is not allowed to connect to this MySQL server` after one had inserted a new MySQL user with a `GRANT` command.
- Changed to use `TCP_NODELAY` also on Linux (should give faster TCP/IP connections).

D.5.16. Changes in release 3.22.20 (18 Mar 1999)

- Fixed `STD()` for big tables when result should be 0.
- The update log didn't have newlines on some operating systems.
- `INSERT DELAYED` had some garbage at end in the update log.

D.5.17. Changes in release 3.22.19 (01 Mar 1999)

- Fixed bug in **mysql_install_db** (from 3.22.17).
- Changed default key cache size to 8MB.
- Fixed problem with queries that needed temporary tables with `BLOB` columns.

D.5.18. Changes in release 3.22.18 (26 Feb 1999)

- Fixes a fatal problem in 3.22.17 on Linux; after `shutdown` not all threads died properly.
- Added option `-O flush_time=#` to **mysqld**. This is mostly useful on Windows and tells how often MySQL should close all unused tables and flush all updated tables to disk.
- Fixed problem that a `VARCHAR` column compared with `CHAR` column didn't use keys efficiently.

D.5.19. Changes in release 3.22.17 (22 Feb 1999: Production)

- Fixed a core dump problem when using `--log-update` and connecting without a default database.

- Fixed some **configure** and portability problems.
- Using `LEFT JOIN` on tables that had circular dependencies caused **mysqld** to hang forever.

D.5.20. Changes in release 3.22.16 (05 Feb 1999)

- **mysqladmin processlist** could kill the server if a new user logged in.
- `DELETE FROM tbl_name WHERE key_column=col_name` didn't find any matching rows. Fixed.
- `DATE_ADD(column, ...)` didn't work.
- `INSERT DELAYED` could deadlock with status `upgrading lock`.
- Extended `ENCRYPT()` to take longer salt strings than 2 characters.
- `longlong2str` is now much faster than before. For `Intel x86` platforms, this function is written in optimized assembler.
- Added the `MODIFY` keyword to `ALTER TABLE`.

D.5.21. Changes in release 3.22.15 (27 Jan 1999)

- `GRANT` used with `IDENTIFIED BY` didn't take effect until privileges were flushed.
- Name change of some variables in `SHOW STATUS`.
- Fixed problem with `ORDER BY` with 'only index' optimization when there were multiple key definitions for a used column.
- `DATE` and `DATETIME` columns are now up to 5 times faster than before.
- `INSERT DELAYED` can be used to let the client do other things while the server inserts rows into a table.
- `LEFT JOIN USING (col1,col2)` didn't work if one used it with tables from 2 different databases.
- `LOAD DATA LOCAL INFILE` didn't work in the Unix version because of a missing file.
- Fixed problems with `VARCHAR`/`BLOB` on very short rows (< 4 bytes); error 127 could occur when deleting rows.
- Updating `BLOB/TEXT` through formulas didn't work for short (< 256 char) strings.
- When you did a `GRANT` on a new host, **mysqld** could die on the first connect from this host.
- Fixed bug when one used `ORDER BY` on column name that was the same name as an alias.
- Added `BENCHMARK(loop_count,expression)` function to time expressions.

D.5.22. Changes in release 3.22.14 (01 Jan 1999: Gamma)

- Allow empty arguments to **mysqld** to make it easier to start from shell scripts.
- Setting a `TIMESTAMP` column to `NULL` didn't record the timestamp value in the update log.
- Fixed lock handler bug when one did `INSERT INTO TABLE ... SELECT ... GROUP BY`.
- Added a patch for `localtime_r()` on Windows so that it no longer crashes if your date is > 2039, but instead returns a time of all zero.
- Names for user-defined functions are no longer case-sensitive.
- Added escape of `^Z` (ASCII 26) to `\Z` as `^Z` doesn't work with pipes on Windows.
- **mysql_fix_privileges** adds a new column to the `mysql.func` to support aggregate user-defined functions in future MySQL releases.

D.5.23. Changes in release 3.22.13 (16 Dec 1998)

- Saving `NOW()`, `CURDATE()` or `CURTIME()` directly in a column didn't work.
- `SELECT COUNT(*) ... LEFT JOIN ...` didn't work with no `WHERE` part.
- Updated `config.guess` to allow MySQL to configure on UnixWare 7.1.x.
- Changed the implementation of `pthread_cond()` on the Windows version. `get_lock()` now correctly times out on Windows!

D.5.24. Changes in release 3.22.12 (09 Dec 1998)

- Fixed problem when using `DATE_ADD()` and `DATE_SUB()` in a `WHERE` clause.
- You can now set the password for a user with the `GRANT ... TO user IDENTIFIED BY 'password'` syntax.
- Fixed bug in `GRANT` checking with `SELECT` on many tables.
- Added missing file **mysql_fix_privilege_tables** to the RPM distribution. This is not run by default because it relies on the client package.
- Added option `SQL_SMALL_RESULT` to `SELECT` to force use of fast temporary tables when you know that the result set is small.
- Allow use of negative real numbers without a decimal point.
- Day number is now adjusted to maximum days in month if the resulting month after `DATE_ADD`/`DATE_SUB()` doesn't have enough days.
- Fix that `GRANT` compares columns in case-insensitive fashion.
- Fixed a bug in `sql_list.h` that made `ALTER TABLE` dump core in some contexts.
- The hostname in `user@hostname` can now include `'.'` and `'-'` without quotes in the context of the `GRANT`, `REVOKE` and `SET PASSWORD FOR ...` statements.

- Fix for **isamchk** for tables which need big temporary files.

D.5.25. Changes in release 3.22.11 (24 Nov 1998)

- **Important**: You must run the **mysql_fix_privilege_tables** script when you upgrade to this version! This is needed because of the new `GRANT` system. If you don't do this, you get `Access denied` when you try to use `ALTER TABLE`, `CREATE INDEX`, or `DROP INDEX`.
- `GRANT` to allow/deny users table and column access.
- Changed `USER()` to return a value in `user@host` format. Formerly it returned only `user`.
- Changed the syntax for how to set `PASSWORD` for another user.
- New command `FLUSH STATUS` that resets most status variables to zero.
- New status variables: `aborted_threads`, `aborted_connects`.
- New option variable: `connection_timeout`.
- Added support for Thai sorting (by Pruet Boonma `<pruet@ds90.intanon.nectec.or.th>`).
- Slovak and Japanese error messages.
- Configuration and portability fixes.
- Added option `SET SQL_WARNINGS=1` to get a warning count also for simple (single-row) inserts.
- MySQL now uses `SIGTERM` instead of `SIGQUIT` with shutdown to work better on FreeBSD.
- Added option `\G` (print vertically) to **mysql**.
- `SELECT HIGH_PRIORITY ...` killed **mysqld**.
- `IS NULL` on a `AUTO_INCREMENT` column in a `LEFT JOIN` didn't work as expected.
- New function `MAKE_SET()`.

D.5.26. Changes in release 3.22.10 (04 Nov 1998)

- **mysql_install_db** no longer starts the MySQL server! You should start **mysqld** with **safe_mysqld** after installing it! The MySQL RPM , however, starts the server as before.
- Added `--bootstrap` option to **mysqld** and recoded **mysql_install_db** to use it. This makes it easier to install MySQL with RPMs.
- Changed `+`, `-` (sign and minus), `*`, `/`, `%`, `ABS()` and `MOD()` to be `BIGINT` aware (64-bit safe).
- Fixed a bug in `ALTER TABLE` that caused **mysqld** to crash.
- MySQL now always reports the conflicting key values when a duplicate key entry occurs. (Before this was only reported for `INSERT`.)

- New syntax: `INSERT INTO tbl_name SET col_name=value, col_name=value, ...`
- Most errors in the `.err` log are now prefixed with a time stamp.
- Added option `MYSQL_INIT_COMMAND` to `mysql_options()` to make a query on connect or reconnect.
- Added option `MYSQL_READ_DEFAULT_FILE` and `MYSQL_READ_DEFAULT_GROUP` to `mysql_options()` to read the following parameters from the MySQL option files: `port`, `socket`, `compress`, `password`, `pipe`, `timeout`, `user`, `init-command`, `host` and `database`.
- Added `maybe_null` to the UDF structure.
- Added option `IGNORE` to `INSERT` statements with many rows.
- Fixed some problems with sorting of the `koi8` character sets; users of `koi8` **must** run **isamchk -rq** on each table that has an index on a `CHAR` or `VARCHAR` column.
- New script **mysql_setpermission**, by Luuk de Boer. It allows easy creation of new users with permissions for specific databases.
- Allow use of hexadecimal strings (0x...) when specifying a constant string (like in the column separators with `LOAD DATA INFILE`).
- Ported to OS/2 (thanks to Antony T. Curtis `<antony.curtis@olcs.net>`).
- Added more variables to `SHOW STATUS` and changed format of output to be like `SHOW VARIABLES`.
- Added `extended-status` command to **mysqladmin** which shows the new status variables.

D.5.27. Changes in release 3.22.9 (19 Oct 1998)

- `SET SQL_LOG_UPDATE=0` caused a lockup of the server.
- New SQL statement: `FLUSH [ TABLES | HOSTS | LOGS | PRIVILEGES ] [, ...]`
- New SQL statement: `KILL thread_id`.
- Added casts and changed include files to make MySQL easier to compile on AIX and DEC OSF/1 4.x
- Fixed conversion problem when using `ALTER TABLE` from a `INT` to a short `CHAR()` column.
- Added `SELECT HIGH_PRIORITY`; this gets a lock for the `SELECT` even if there is a thread waiting for another `SELECT` to get a `WRITE LOCK`.
- Moved `wild_compare()` to string class to be able to use `LIKE` on `BLOB`/`TEXT` columns with `\0`.
- Added `ESCAPE` option to `LIKE`.
- Added a lot more output to **mysqladmin debug**.
- You can now start **mysqld** on Windows with the `--flush` option. This flushes all tables to disk

after each update. This makes things much safer on the Windows platforms but also **much** slower.

D.5.28. Changes in release 3.22.8 (06 Oct 1998)

- Czech character sets should now work much better.
- `DATE_ADD()` and `DATE_SUB()` didn't work with group functions.
- **mysql** now also tries to reconnect on `USE database` commands.
- Fix problem with `ORDER BY` and `LEFT JOIN` and `const` tables.
- Fixed problem with `ORDER BY` if the first `ORDER BY` column was a key and the rest of the `ORDER BY` columns wasn't part of the key.
- Fixed a big problem with `OPTIMIZE TABLE`.
- MySQL clients on NT now by default first try to connect with named pipes and after this with TCP/IP.
- Fixed a problem with `DROP TABLE` and **mysqladmin shutdown** on Windows (a fatal bug from 3.22.6).
- Fixed problems with `TIME columns` and negative strings.
- Added an extra thread signal loop on shutdown to avoid some error messages from the client.
- MySQL now uses the next available number as extension for the update log file.
- Added patches for UNIXWARE 7.

D.5.29. Changes in release 3.22.7 (21 Sep 1998: Beta)

- Added `LIMIT` clause for the `DELETE` statement.
- You can now use the `/*! ... */` syntax to hide MySQL-specific keywords when you write portable code. MySQL parses the code inside the comments as if the surrounding `/*!` and `*/` comment characters didn't exist.
- `OPTIMIZE TABLE` *`tbl_name`* can now be used to reclaim disk space after many deletes. Currently, this uses `ALTER TABLE` to regenerate the table, but in the future it will use an integrated **isamchk** for more speed.
- Upgraded **libtool** to make the configuration process more portable.
- Fixed slow `UPDATE` and `DELETE` operations when using `DATETIME` or `DATE` keys.
- Changed optimizer to make it better at deciding when to do a full join and when using keys.
- You can now use **mysqladmin proc** to display information about your own threads. Only users with the `PROCESS` privilege can get information about all threads. (In 4.0.2, you need the `SUPER` privilege for this.)
- Added handling of formats `YYMMDD`, `YYYYMMDD`, `YYMMDDHHMMSS` for numbers when using `DAT-`

ETIME and TIMESTAMP types. (Formerly these formats only worked with strings.)

- Added connect option CLIENT_IGNORE_SPACE to allow use of spaces after function names and before '(' (Powerbuilder requires this). This makes all function names reserved words.
- Added the --log-long-format option to **mysqld** to enable timestamps and INSERT_IDs in the update log.
- Added --where option to **mysqldump** (patch by Jim Faucette).
- The lexical analyzer now uses ``perfect hashing'' for faster parsing of SQL statements.

D.5.30. Changes in release 3.22.6 (31 Aug 1998)

- Faster **mysqldump**.
- For the LOAD DATA INFILE statement, you can now use the new LOCAL keyword to read the file from the client. **mysqlimport** automatically uses LOCAL when importing with the TCP/IP protocol.
- Fixed small optimize problem when updating keys.
- Changed makefiles to support shared libraries.
- MySQL-NT can now use named pipes, which means that you can now use MySQL-NT without having to install TCP/IP.

D.5.31. Changes in release 3.22.5 (20 Aug 1998: Alpha)

- All table lock handing is changed to avoid some very subtle deadlocks when using DROP TABLE, ALTER TABLE, DELETE FROM TABLE and **mysqladmin flush-tables** under heavy usage. Changed locking code to get better handling of locks of different types.
- Updated DBI to 1.00 and DBD to 1.2.0.
- Added a check that the error message file contains error messages suitable for the current version of **mysqld**. (To avoid errors if you accidentally try to use an old error message file.)
- All count structures in the client (affected_rows(), insert_id(), ...) are now of type BIGINT to allow 64-bit values to be used. This required a minor change in the MySQL protocol which should affect only old clients when using tables with AUTO_INCREMENT values > 16MB.
- The return type of mysql_fetch_lengths() has changed from uint * to ulong *. This may give a warning for old clients but should work on most machines.
- Change mysys and dbug libraries to allocate all thread variables in one struct. This makes it easier to make a threaded libmysql.dll library.
- Use the result from gethostname() (instead of uname()) when constructing .pid filenames.
- New better compressed client/server protocol.
- COUNT(), STD() and AVG() are extended to handle more than 4GB rows.
- You can now store values in the range -838:59:59 <= x <= 838:59:59 in a TIME column.

- **Warning: Incompatible change!!** If you set a TIME column to too short a value, MySQL now assumes the value is given as: [[[D]HH:]MM:]SS instead of HH[:MM[:SS]].
- TIME_TO_SEC() and SEC_TO_TIME() can now handle negative times and hours up to 32767.
- Added new option SET SQL_LOG_UPDATE={0|1} to allow users with the PROCESS privilege to bypass the update log. (Modified patch from Sergey A Mukhin <violet@rosnet.net>.)
- Fixed fatal bug in LPAD().
- Initialize line buffer in mysql.cc to make BLOB reading from pipes safer.
- Added -O max_connect_errors=# option to **mysqld**. Connect errors are now reset for each correct connection.
- Increased the default value of max_allowed_packet to 1M in **mysqld**.
- Added --low-priority-updates option to **mysqld**, to give table-modifying operations (INSERT, REPLACE, UPDATE, DELETE) lower priority than retrievals. You can now use {INSERT | REPLACE | UPDATE | DELETE} LOW_PRIORITY ... You can also use SET SQL_LOW_PRIORITY_UPDATES={0|1} to change the priority for one thread. One side effect is that LOW_PRIORITY is now a reserved word. :(
- Add support for INSERT INTO table ... VALUES(...),(...),(...), to allow inserting multiple rows with a single statement.
- INSERT INTO *tbl_name* is now also cached when used with LOCK TABLES. (Previously only INSERT ... SELECT and LOAD DATA INFILE were cached.)
- Allow GROUP BY functions with HAVING:

  ```
  mysql> SELECT col FROM table GROUP BY col HAVING COUNT(*)>0;
  ```

- **mysqld** now ignores trailing ';' characters in queries. This is to make it easier to migrate from some other SQL servers that require the trailing ';'.
- Fix for corrupted fixed-format output generated by SELECT INTO OUTFILE.
- **Warning: Incompatible change!** Added Oracle GREATEST() and LEAST() functions. You must now use these instead of the MAX() and MIN() functions to get the largest/smallest value from a list of values. These can now handle REAL, BIGINT and string (CHAR or VARCHAR) values.
- **Warning: Incompatible change!** DAYOFWEEK() had offset 0 for Sunday. Changed the offset to 1.
- Give an error for queries that mix GROUP BY columns and fields when there is no GROUP BY specification.
- Added --vertical option to **mysql**, for printing results in vertical mode.
- Index-only optimization; some queries are now resolved using only indexes. Until MySQL 4.0, this works only for numeric columns. See Section 7.4.5, “How MySQL Uses Indexes”.
- Lots of new benchmarks.
- A new C API chapter and lots of other improvements in the manual.

D.5.32. Changes in release 3.22.4 (06 Jul 1998: Beta)

- Added `--tmpdir` option to **mysqld**, for specifying the location of the temporary file directory.
- MySQL now automatically changes a query from an ODBC client:

  ```
  SELECT ... FROM table WHERE auto_increment_column IS NULL
  ```

  to:

  ```
  SELECT ... FROM table WHERE auto_increment_column == LAST_INSERT_ID()
  ```

  This allows some ODBC programs (Delphi, Access) to retrieve the newly inserted row to fetch the `AUTO_INCREMENT` value.
- `DROP TABLE` now waits for all users to free a table before deleting it.
- Fixed small memory leak in the new connect protocol.
- New functions `BIN()`, `OCT()`, `HEX()` and `CONV()` for converting between different number bases.
- Added function `SUBSTRING()` with two arguments.
- If you created a table with a record length smaller than 5, you couldn't delete rows from the table.
- Added optimization to remove `const` reference tables from `ORDER BY` and `GROUP BY`.
- **mysqld** now automatically disables system locking on Linux and Windows, and for systems that use MIT-pthreads. You can force the use of locking with the `--enable-external-locking` option.
- Added `--console` option to **mysqld**, to force a console window (for error messages) when using Windows.
- Fixed table locks for Windows.
- Allow '$' in identifiers.
- Changed name of user-specific configuration file from `my.cnf` to `.my.cnf` (Unix only).
- Added `DATE_ADD()` and `DATE_SUB()` functions.

D.5.33. Changes in release 3.22.3 (30 Jun 1998)

- Fixed a lock problem (bug in MySQL 3.22.1) when closing temporary tables.
- Added missing `mysql_ping()` to the client library.
- Added `--compress` option to all MySQL clients.
- Changed `byte` to `char` in `mysql.h` and `mysql_com.h`.

D.5.34. Changes in release 3.22.2

- Searching on multiple constant keys that matched more than 30% of the rows didn't always use the best possible key.

- New functions <<, >>, RPAD() and LPAD().
- You can now save default options (like passwords) in a configuration file (my.cnf).
- Lots of small changes to get ORDER BY to work when no records are found when using fields that are not in GROUP BY (MySQL extension).
- Added --chroot option to **mysqld**, to start **mysqld** in a chroot environment (by Nikki Chumakov <nikkic@cityline.ru>).
- Trailing spaces are now ignored when comparing case-sensitive strings; this should fix some problems with ODBC and flag 512!
- Fixed a core dump bug in the range optimizer.
- Added --one-thread option to **mysqld**, for debugging with LinuxThreads (or glibc). (This replaces the -T32 flag)
- Added DROP TABLE IF EXISTS to prevent an error from occurring if the table doesn't exist.
- IF and EXISTS are now reserved words (they would have to be sooner or later).
- Added lots of new options to **mysqldump**.
- Server error messages are now in mysqld_error.h.
- The client/server protocol now supports compression.
- All bugfixes from MySQL 3.21.32.

D.5.35. Changes in release 3.22.1 (Jun 1998)

- Added new C API function mysql_ping().
- Added new API functions mysql_init() and mysql_options(). You now MUST call mysql_init() before you call mysql_real_connect(). You don't have to call mysql_init() if you only use mysql_connect().
- Added mysql_options(...,MYSQL_OPT_CONNECT_TIMEOUT,...) so you can set a timeout for connecting to a server.
- Added --timeout option to **mysqladmin**, as a test of mysql_options().
- Added AFTER column and FIRST options to ALTER TABLE ... ADD columns. This makes it possible to add a new column at some specific location within a row in an existing table.
- WEEK() now takes an optional argument to allow handling of weeks when the week starts on Monday (some European countries). By default, WEEK() assumes the week starts on Sunday.
- TIME columns weren't stored properly (bug in MySQL 3.22.0).
- UPDATE now returns information about how many rows were matched and updated, and how many ``warnings'' occurred when doing the update.
- Fixed incorrect result from FORMAT(-100,2).
- ENUM and SET columns were compared in binary (case-sensitive) fashion; changed to be case-

insensitive.

D.5.36. Changes in release 3.22.0 (18 May 1998: Alpha)

- New (backward-compatible) connect protocol that allows you to specify the database to use when connecting, to get much faster connections to a specific database.

  The `mysql_real_connect()` call is changed to:

  ```
  mysql_real_connect(MYSQL *mysql, const char *host, const char *user,
                     const char *passwd, const char *db, uint port,
                     const char *unix_socket, uint client_flag)
  ```

- Each connection is handled by its own thread, rather than by the master `accept()` thread. This fixes permanently the telnet bug that was a topic on the mail list some time ago.

- All TCP/IP connections are now checked with backward-resolution of the hostname to get better security. **mysqld** now has a local hostname resolver cache so connections should actually be faster than before, even with this feature.

- A site automatically is blocked from future connections if someone repeatedly connects with an ``improper header'' (like when one uses telnet).

- You can now refer to tables in different databases with references of the form *`tbl_name@db_name`* or *`db_name.tbl_name`*. This makes it possible to give a user read access to some tables and write access to others simply by keeping them in different databases!

- Added `--user` option to **mysqld**, to allow it to run as another Unix user (if it is started as the Unix `root` user).

- Added caching of users and access rights (for faster access rights checking)

- Normal users (not anonymous ones) can change their password with **mysqladmin password "*newpwd*"**. This uses encrypted passwords that are not logged in the normal MySQL log!

- All important string functions are now coded in assembler for x86 Linux machines. This gives a speedup of 10% in many cases.

- For tables that have many columns, the column names are now hashed for much faster column name lookup (this speeds up some benchmark tests a lot!)

- Some benchmarks are changed to get better individual timing. (Some loops were so short that a specific test took < 2 seconds. The loops have been changed to take about 20 seconds to make it easier to compare different databases. A test that took 1-2 seconds before now takes 11-24 seconds, which is much better)

- Re-arranged `SELECT` code to handle some very specific queries involving group functions (like `COUNT(*)`) without a `GROUP BY` but with `HAVING`. The following now works:

  ```
  mysql> SELECT COUNT(*) as C FROM table HAVING C > 1;
  ```

- Changed the protocol for field functions to be faster and avoid some calls to `malloc()`.

- Added `-T32` option to **mysqld**, for running all queries under the main thread. This makes it possible to debug **mysqld** under Linux with **gdb**!

- Added optimization of `not_null_column IS NULL` (needed for some Access queries).
- Allow `STRAIGHT_JOIN` to be used between two tables to force the optimizer to join them in a specific order.
- String functions now return `VARCHAR` rather than `CHAR` and the column type is now `VARCHAR` for fields saved as `VARCHAR`. This should make the MyODBC driver better, but may break some old MySQL clients that don't handle `FIELD_TYPE_VARCHAR` the same way as `FIELD_TYPE_CHAR`.
- `CREATE INDEX` and `DROP INDEX` are now implemented through `ALTER TABLE`. `CREATE TABLE` is still the recommended (fast) way to create indexes.
- Added `--set-variable` option `wait_timeout` to **mysqld**.
- Added time column to **mysqladmin processlist** to show how long a query has taken or how long a thread has slept.
- Added lots of new variables to `show variables` and some new to `show status`.
- Added new type `YEAR`. `YEAR` is stored in 1 byte with allowable values of 0, and 1901 to 2155.
- Added new `DATE` type that is stored in 3 bytes rather than 4 bytes. All new tables are created with the new date type if you don't use the `--old-protocol` option to **mysqld**.
- Fixed bug in record caches; for some queries, you could get `Error from table handler: #` on some operating systems.
- Added `--enable-assembler` option to **configure**, for x86 machines (tested on Linux + **gcc**). This enables assembler functions for the most important string functions for more speed!

D.6. Changes in release 3.21.x

MySQL 3.21 is quite old now, and should be avoided if possible. This information is kept here for historical purposes only.

D.6.1. Changes in release 3.21.33 (08 Jul 1998)

- Fixed problem when sending `SIGHUP` to **mysqld**; **mysqld** dumped core when starting from boot on some systems.
- Fixed problem with losing a little memory for some connections.
- `DELETE FROM` *`tbl_name`* without a `WHERE` condition is now done the long way when you use `LOCK TABLES` or if the table is in use, to avoid race conditions.
- `INSERT INTO TABLE (timestamp_column) VALUES (NULL);` didn't set timestamp.

D.6.2. Changes in release 3.21.32 (30 Jun 1998)

- Fixed some possible race conditions when doing many reopen/close on the same tables under heavy load! This can happen if you execute **mysqladmin refresh** often. This could in some very rare cases corrupt the header of the index file and cause error 126 or 138.

- Fixed fatal bug in `refresh()` when running with the `--skip-external-locking` option. There was a ``very small'' time gap after a **mysqladmin refresh** when a table could be corrupted if one thread updated a table while another thread did **mysqladmin refresh** and another thread started a new update on the same table before the first thread had finished. A refresh (or `--flush-tables`) now does not return until all used tables are closed!
- `SELECT DISTINCT` with a `WHERE` clause that didn't match any rows returned a row in some contexts (bug only in 3.21.31).
- `GROUP BY` + `ORDER BY` returned one empty row when no rows where found.
- Fixed a bug in the range optimizer that wrote `Use_count: Wrong count for ...` in the error log file.

D.6.3. Changes in release 3.21.31 (10 Jun 1998)

- Fixed a sign extension problem for the `TINYINT` type on Irix.
- Fixed problem with `LEFT("constant_string",function)`.
- Fixed problem with `FIND_IN_SET()`.
- `LEFT JOIN` dumped core if the second table is used with a constant `WHERE/ON` expression that uniquely identifies one record.
- Fixed problems with `DATE_FORMAT()` and incorrect dates. `DATE_FORMAT()` now ignores `'%'` to make it possible to extend it more easily in the future.

D.6.4. Changes in release 3.21.30

- **mysql** now returns an exit code > 0 if the query returned an error.
- Saving of command-line history to file in **mysql** client. By Tommy Larsen `<tommy@mix.hive.no>`.
- Fixed problem with empty lines that were ignored in `mysql.cc`.
- Save the pid of the signal handler thread in the pid file instead of the pid of the main thread.
- Added patch by `<tommy@valley.ne.jp>` to support Japanese characters SJIS and UJIS.
- Changed **safe_mysqld** to redirect startup messages to *`host_name`*`.err` instead of *`host_name`*`.log` to reclaim file space on **mysqladmin refresh**.
- `ENUM` always had the first entry as default value.
- `ALTER TABLE` wrote two entries to the update log.
- `sql_acc()` now closes the `mysql` grant tables after a reload to save table space and memory.
- Changed `LOAD DATA` to use less memory with tables and `BLOB` columns.
- Sorting on a function which made a division / 0 produced a wrong set in some cases.

- Fixed `SELECT` problem with `LEFT()` when using the `czech` character set.
- Fixed problem in **isamchk**; it couldn't repair a packed table in a very unusual case.
- `SELECT` statements with `&` or `|` (bit functions) failed on columns with `NULL` values.
- When comparing a field = field, where one of the fields was a part key, only the length of the part key was compared.

D.6.5. Changes in release 3.21.29

- `LOCK TABLES` + `DELETE from` *`tbl_name`* never removed locks properly.
- Fixed problem when grouping on an `OR` function.
- Fixed permission problem with `umask()` and creating new databases.
- Fixed permission problem on result file with `SELECT ... INTO OUTFILE ...`
- Fixed problem in range optimizer (core dump) for a very complex query.
- Fixed problem when using `MIN(integer)` or `MAX(integer)` in `GROUP BY`.
- Fixed bug on Alpha when using integer keys. (Other keys worked on Alpha.)
- Fixed bug in `WEEK("XXXX-xx-01")`.

D.6.6. Changes in release 3.21.28

- Fixed socket permission (clients couldn't connect to Unix socket on Linux).
- Fixed bug in record caches; for some queries, you could get `Error from table handler: #` on some operating systems.

D.6.7. Changes in release 3.21.27

- Added user level lock functions `GET_LOCK(string,timeout)`, `RELEASE_LOCK(string)`.
- Added `Opened_tables` to `show status`.
- Changed connect timeout to 3 seconds to make it somewhat harder for crackers to kill **mysqld** through telnet + TCP/IP.
- Fixed bug in range optimizer when using `WHERE` *`key_part_1 >= something`* `AND` *`key_part_2 <= something_else`*.
- Changed **configure** for detection of FreeBSD 3.0 9803xx and above
- `WHERE` with *`string_col_key = constant_string`* didn't always find all rows if the column had many values differing only with characters of the same sort value (like e and e with an accent).

- Strings keys looked up with 'ref' were not compared in case-sensitive fashion.
- Added `umask()` to make log files non-readable for normal users.
- Ignore users with old (8-byte) password on startup if not using `--old-protocol` option to **mysqld**.
- `SELECT` which matched all key fields returned the values in the case of the matched values, not of the found values. (Minor problem.)

D.6.8. Changes in release 3.21.26

- `FROM_DAYS(0)` now returns "0000-00-00".
- In `DATE_FORMAT()`, PM and AM were swapped for hours 00 and 12.
- Extended the default maximum key size to 256.
- Fixed bug when using `BLOB`/`TEXT` in `GROUP BY` with many tables.
- An `ENUM` field that is not declared `NOT NULL` has `NULL` as the default value. (Previously, the default value was the first enumeration value.)
- Fixed bug in the join optimizer code when using many part keys on the same key: `INDEX (Organization,Surname(35),Initials(35))`.
- Added some tests to the table order optimizer to get some cases with `SELECT ... FROM many_tables` much faster.
- Added a retry loop around `accept()` to possibly fix some problems on some Linux machines.

D.6.9. Changes in release 3.21.25

- Changed `typedef 'string'` to `typedef 'my_string'` for better portability.
- You can now kill threads that are waiting on a disk-full condition.
- Fixed some problems with user-defined functions.
- Added long options to **isamchk**. Try **isamchk --help**.
- Fixed a bug when using 8 bytes long (alpha); `filesort()` didn't work. Affects `DISTINCT`, `ORDER BY` and `GROUP BY` on 64-bit processors.

D.6.10. Changes in release 3.21.24

- Dynamic loadable functions. Based on source from Alexis Mikhailov.
- You couldn't delete from a table if no one had done a `SELECT` on the table.
- Fixed problem with range optimizer with many `OR` operators on key parts inside each other.

- Recoded `MIN()` and `MAX()` to work properly with strings and `HAVING`.
- Changed default umask value for new files from `0664` to `0660`.
- Fixed problem with `LEFT JOIN` and constant expressions in the `ON` part.
- Added Italian error messages from `<brenno@dewinter.com>`.
- **configure** now works better on OSF/1 (tested on 4.0D).
- Added hooks to allow `LIKE` optimization with international character support.
- Upgraded `DBI` to 0.93.

D.6.11. Changes in release 3.21.23

- The following symbols are now reserved words: `TIME`, `DATE`, `TIMESTAMP`, `TEXT`, `BIT`, `ENUM`, `NO`, `ACTION`, `CHECK`, `YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE`, `SECOND`, `STATUS`, `VARIABLES`.
- Setting a `TIMESTAMP` to `NULL` in `LOAD DATA INFILE ...` didn't set the current time for the `TIMESTAMP`.
- Fix `BETWEEN` to recognize binary strings. Now `BETWEEN` is case-sensitive.
- Added `--skip-thread-priority` option to **mysqld**, for systems where **mysqld** thread scheduling doesn't work properly (BSDI 3.1).
- Added ODBC functions `DAYNAME()` and `MONTHNAME()`.
- Added function `TIME_FORMAT()`. This works like `DATE_FORMAT()`, but takes a time string (`'HH:MM:SS'`) as argument.
- Fixed unlikely(?) key optimizer bug when using `OR` operators of key parts inside `AND` expressions.
- Added `variables` command to **mysqladmin**.
- A lot of small changes to the binary releases.
- Fixed a bug in the new protocol from MySQL 3.21.20.
- Changed `ALTER TABLE` to work with Windows (Windows can't rename open files). Also fixed a couple of small bugs in the Windows version.
- All standard MySQL clients are now ported to MySQL for Windows.
- MySQL can now be started as a service on NT.

D.6.12. Changes in release 3.21.22

- Starting with this version, all MySQL distributions are configured, compiled and tested with `crash-me` and the benchmarks on the following platforms: SunOS 5.6 sun4u, SunOS 5.5.1 sun4u, SunOS 4.14 sun4c, SunOS 5.6 i86pc, Irix 6.3 mips5k, HP-UX 10.20 hppa, AIX 4.2.1 ppc, OSF/1 V4.0 alpha, FreeBSD 2.2.2 i86pc and BSDI 3.1 i386.

- Fix `COUNT(*)` problems when the `WHERE` clause didn't match any records. (Bug from 3.21.17.)
- Removed that `NULL = NULL` is true. Now you must use `IS NULL` or `IS NOT NULL` to test whether a value is `NULL`. (This is according to standard SQL but may break old applications that are ported from `mSQL`.) You can get the old behavior by compiling with `-DmSQL_COMPLIANT`.
- Fixed bug that caused core-dump when using many `LEFT OUTER JOIN` clauses.
- Fixed bug in `ORDER BY` on string formula with possible `NULL` values.
- Fixed problem in range optimizer when using `<=` on sub index.
- Added functions `DAYOFYEAR()`, `DAYOFMONTH()`, `MONTH()`, `YEAR()`, `WEEK()`, `QUARTER()`, `HOUR()`, `MINUTE()`, `SECOND()` and `FIND_IN_SET()`.
- Added `SHOW VARIABLES` command.
- Added support of ``long constant strings'' from standard SQL:

  ```
  mysql> SELECT 'first ' 'second';       -> 'first second'
  ```

- Upgraded Msql-Mysql-modules to 1.1825.
- Upgraded **mysqlaccess** to 2.02.
- Fixed problem with Russian character set and `LIKE`.
- Ported to OpenBSD 2.1.
- New Dutch error messages.

D.6.13. Changes in release 3.21.21a

- Configure changes for some operating systems.

D.6.14. Changes in release 3.21.21

- Fixed optimizer bug when using `WHERE data_field = date_field2 AND date_field2 = constant`.
- Added `SHOW STATUS` command.
- Removed `manual.ps` from the source distribution to make it smaller.

D.6.15. Changes in release 3.21.20

- Changed the maximum table name and column name lengths from 32 to 64.
- Aliases can now be of ``any'' length.
- Fixed **mysqladmin stat** to return the right number of queries.

- Changed protocol (downward compatible) to mark if a column has the `AUTO_INCREMENT` attribute or is a `TIMESTAMP`. This is needed for the new Java driver.
- Added Hebrew sorting order by Zeev Suraski.
- Solaris 2.6: Fixed **configure** bugs and increased maximum table size from 2GB to 4GB.

D.6.16. Changes in release 3.21.19

- Upgraded `DBD` to 1.1823. This version implements `mysql_use_result` in `DBD-Mysql`.
- Benchmarks updated for empress (by Luuk).
- Fixed a case of slow range searching.
- Configure fixes (`Docs` directory).
- Added function `REVERSE()` (by Zeev Suraski).

D.6.17. Changes in release 3.21.18

- Issue error message if client C functions are called in wrong order.
- Added automatic reconnect to the `libmysql.c` library. If a write command fails, an automatic reconnect is done.
- Small sort sets no longer use temporary files.
- Upgraded `DBI` to 0.91.
- Fixed a couple of problems with `LEFT OUTER JOIN`.
- Added `CROSS JOIN` syntax. `CROSS` is now a reserved word.
- Recoded **yacc**/**bison** stack allocation to be even safer and to allow MySQL to handle even bigger expressions.
- Fixed a couple of problems with the update log.
- `ORDER BY` was slow when used with key ranges.

D.6.18. Changes in release 3.21.17

- Changed documentation string of `--with-unix-socket-path` to avoid confusion.
- Added ODBC and standard SQL style `LEFT OUTER JOIN`.
- The following are new reserved words: `LEFT`, `NATURAL`, `USING`.
- The client library now uses the value of the environment variable `MYSQL_HOST` as the default host if it's defined.

- `SELECT col_name, SUM(expr)` now returns `NULL` for `col_name` when there are matching rows.
- Fixed problem with comparing binary strings and `BLOB` values with ASCII characters over 127.
- Fixed lock problem: when freeing a read lock on a table with multiple read locks, a thread waiting for a write lock would have been given the lock. This shouldn't affect data integrity, but could possibly make **mysqld** restart if one thread was reading data that another thread modified.
- `LIMIT offset,count` didn't work in `INSERT ... SELECT`.
- Optimized key block caching. This is quicker than the old algorithm when using bigger key caches.

D.6.19. Changes in release 3.21.16

- Added ODBC 2.0 & 3.0 functions `POWER()`, `SPACE()`, `COT()`, `DEGREES()`, `RADIANS()`, `ROUND(2 arg)` and `TRUNCATE()`.
- **Warning: Incompatible change!** `LOCATE()` parameters were swapped according to ODBC standard. Fixed.
- Added function `TIME_TO_SEC()`.
- In some cases, default values were not used for `NOT NULL` fields.
- Timestamp wasn't always updated properly in `UPDATE SET ...` statements.
- Allow empty strings as default values for `BLOB` and `TEXT`, to be compatible with **mysqldump**.

D.6.20. Changes in release 3.21.15

- **Warning: Incompatible change!** `mysqlperl` is now from Msql-Mysql-modules. This means that `connect()` now takes `host`, `database`, `user`, `password` arguments! The old version took `host`, `database`, `password`, `user`.
- Allow `DATE '1997-01-01'`, `TIME '12:10:10'` and `TIMESTAMP '1997-01-01 12:10:10'` formats required by standard SQL. **Warning: Incompatible change!** This has the unfortunate side effect that you no longer can have columns named `DATE`, `TIME` or `TIMESTAMP`. :(Old columns can still be accessed through `tbl_name.col_name`!)
- Changed Makefiles to hopefully work better with BSD systems. Also, `manual.dvi` is now included in the distribution to avoid having stupid **make** programs trying to rebuild it.
- `readline` library upgraded to version 2.1.
- A new sortorder german-1. That is a normal ISO-Latin1 with a german sort order.
- Perl `DBI`/`DBD` is now included in the distribution. `DBI` is now the recommended way to connect to MySQL from Perl.
- New portable benchmark suite with `DBD`, with test results from `mSQL` 2.0.3, MySQL, PostgreSQL 6.2.1 and Solid server 2.2.
- `crash-me` is now included with the benchmarks; this is a Perl program designed to find as many

limits as possible in an SQL server. Tested with `mSQL`, PostgreSQL, Solid and MySQL.

- Fixed bug in range-optimizer that crashed MySQL on some queries.
- Table and column name completion for **mysql** command-line tool, by Zeev Suraski and Andi Gutmans.
- Added new command `REPLACE` that works like `INSERT` but replaces conflicting records with the new record. `REPLACE INTO TABLE ... SELECT ...` works also.
- Added new commands `CREATE DATABASE` *`db_name`* and `DROP DATABASE` *`db_name`*.
- Added `RENAME` option to `ALTER TABLE`: `ALTER TABLE name RENAME TO new_name`.
- **make_binary_distribution** now includes `libgcc.a` in `libmysqlclient.a`. This should make linking work for people who don't have **gcc**.
- Changed `net_write()` to `my_net_write()` because of a name conflict with Sybase.
- New function `DAYOFWEEK()` compatible with ODBC.
- Stack checking and **bison** memory overrun checking to make MySQL safer with weird queries.

D.6.21. Changes in release 3.21.14b

- Fixed a couple of small **configure** problems on some platforms.

D.6.22. Changes in release 3.21.14a

- Ported to SCO Openserver 5.0.4 with FSU Pthreads.
- HP-UX 10.20 should work.
- Added new function `DATE_FORMAT()`.
- Added `NOT IN`.
- Added automatic removal of 'ODBC function conversions': `{fn now() }`
- Handle ODBC 2.50.3 option flags.
- Fixed comparison of `DATE` and `TIME` values with `NULL`.
- Changed language name from germany to german to be consistent with the other language names.
- Fixed sorting problem on functions returning a `FLOAT`. Previously, the values were converted to `INT` values before sorting.
- Fixed slow sorting when sorting on key field when using *`key_col=constant`*.
- Sorting on calculated `DOUBLE` values sorted on integer results instead.
- **mysql** no longer requires a database argument.

- Changed the place where `HAVING` should be. According to the SQL standards, it should be after `GROUP BY` but before `ORDER BY`. MySQL 3.20 incorrectly had it last.
- Added Sybase command `USE database` to start using another database.
- Added automatic adjusting of number of connections and table cache size if the maximum number of files that can be opened is less than needed. This should fix that **mysqld** doesn't crash even if you haven't done a **ulimit -n 256** before starting **mysqld**.
- Added lots of limit checks to make it safer when running with too little memory or when doing weird queries.

D.6.23. Changes in release 3.21.13

- Added retry of interrupted reads and clearing of `errno`. This makes Linux systems much safer!
- Fixed locking bug when using many aliases on the same table in the same `SELECT`.
- Fixed bug with `LIKE` on number key.
- New error message so you can check whether the connection was lost while the command was running or whether the connection was down from the start.
- Added `--table` option to **mysql** to print in table format. Moved time and row information after query result. Added automatic reconnect of lost connections.
- Added `!=` as a synonym for `<>`.
- Added function `VERSION()` to make easier logs.
- New multi-user test `tests/fork_test.pl` to put some strain on the thread library.

D.6.24. Changes in release 3.21.12

- Fixed `ftruncate()` call in MIT-pthreads. This made **isamchk** destroy the `.ISM` files on (Free)BSD 2.x systems.
- Fixed broken `__P_` patch in MIT-pthreads.
- Many memory overrun checks. All string functions now return `NULL` if the returned string should be longer than `max_allowed_packet` bytes.
- Changed the name of the `INTERVAL` type to `ENUM`, because `INTERVAL` is used in standard SQL.
- In some cases, doing a `JOIN` + `GROUP` + `INTO OUTFILE`, the result wasn't grouped.
- `LIKE` with `'_'` as last character didn't work. Fixed.
- Added extended standard SQL `TRIM()` function.
- Added `CURTIME()`.
- Added `ENCRYPT()` function by Zeev Suraski.

- Fixed better `FOREIGN KEY` syntax skipping. New reserved words: `MATCH`, `FULL`, `PARTIAL`.
- **mysqld** now allows IP number and hostname for the `--bind-address` option.
- Added `SET CHARACTER SET cp1251_koi8` to enable conversions of data to and from the `cp1251_koi8` character set.
- Lots of changes for Windows 95 port. In theory, this version should now be easily portable to Windows 95.
- Changed the `CREATE COLUMN` syntax of `NOT NULL` columns to be after the `DEFAULT` value, as specified in the SQL standard. This makes **mysqldump** with `NOT NULL` and default values incompatible with MySQL 3.20.
- Added many function name aliases so the functions can be used with ODBC or standard SQL syntax.
- Fixed syntax of `ALTER TABLE` *`tbl_name`* `ALTER COLUMN` *`col_name`* `SET DEFAULT NULL`.
- Added `CHAR` and `BIT` as synonyms for `CHAR(1)`.
- Fixed core dump when updating as a user who has only `SELECT` privilege.
- `INSERT ... SELECT ... GROUP BY` didn't work in some cases. An `Invalid use of group function` error occurred.
- When using `LIMIT`, `SELECT` now always uses keys instead of record scan. This gives better performance on `SELECT` and a `WHERE` that matches many rows.
- Added Russian error messages.

D.6.25. Changes in release 3.21.11

- Configure changes.
- MySQL now works with the new thread library on BSD/OS 3.0.
- Added new group functions `BIT_OR()` and `BIT_AND()`.
- Added compatibility functions `CHECK` and `REFERENCES`. `CHECK` is now a reserved word.
- Added `ALL` option to `GRANT` for better compatibility. (`GRANT` is still a dummy function.)
- Added partly translated Dutch error messages.
- Fixed bug in `ORDER BY` and `GROUP BY` with `NULL` columns.
- Added function `LAST_INSERT_ID()` SQL function to retrieve last `AUTO_INCREMENT` value. This is intended for clients to ODBC that can't use the `mysql_insert_id()` API function, but can be used by any client.
- Added `--flush-logs` option to **mysqladmin**.
- Added command `STATUS` to **mysql**.
- Fixed problem with `ORDER BY`/`GROUP BY` because of bug in **gcc**.

- Fixed problem with `INSERT ... SELECT ... GROUP BY`.

D.6.26. Changes in release 3.21.10

- New program **mysqlaccess**.
- `CREATE` now supports all ODBC types and the `mSQL TEXT` type. All ODBC 2.5 functions are also supported (added `REPEAT`). This provides better portability.
- Added text types `TINYTEXT`, `TEXT`, `MEDIUMTEXT` and `LONGTEXT`. These are actually `BLOB` types, but all searching is done in case-insensitive fashion.
- All old `BLOB` fields are now `TEXT` fields. This only changes that all searching on strings is done in case-sensitive fashion. You must do an `ALTER TABLE` and change the data type to `BLOB` if you want to have tests done in case-sensitive fashion.
- Fixed some **configure** issues.
- Made the locking code a bit safer. Fixed very unlikely deadlock situation.
- Fixed a couple of bugs in the range optimizer. Now the new range benchmark `test-select` works.

D.6.27. Changes in release 3.21.9

- Added `--enable-unix-socket=`*`pathname`* option to **configure**.
- Fixed a couple of portability problems with include files.
- Fixed bug in range calculation that could return empty set when searching on multiple key with only one entry (very rare).
- Most things ported to FSU Pthreads, which should allow MySQL to run on SCO. See Section 2.12.5.8, “SCO Notes”.

D.6.28. Changes in release 3.21.8

- Works now in Solaris 2.6.
- Added handling of calculation of `SUM()` functions. For example, you can now use `SUM(column)/COUNT(column)`.
- Added handling of trigometric functions: `PI()`, `ACOS()`, `ASIN()`, `ATAN()`, `COS()`, `SIN()` and `TAN()`.
- New languages: Norwegian, Norwegian-ny and Portuguese.
- Fixed parameter bug in `net_print()` in `procedure.cc`.
- Fixed a couple of memory leaks.
- Now allow also the old `SELECT ... INTO OUTFILE` syntax.

- Fixed bug with `GROUP BY` and `SELECT` on key with many values.
- `mysql_fetch_lengths()` sometimes returned incorrect lengths when you used `mysql_use_result()`. This affected at least some cases of **mysqldump --quick**.
- Fixed bug in optimization of `WHERE const op field`.
- Fixed problem when sorting on `NULL` fields.
- Fixed a couple of 64-bit (Alpha) problems.
- Added `--pid-file=#` option to **mysqld**.
- Added date formatting to `FROM_UNIXTIME()`, originally by Zeev Suraski.
- Fixed bug in `BETWEEN` in range optimizer (did only test = of the first argument).
- Added machine-dependent files for MIT-pthreads i386-SCO. There is probably more to do to get this to work on SCO 3.5.

D.6.29. Changes in release 3.21.7

- Changed `Makefile.am` to take advantage of Automake 1.2.
- Added the beginnings of a benchmark suite.
- Added more secure password handling.
- Added new client function `mysql_errno()`, to get the error number of the error message. This makes error checking in the client much easier. This makes the new server incompatible with the 3.20.x server when running without `--old-protocol`. The client code is backward-compatible. More information can be found in the `README` file!
- Fixed some problems when using very long, illegal names.

D.6.30. Changes in release 3.21.6

- Fixed more portability issues (incorrect `sigwait` and `sigset` defines).
- **configure** should now be able to detect the last argument to `accept()`.

D.6.31. Changes in release 3.21.5

- Should now work with FreeBSD 3.0 if used with `FreeBSD-3.0-libc_r-1.0.diff`, which can be found at http://dev.mysql.com/downloads/os-freebsd.html.
- Added new `-O tmp_table_size=#` option to **mysqld**.
- New function `FROM_UNIXTIME(timestamp)` which returns a date string in `'YYYY-MM-DD HH:MM:SS'` format.
- New function `SEC_TO_TIME(seconds)` which returns a string in `'HH:MM:SS'` format.

- New function `SUBSTRING_INDEX()`, originally by Zeev Suraski.

D.6.32. Changes in release 3.21.4

- Should now configure and compile on OSF/1 4.0 with the DEC compiler.
- Configuration and compilation on BSD/OS 3.0 works, but due to some bugs in BSD/OS 3.0, **mysqld** doesn't work on it yet.
- Configuration and compilation on FreeBSD 3.0 works, but I couldn't get `pthread_create` to work.

D.6.33. Changes in release 3.21.3

- Added reverse check lookup of hostnames to get better security.
- Fixed some possible buffer overflows if filenames that are too long are used.
- **mysqld** doesn't accept hostnames that start with digits followed by a `'.'`, because the hostname may look like an IP number.
- Added `--skip-networking` option to **mysqld**, to allow only socket connections. (This does not work with MIT-pthreads!)
- Added check of too long table names for alias.
- Added check whether database name is okay.
- Added check whether too long table names.
- Removed incorrect `free()` that killed the server on `CREATE DATABASE` or `DROP DATABASE`.
- Changed some **mysqld** `-O` options to better names.
- Added `-O join_cache_size=#` option to **mysqld**.
- Added `-O max_join_size=#` option to **mysqld**, to be able to set a limit how big queries (in this case big = slow) one should be able to handle without specifying `SET SQL_BIG_SELECTS=1`. A # = is about 10 examined records. The default is ``unlimited."
- When comparing a `TIME`, `DATE`, `DATETIME` or `TIMESTAMP` column to a constant, the constant is converted to a time value before performing the comparison. This makes it easier to get ODBC (particularly Access97) to work with these types. It should also make dates easier to use and the comparisons should be quicker than before.
- Applied patch from Jochen Wiedmann that allows `query()` in `mysqlperl` to take a query with `\0` in it.
- Storing a timestamp with a two-digit year (`YYMMDD`) didn't work.
- Fix that timestamp wasn't automatically updated if set in an `UPDATE` clause.
- Now the automatic timestamp field is the FIRST timestamp field.

- `SELECT * INTO OUTFILE`, which didn't correctly if the outfile existed.
- **mysql** now shows the thread ID when starting or doing a reconnect.
- Changed the default sort buffer size from 2MB to 1MB.

D.6.34. Changes in release 3.21.2

- The range optimizer is coded, but only 85% tested. It can be enabled with `--new`, but it crashes core a lot yet...
- More portable. Should compile on AIX and alpha-digital. At least the `isam` library should be relatively 64-bit clean.
- New **isamchk** which can detect and fix more problems.
- New options for `isamlog`.
- Using new version of Automake.
- Many small portability changes (from the AIX and alpha-digital port) Better checking of pthread(s) library.
- czech error messages by `<snajdr@pvt.net>`.
- Decreased size of some buffers to get fewer problems on systems with little memory. Also added more checks to handle ``out of memory'' problems.
- **mysqladmin**: you can now do **mysqladmin kill 5,6,7,8** to kill multiple threads.
- When the maximum connection limit is reached, one extra connection by a user with the **process_acl** privilege is granted.
- Added `-O backlog=#` option to **mysqld**.
- Increased maximum packet size from 512KB to 1024KB for client.
- Almost all of the function code is now tested in the internal test suite.
- `ALTER TABLE` now returns warnings from field conversions.
- Port changed to 3306 (got it reserved from ISI).
- Added a fix for Visual FoxBase so that any schema name from a table specification is automatically removed.
- New function `ASCII()`.
- Removed function `BETWEEN(a,b,c)`. Use the standard SQL syntax instead: `expr BETWEEN` *`expr`* `AND` *`expr`*.
- MySQL no longer has to use an extra temporary table when sorting on functions or `SUM()` functions.
- Fixed bug that you couldn't use *`tbl_name.field_name`* in `UPDATE`.
- Fixed `SELECT DISTINCT` when using 'hidden group'. For example:

```
mysql> SELECT DISTINCT MOD(some_field,10) FROM test
    ->        GROUP BY some_field;
```

Note: `some_field` is normally in the `SELECT` part. Standard SQL should require it.

D.6.35. Changes in release 3.21.0

- New reserved words used: `INTERVAL`, `EXPLAIN`, `READ`, `WRITE`, `BINARY`.
- Added ODBC function `CHAR(num,...)`.
- New operator `IN`. This uses a binary search to find a match.
- New command `LOCK TABLES` *`tbl_name`* `[AS` *`alias`*`] {READ|WRITE} ...`
- Added `--log-update` option to **mysqld**, to get a log suitable for incremental updates.
- New command `EXPLAIN SELECT ...` to get information about how the optimizer does the join.
- For easier client code, the client should no longer use `FIELD_TYPE_TINY_BLOB`, `FIELD_TYPE_MEDIUM_BLOB`, `FIELD_TYPE_LONG_BLOB` or `FIELD_TYPE_VAR_STRING` (as previously returned by `mysql_list_fields`). You should instead use only `FIELD_TYPE_BLOB` or `FIELD_TYPE_STRING`. If you want exact types, you should use the command `SHOW FIELDS`.
- Added varbinary syntax: `0x######` which can be used as a string (default) or a number.
- `FIELD_TYPE_CHAR` is renamed to `FIELD_TYPE_TINY`.
- Changed all fields to C++ classes.
- Removed FORM struct.
- Fields with `DEFAULT` values no longer need to be `NOT NULL`.
- New field types:
  - `ENUM`

    A string which can take only a couple of defined values. The value is stored as a 1-3 byte number that is mapped automatically to a string. This is sorted according to string positions!
  - `SET`

    A string which may have one or many string values separated by ','. The string is stored as a 1-, 2-, 3-, 4- or 8-byte number where each bit stands for a specific set member. This is sorted according to the unsigned value of the stored packed number.
- Now all function calculation is done with `double` or `long long`. This provides the full 64-bit range with bit functions and fixes some conversions that previously could result in precision losses. One should avoid using `unsigned long long` columns with full 64-bit range (numbers bigger than 9223372036854775807) because calculations are done with `signed long long`.
- `ORDER BY` now puts `NULL` field values first. `GROUP BY` also works with `NULL` values.
- Full `WHERE` with expressions.

- New range optimizer that can resolve ranges when some keypart prefix is constant. Example:

```
mysql> SELECT * FROM tbl_name
    ->          WHERE key_part_1='customer'
    ->          AND key_part_2>=10 AND key_part_2<=10;
```

D.7. Changes in release 3.20.x

MySQL 3.20 is quite old now, and should be avoided if possible. This information is kept here for historical purposes only.

Changes from 3.20.18 to 3.20.32b are not documented here because the 3.21 release branched here. And the relevant changes are also documented as changes to the 3.21 version.

D.7.1. Changes in release 3.20.18

- Added `-p#` (remove # directories from path) to `isamlog`. All files are written with a relative path from the database directory Now **mysqld** shouldn't crash on shutdown when using the `--log-isam` option.
- New `mysqlperl` version. It is now compatible with `msqlperl-0.63`.
- New `DBD` module available.
- Added group function `STD()` (standard deviation).
- The **mysqld** server is now compiled by default without debugging information. This makes the daemon smaller and faster.
- Now one usually only has to specify the `--basedir` option to **mysqld**. All other paths are relative in a normal installation.
- `BLOB` columns sometimes contained garbage when used with a `SELECT` on more than one table and `ORDER BY`.
- Fixed that calculations that are not in `GROUP BY` work as expected (standard SQL extension). Example:

```
mysql> SELECT id,id+1 FROM table GROUP BY id;
```

- The test of using `MYSQL_PWD` was reversed. Now `MYSQL_PWD` is enabled as default in the default release.
- Fixed conversion bug which caused **mysqld** to core dump with Arithmetic error on SPARC-386.
- Added `--unbuffered` option to **mysql**, for new **mysqlaccess**.
- When using overlapping (unnecessary) keys and join over many tables, the optimizer could get confused and return 0 records.

D.7.2. Changes in release 3.20.17

- You can now use `BLOB` columns and the functions `IS NULL` and `IS NOT NULL` in the `WHERE` clause.
- All communication packets and row buffers are now allocated dynamically on demand. The default value of `max_allowed_packet` is now 64KB for the server and 512KB for the client. This is mainly used to catch incorrect packets that could trash all memory. The server limit may be changed when it is started.
- Changed stack usage to use less memory.
- Changed **safe_mysqld** to check for running daemon.
- The `ELT()` function is renamed to `FIELD()`. The new `ELT()` function returns a value based on an index: `FIELD()` is the inverse of `ELT()` Example: `ELT(2,"A","B","C")` returns `"B"`. `FIELD("B","A","B","C")` returns `2`.
- `COUNT(field)`, where `field` could have a `NULL` value, now works.
- A couple of bugs fixed in `SELECT ... GROUP BY`.
- Fixed memory overrun bug in `WHERE` with many unoptimizable brace levels.
- Fixed some small bugs in the grant code.
- If hostname isn't found by `get_hostname`, only the IP is checked. Previously, you got `Access denied`.
- Inserts of timestamps with values didn't always work.
- `INSERT INTO ... SELECT ... WHERE` could give the error `Duplicated field`.
- Added some tests to **safe_mysqld** to make it ``safer.''
- `LIKE` was case-sensitive in some places and case-insensitive in others. Now `LIKE` is always case-insensitive.
- `mysql.cc`: Allow `'#'` anywhere on the line.
- New command `SET SQL_SELECT_LIMIT=#`. See the FAQ for more details.
- New version of the **mysqlaccess** script.
- Change `FROM_DAYS()` and `WEEKDAY()` to also take a full `TIMESTAMP` or `DATETIME` as argument. Before they only took a number of type `YYYYMMDD` or `YYMMDD`.
- Added new function `UNIX_TIMESTAMP(timestamp_column)`.

D.7.3. Changes in release 3.20.16

- More changes in MIT-pthreads to get them safer. Fixed also some link bugs at least in SunOS.
- Changed **mysqld** to work around a bug in MIT-pthreads. This makes multiple small `SELECT` operations 20 times faster. Now `lock_test.pl` should work.
- Added `mysql_FetchHash(handle)` to `mysqlperl`.
- The **mysqlbug** script is now distributed built to allow for reporting bugs that appear during the build

with it.

- Changed `libmysql.c` to prefer `getpwuid()` instead of `cuserid()`.
- Fixed bug in `SELECT` optimizer when using many tables with the same column used as key to different tables.
- Added new `latin2` and Russian `KOI8` character tables.
- Added support for a dummy `GRANT` command to satisfy Powerbuilder.

D.7.4. Changes in release 3.20.15

- Fixed fatal bug `packets out of order` when using MIT-pthreads.
- Removed possible loop when a thread waits for command from client and `fcntl()` fails. Thanks to Mike Bretz for finding this bug.
- Changed alarm loop in `mysqld.cc` because shutdown didn't always succeed in Linux.
- Removed use of `termbits` from `mysql.cc`. This conflicted with `glibc` 2.0.
- Fixed some syntax errors for at least BSD and Linux.
- Fixed bug when doing a `SELECT` as superuser without a database.
- Fixed bug when doing `SELECT` with group calculation to outfile.

D.7.5. Changes in release 3.20.14

- If one gives `-p` or `--password` option to **mysql** without an argument, the user is solicited for the password from the tty.
- Added default password from `MYSQL_PWD` (by Elmar Haneke).
- Added command `kill` to **mysqladmin** to kill a specific MySQL thread.
- Sometimes when doing a reconnect on a down connection this succeeded first on second try.
- Fixed adding an `AUTO_INCREMENT` key with `ALTER_TABLE`.
- `AVG()` gave too small value on some `SELECT` statements with `GROUP BY` and `ORDER BY`.
- Added new `DATETIME` type (by Giovanni Maruzzelli `<maruzz@matrice.it>`).
- Fixed that defining `DONT_USE_DEFAULT_FIELDS` works.
- Changed to use a thread to handle alarms instead of signals on Solaris to avoid race conditions.
- Fixed default length of signed numbers. (George Harvey `<georgeh@pinacl.co.uk>`.)
- Allow anything for `CREATE INDEX`.
- Add prezeros when packing numbers to `DATE`, `TIME` and `TIMESTAMP`.

- Fixed a bug in OR of multiple tables (gave empty set).
- Added many patches to MIT-pthreads. This fixes at least one lookup bug.

D.7.6. Changes in release 3.20.13

- Added standard SQL DATE and TIME types.
- Fixed bug in SELECT with AND-OR levels.
- Added support for Slovenian characters. The Contrib directory contains source and instructions for adding other character sets.
- Fixed bug with LIMIT and ORDER BY.
- Allow ORDER BY and GROUP BY on items that aren't in the SELECT list. (Thanks to Wim Bonis <bonis@kiss.de>, for pointing this out.)
- Allow setting of timestamp values in INSERT.
- Fixed bug with SELECT ... WHERE ... = NULL.
- Added changes for glibc 2.0. To get glibc to work, you should add the gibc-2.0-sigwait-patch before compiling glibc.
- Fixed bug in ALTER TABLE when changing a NOT NULL field to allow NULL values.
- Added some standard SQL synonyms as field types to CREATE TABLE. CREATE TABLE now allows FLOAT(4) and FLOAT(8) to mean FLOAT and DOUBLE.
- New utility program **mysqlaccess** by <Yves.Carlier@rug.ac.be>. This program shows the access rights for a specific user and the grant rows that determine this grant.
- Added WHERE const op field (by <bonis@kiss.de>).

D.7.7. Changes in release 3.20.11

- When using SELECT ... INTO OUTFILE, all temporary tables are ISAM instead of HEAP to allow big dumps.
- Changed date functions to be string functions. This fixed some ``funny'' side effects when sorting on dates.
- Extended ALTER TABLE for standard SQL compliance.
- Some minor compatibility changes.
- Added --port and --socket options to all utility programs and **mysqld**.
- Fixed MIT-pthreads readdir_r(). Now **mysqladmin create database** and **mysqladmin drop database** should work.
- Changed MIT-pthreads to use our tempnam(). This should fix the ``sort aborted'' bug.

- Added sync of records count in `sql_update`. This fixed slow updates on first connection. (Thanks to Vaclav Bittner for the test.)

D.7.8. Changes in release 3.20.10

- New insert type: `INSERT INTO ... SELECT ...`
- `MEDIUMBLOB` fixed.
- Fixed bug in `ALTER TABLE` and `BLOB` values.
- `SELECT ... INTO OUTFILE` now creates the file in the current database directory.
- `DROP TABLE` now can take a list of tables.
- Oracle synonym `DESCRIBE` (`DESC`).
- Changes to **make_binary_distribution**.
- Added some comments to installation instructions about **configure** C++ link test.
- Added `--without-perl` option to **configure**.
- Lots of small portability changes.

D.7.9. Changes in release 3.20.9

- `ALTER TABLE` didn't copy null bit. As a result, fields that were allowed to have `NULL` values were always `NULL`.
- `CREATE` didn't take numbers as `DEFAULT`.
- Some compatibility changes for SunOS.
- Removed `config.cache` from old distribution.

D.7.10. Changes in release 3.20.8

- Fixed bug with `ALTER TABLE` and multiple-part keys.

D.7.11. Changes in release 3.20.7

- New commands: `ALTER TABLE`, `SELECT ... INTO OUTFILE` and `LOAD DATA INFILE`.
- New function: `NOW()`.
- Added new field `File_priv` to `mysql/user` table.
- New script `add_file_priv` which adds the new field `File_priv` to the `user` table. This script

must be executed if you want to use the new `SELECT ... INTO` and `LOAD DATA INFILE ...` commands with a version of MySQL earlier than 3.20.7.

- Fixed bug in locking code, which made `lock_test.pl` test fail.
- New files `NEW` and `BUGS`.
- Changed `select_test.c` and `insert_test.c` to include `config.h`.
- Added `status` command to **mysqladmin** for short logging.
- Increased maximum number of keys to 16 and maximum number of key parts to 15.
- Use of sub keys. A key may now be a prefix of a string field.
- Added `-k` option to **mysqlshow**, to get key information for a table.
- Added long options to **mysqldump**.

D.7.12. Changes in release 3.20.6

- Portable to more systems because of MIT-pthreads, which is used automatically if **configure** cannot find a `-lpthreads` library.
- Added GNU-style long options to almost all programs. Test with *`program`* `--help`.
- Some shared library support for Linux.
- The FAQ is now in `.texi` format and is available in `.html`, `.txt` and `.ps` formats.
- Added new SQL function `RAND([init])`.
- Changed `sql_lex` to handle `\0` unquoted, but the client can't send the query through the C API, because it takes a str pointer. You must use `mysql_real_query()` to send the query.
- Added API function `mysql_get_client_info()`.
- **mysqld** now uses the `N_MAX_KEY_LENGTH` from `nisam.h` as the maximum allowable key length.
- The following now works:

  ```
  mysql> SELECT filter_nr,filter_nr FROM filter ORDER BY filter_nr;
  ```

  Previously, this resulted in the error: `Column: 'filter_nr' in order clause is ambiguous`.
- **mysql** now outputs `'\0'`, `'\t'`, `'\n'` and `'\\'` when encountering ASCII 0, tab, newline, or `'\'` while writing tab-separated output. This is to allow printing of binary data in a portable format. To get the old behavior, use `-r` (or `--raw`).
- Added german error messages (60 of 80 error messages translated).
- Added new API function `mysql_fetch_lengths(MYSQL_RES *)`, which returns an array of column lengths (of type `uint`).
- Fixed bug with `IS NULL` in `WHERE` clause.

- Changed the optimizer a little to get better results when searching on a key part.
- Added SELECT option STRAIGHT_JOIN to tell the optimizer that it should join tables in the given order.
- Added support for comments starting with '--' in mysql.cc (Postgres syntax).
- You can have SELECT expressions and table columns in a SELECT which are not used in the group part. This makes it efficient to implement lookups. The column that is used should be a constant for each group because the value is calculated only once for the first row that is found for a group.

  ```
  mysql> SELECT id,lookup.text,SUM(*) FROM test,lookup
      ->        WHERE test.id=lookup.id GROUP BY id;
  ```

- Fixed bug in SUM(function) (could cause a core dump).
- Changed AUTO_INCREMENT placement in the SQL query:

  ```
  INSERT INTO table (auto_field) VALUES (0);
  ```

  inserted 0, but it should insert an AUTO_INCREMENT value.
- mysqlshow.c: Added number of records in table. Had to change the client code a little to fix this.
- **mysql** now allows doubled '' or "" within strings for embedded ' or ".
- New math functions: EXP(), LOG(), SQRT(), ROUND(), CEILING().

D.7.13. Changes in release 3.20.3

- The **configure** source now compiles a thread-free client library -lmysqlclient. This is the only library that needs to be linked with client applications. When using the binary releases, you must link with -lmysql -lmysys -ldbug -lmystrings as before.
- New readline library from bash-2.0.
- LOTS of small changes to **configure** and makefiles (and related source).
- It should now be possible to compile in another directory using VPATH. Tested with GNU Make 3.75.
- **safe_mysqld** and **mysql.server** changed to be more compatible between the source and the binary releases.
- LIMIT now takes one or two numeric arguments. If one argument is given, it indicates the maximum number of rows in a result. If two arguments are given, the first argument indicates the offset of the first row to return, the second is the maximum number of rows. With this it's easy to do a poor man's next page/previous page WWW application.
- Changed name of SQL function FIELDS() to ELT(). Changed SQL function INTERVALL() to INTERVAL().
- Made SHOW COLUMNS a synonym for SHOW FIELDS. Added compatibility syntax FRIEND KEY to CREATE TABLE. In MySQL, this creates a non-unique key on the given columns.
- Added CREATE INDEX and DROP INDEX as compatibility functions. In MySQL, CREATE IN-

DEX only checks whether the index exists and issues an error if it doesn't exist. DROP INDEX always succeeds.

- mysqladmin.c: added client version to version information.
- Fixed core dump bug in sql_acl (core on new connection).
- Removed host, user and db tables from database test in the distribution.
- FIELD_TYPE_CHAR can now be signed (-128 to 127) or unsigned (0 to 255) Previously, it was always unsigned.
- Bug fixes in CONCAT() and WEEKDAY().
- Changed a lot of source to get **mysqld** to be compiled with SunPro compiler.
- SQL functions must now have a '(' immediately after the function name (no intervening space). For example, 'USER(' is regarded as beginning a function call, and 'USER (' is regarded as an identifier USER followed by a '(', not as a function call.

D.7.14. Changes in release 3.20.0

- The source distribution is done with **configure** and Automake. It makes porting much easier. The readline library is included in the distribution.
- Separate client compilation: the client code should be very easy to compile on systems which don't have threads.
- The old Perl interface code is automatically compiled and installed. Automatic compiling of DBD follows when the new DBD code is ported.
- Dynamic language support: **mysqld** can now be started with Swedish or English (default) error messages.
- New functions: INSERT(), RTRIM(), LTRIM() and FORMAT().
- **mysqldump** now works correctly for all field types (even AUTO_INCREMENT). The format for SHOW FIELDS FROM *tbl_name* is changed so the Type column contains information suitable for CREATE TABLE. In previous releases, some CREATE TABLE information had to be patched when re-creating tables.
- Some parser bugs from 3.19.5 (BLOB and TIMESTAMP) are corrected. TIMESTAMP now returns different date information depending on its create length.
- Changed parser to allow a database, table or field name to start with a number or '_'.
- All old C code from Unireg changed to C++ and cleaned up. This makes the daemon a little smaller and easier to understand.
- A lot of small bugfixes done.
- New INSTALL files (not final version) and some information regarding porting.

D.8. Changes in release 3.19.x

MySQL 3.19 is quite old now, and should be avoided if possible. This information is kept here for historical purposes only.

D.8.1. Changes in release 3.19.5

- Some new functions, some more optimization on joins.
- Should now compile clean on Linux (2.0.x).
- Added functions `DATABASE()`, `USER()`, `POW()`, `LOG10()` (needed for ODBC).
- In a `WHERE` with an `ORDER BY` on fields from only one table, the table is now preferred as first table in a multi-join.
- `HAVING` and `IS NULL` or `IS NOT NULL` now works.
- A group on one column and a sort on a group function (`SUM()`, `AVG()`...) didn't work together. Fixed.
- **mysqldump**: Didn't send password to server.

D.8.2. Changes in release 3.19.4

- Fixed horrible locking bug when inserting in one thread and reading in another thread.
- Fixed one-off decimal bug. 1.00 was output as 1.0.
- Added attribute `'Locked'` to process list as information if a query is locked by another query.
- Fixed full magic timestamp. Timestamp length may now be 14, 12, 10, 8, 6, 4 or 2 bytes.
- Sort on some numeric functions could sort incorrectly on last number.
- `IF(arg,syntax_error,syntax_error)` crashed.
- Added functions `CEILING()`, `ROUND()`, `EXP()`, `LOG()` and `SQRT()`.
- Enhanced `BETWEEN` to handle strings.

D.8.3. Changes in release 3.19.3

- Fixed `SELECT` with grouping on `BLOB` columns not to return incorrect `BLOB` info. Grouping, sorting and distinct on `BLOB` columns does not yet work as expected (probably it groups/sorts by the first 7 characters in the `BLOB`). Grouping on formulas with a fixed string size (use `MID()` on a `BLOB`) should work.
- When doing a full join (no direct keys) on multiple tables with `BLOB` fields, the `BLOB` was garbage on output.
- Fixed `DISTINCT` with calculated columns.

D.9. InnoDB Change History

Starting from 4.0.22 and 4.1.5, all InnoDB changes are included in the MySQL Change History, and this manual section is no longer separately maintained.

D.9.1. MySQL/InnoDB-4.0.21, September 10, 2004

Functionality added or changed:

- Renamed the `innodb.status.<pid>` files (created in the data directory) to `innodb_status.<pid>`. This avoids problems on filesystems that do not allow multiple periods in filenames.
- Added `innodb_status_file` system variable to **mysqld** to control whether output from `SHOW INNODB STATUS` is written to a `innodb_status.<pid>` file in the data directory. By default, the file is not created. To create it, start **mysqld** with the `--innodb_status_file=1` option.
- Changes for NetWare to exit InnoDB gracefully on NetWare even in a case of an assertion failure, instead of intentionally crashing the `` `mysqld' `` server process.

Bugs fixed:

- Fixed a bug in `ON DELETE CASCADE` and `ON UPDATE CASCADE` foreign key constraints: long chains of cascaded operations would cause a stack overflow and crash the server. Cascaded operations are now limited to 15 levels. (Bug #4446 [http://bugs.mysql.com/4446])
- Fixed a possible bug in `LOCK TABLES` introduced in MySQL/InnoDB-4.0.19: The count of tables explicitly locked by a transaction was incremented only after the locks were granted, but decremented when the lock structures were destroyed.
- Fixed a bug in `UNLOCK TABLES` in `AUTOCOMMIT=0` mode, introduced in MySQL/InnoDB-4.0.19: The memory allocated for some locks acquired by the transaction could be deallocated before those locks were released. The bug can lead to crashes and memory corruption of the buffer pool when the transaction acquires a large number of locks (table locks or row-level locks).
- Increment the InnoDB watchdog timeout during `CHECK TABLE`. A long-running `CHECK TABLE` would cause InnoDB to complain about a 'long semaphore wait', and crash the server, if a query had to wait more than 600 seconds behind that `CHECK TABLE` operation. (Bug #2694 [http://bugs.mysql.com/2694])
- If you configure `innodb_additional_mem_pool_size` so small that InnoDB memory allocation spills over from it, then every 4 billionth spill may cause memory corruption. A symptom is a printout like below in the `.err` log. The workaround is to make `innodb_additional_mem_pool_size` big enough to hold all memory allocation. Use `SHOW INNODB STATUS` to determine that there is plenty of free space available in the additional mem pool, and the total allocated memory stays rather constant.

  ```
  InnoDB: Error: Mem area size is 0. Possibly a memory overrun of the
  InnoDB: previous allocated area!
  InnoDB: Apparent memory corruption: mem dump  len 500; hex
  ```

- The special meaning of the table names `innodb_monitor`, `innodb_lock_monitor`, `innodb_tablespace_monitor`, `innodb_table_monitor`, and `innodb_validate` in `CREATE TABLE` and `DROP TABLE` statements was accidentally removed in MySQL/In-

noDB-4.0.19. The diagnostic functions attached to these special table names (see Section 15.12.1, "SHOW INNODB STATUS and the InnoDB Monitors") are accessible again in MySQL/InnoDB-4.0.21.

- When the private SQL parser of InnoDB was modified in MySQL/InnoDB-4.0.19 in order to allow the use of the apostrophe ('') in table and column names, the fix relied on a previously unused function mem_realloc(), whose implementation was incorrect. As a result, InnoDB can incorrectly parse column and table names as the empty string. The InnoDB realloc() implementation has been corrected in MySQL/InnoDB-4.0.21.

- Fixed a glitch introduced in 4.0.18 and 4.1.2: in SHOW TABLE STATUS InnoDB systematically overestimated the row count by 1 if the table fit on a single 16 kB data page.

- InnoDB created temporary files with the C library function tmpfile(). On Windows, the files would be created in the root directory of the current file system. To correct this behavior, the invocations of tmpfile() were replaced with code that uses the function create_temp_file() in the MySQL portability layer. (Bug #3998 [http://bugs.mysql.com/3998])

- If ALTER TABLE ... DROP FOREIGN KEY ... fails because of a wrong constraint name, return a table handler error number 150 instead of 152.

- If there was little file I/O in InnoDB, but the insert buffer was used, it could happen that 'Pending normal aio reads' was bigger than 0, but the I/O handler thread did not get waken up in 600 seconds. This resulted in a hang, and crashing of InnoDB.

- If we RENAMEd a table, InnoDB forgot to load the FOREIGN KEY constraints that reference the new table name, and forgot to check that they are compatible with the table.

D.9.2. MySQL/InnoDB-4.1.4, August 31, 2004

Functionality added or changed:

- **Important:** Made internal representation of TIMESTAMP values in InnoDB in 4.1 to be the same as in 4.0. This difference resulted in incorrect datetime values in TIMESTAMP columns in InnoDB tables after an upgrade from 4.0 to 4.1. (Bug #4492 [http://bugs.mysql.com/4492]) **Warning: extra steps during upgrade required!** This means that if you are upgrading from 4.1.x, where x <= 3, to 4.1.4 you should use mysqldump for saving and then restoring your InnoDB tables with TIMESTAMP columns. No conversion is needed if you upgrade from 3.23 or 4.0 to 4.1.4 or later.

- Added a new startup option innodb_locks_unsafe_for_binlog. This option forces InnoDB not to use next-key locking in searches and index scans.

- Added innodb_status_file system variable to **mysqld** to control whether output from SHOW INNODB STATUS is written to a innodb_status.<*pid*> file in the data directory. By default, the file is not created. To create it, start **mysqld** with the --innodb_status_file=1 option.

- Changes for NetWare to exit InnoDB gracefully on NetWare even in a case of an assertion failure, instead of intentionally crashing the **mysqld** server process.

- 'Gap' type row locks without the LOCK_INSERT_INTENTION flag do not need to wait for anything. This is because different users can have conflicting lock types on gaps. This change reduces unnecessary deadlocks.

Bugs fixed:

- Fixed a bug in `ON DELETE CASCADE` and `ON UPDATE CASCADE` foreign key constraints: long chains of cascaded operations would cause a stack overflow and crash the server. Cascaded operations are now limited to 15 levels. (Bug #4446 [http://bugs.mysql.com/4446])
- Increment the InnoDB watchdog timeout during `CHECK TABLE`. (Bug #2694 [http://bugs.mysql.com/2694])
- If you configure `innodb_additional_mem_pool_size` so small that InnoDB memory allocation spills over from it, then every 4 billionth spill may cause memory corruption. A symptom is a printout like below in the `.err` log.

  ```
  InnoDB: Error: Mem area size is 0. Possibly a memory overrun of the
  InnoDB: previous allocated area!
  InnoDB: Apparent memory corruption: mem dump  len 500; hex
  ```

- Fixed a glitch introduced in 4.0.18 and 4.1.2: in `SHOW TABLE STATUS` InnoDB systematically overestimated the row count by 1 if the table fit on a single 16 kB data page.
- InnoDB created temporary files with the C library function `tmpfile()`. On Windows, the files would be created in the root directory of the current file system. To correct this behavior, the invocations of `tmpfile()` were replaced with code that uses the function `create_temp_file()` in the MySQL portability layer. (Bug #3998 [http://bugs.mysql.com/3998])
- If we `RENAME`d a table, InnoDB forgot to load the foreign key constraints that reference the new table name, and forgot to check that they are compatible with the table.
- If there was little file I/O in InnoDB, but the insert buffer was used, it could happen that 'Pending normal aio reads' was bigger than 0, but the I/O handler thread did not get waken up in 600 seconds. This resulted in a hang, and an intentional crashing of **mysqld**.

D.9.3. MySQL/InnoDB-4.1.3, June 28, 2004

Functionality added or changed:

- **Important:** Starting from MySQL 4.1.3, `InnoDB` uses the same character set comparison functions as MySQL for non-`latin1_swedish_ci` character strings that are not `BINARY`. This changes the sorting order of space and characters < ASCII(32) in those character sets. For `latin1_swedish_ci` character strings and `BINARY` strings, `InnoDB` uses its own pad-spaces-at-end comparison method, which stays unchanged. If you have an `InnoDB` table created with MySQL 4.1.2 or earlier, with an index on a non-`latin1` character set (in the case of 4.1.0 and 4.1.1 with any character set) `CHAR`/`VARCHAR`/or `TEXT` column that is not `BINARY` but may contain characters < ASCII(32), then you should do `ALTER TABLE` or `OPTIMIZE` table on it to **regenerate the index, after upgrading to MySQL 4.1.3 or later**.
- `OPTIMIZE TABLE` for `InnoDB` tables is now mapped to `ALTER TABLE` rather than to `ANALYZE TABLE`.
- Added an interface for storing the binlog offset in the InnoDB log and flushing the log.

Bugs fixed:

- The **critical bug in 4.1.2** (crash recovery skipping all `.ibd` files if you specify `innodb_file_per_table` on Unix) has been fixed. The bug was a combination of two bugs. Crash recovery ignored the files, because the attempt to lock them in the wrong mode failed. From

now on, locks are only obtained for regular files opened in read/write mode, and crash recovery stops if an `.ibd` file for a table exists in a database directory but is unaccessible.

- Do not remember the original `select_lock_type` inside `LOCK TABLES`. (Bug #4047 [http://bugs.mysql.com/4047])

- The special meaning of the table names `innodb_monitor`, `innodb_lock_monitor`, `innodb_tablespace_monitor`, `innodb_table_monitor`, and `innodb_validate` in `CREATE TABLE` and `DROP TABLE` statements was accidentally removed in MySQL/InnoDB-4.1.2. The diagnostic functions attached to these special table names (see Section 15.12.1, "`SHOW INNODB STATUS` and the `InnoDB` Monitors") are accessible again in MySQL/InnoDB-4.1.3.

- When the private SQL parser of InnoDB was modified in MySQL/InnoDB-4.0.19 in order to allow the use of the apostrophe (`''`) in table and column names, the fix relied on a previously unused function `mem_realloc()`, whose implementation was incorrect. As a result, InnoDB can incorrectly parse column and table names as the empty string. The InnoDB `realloc()` implementation has been corrected in MySQL/InnoDB-4.1.3.

- In a clean-up of MySQL/InnoDB-4.1.2, the code for invalidating the query cache was broken. Now the query cache should be correctly invalidated for tables affected by `ON UPDATE CASCADE` or `ON DELETE CASCADE` constraints.

- Fixed a bug: in `LIKE 'abc%'`, the `'%'` did not match the empty string if the character set was not `latin1_swedish_ci`. This bug was fixed by changing the sorting order in these character sets. See the above note about data conversion in 4.1.3.

D.9.4. MySQL/InnoDB-4.1.2, May 30, 2004

NOTE: CRITICAL BUG in 4.1.2 if you specify `innodb_file_per_table` in `my.cnf` on Unix. In crash recovery InnoDB skips the crash recovery for all `.ibd` files and those tables become CORRUPT! The symptom is a message `Unable to lock ...ibd with lock 1, error: 9: fcntl: Bad file descriptor` in the `.err` log in crash recovery.

Functionality added or changed:

- Support multiple character sets. Note that tables created in other collations than `latin1_swedish_ci` cannot be accessed in MySQL/InnoDB 4.0.

- Automatically create a suitable index on a `FOREIGN KEY`, if the user does not create one. Removes most of the cases of `Error 1005 (errno 150)` in table creation.

- Do not assert in `log0log.c`, line 856 if `ib_logfiles` are too small for `innodb_thread_concurrency`. Instead, print instructions how to adjust `my.cnf` and call `exit(1)`.

- If MySQL tries to `SELECT` from an InnoDB table without setting any table locks, print a descriptive error message and assert; some subquery bugs were of this type.

- Allow a key part length in InnoDB to be up to 3,500 bytes; this is needed so that you can create an index on a column with 255 UTF-8 characters.

- All new features from InnoDB-4.0.17, InnoDB-4.0.18, InnoDB-4.0.19 and InnoDB-4.0.20.

Bugs fixed:

- If you configure `innodb_additional_mem_pool_size` so small that InnoDB memory allocation spills over from it, then every 4 billionth spill may cause memory corruption. A symptom is a printout like below in the `.err` log.

  ```
  InnoDB: Error: Mem area size is 0. Possibly a memory overrun of the
  InnoDB: previous allocated area!
  InnoDB: Apparent memory corruption: mem dump  len 500; hex
  ```

- Improved portability to 64-bit platforms, especially Win64.
- Fixed an assertion failure when a purge of a table was not possible because of missing `.ibd` file.
- Fixed a bug: do not retrieve all columns in a table if we only need the 'ref' of the row (usually, the `PRIMARY KEY`) to calculate an `ORDER BY`. (Bug #1942 [http://bugs.mysql.com/1942])
- On Unix-like systems, obtain an exclusive advisory lock on InnoDB files, to prevent corruption when multiple instances of MySQL are running on the same set of data files. The Windows version of InnoDB currently takes a mandatory lock on the files. (Bug #3608 [http://bugs.mysql.com/3608])
- Added a missing space to the output format of `SHOW INNODB STATUS`; reported by Jocelyn Fournier.
- All bugfixes from InnoDB-4.0.17, InnoDB-4.0.18, InnoDB-4.0.19 and InnoDB-4.0.20.

D.9.5. MySQL/InnoDB-4.0.20, May 18, 2004

Bugs fixed:

- Apostrophe characters now are recognized by the internal `InnoDB` parser and can be used within quoted table and column identifiers in `FOREIGN KEY` clauses.
- Make `LOCK TABLE` aware of `InnoDB` row-level locks and `InnoDB` aware of locks set with `LOCK TABLE`. (Bug #3299 [http://bugs.mysql.com/3299])
- Fixed race conditions in `SHOW INNODB STATUS`. (Bug #3596 [http://bugs.mysql.com/3596])

D.9.6. MySQL/InnoDB-4.0.19, May 4, 2004

Functionality added or changed:

- Better error message when the server has to crash because the buffer pool is exhausted by the lock table or the adaptive hash index.
- Print always the count of pending `pread()` and `pwrite()` calls if there is a long semaphore wait.
- Improve space utilization when rows of 1,500 to 8,000 bytes are inserted in the order of the primary key.
- Remove potential buffer overflow errors by sending diagnostic output to stderr or files instead of stdout or fixed-size memory buffers. As a side effect, the output of `SHOW INNODB STATUS` is written to a file `<datadir>/innodb.status.<pid>` every 15 seconds.

Bugs fixed:

- Fixed a bug: `DROP DATABASE` did not work if `FOREIGN KEY` references were defined within the database. (Bug #3058 [http://bugs.mysql.com/3058])
- Remove unnecessary files, functions and variables. Many of these were needed in the standalone version of InnoDB. Remove debug functions and variables from non-debug build.
- Add diagnostic code to analyze an assertion failure in ha_innodb.cc on line 2020 reported by a user. (Bug #2903 [http://bugs.mysql.com/2903])
- Fixed a bug: in a `FOREIGN KEY`, `ON UPDATE CASCADE` was not triggered if the update changed a string to another value identical in alphabetical ordering, for example, '`abc`' -> '`aBc`'.
- Protect the reading of the latest foreign key error explanation buffer with a mutex; in theory, a race condition could cause `SHOW INNODB STATUS` print garbage characters after the error info.
- Fixed a bug: The row count and key cardinality estimate was grossly too small if each clustered index page only contained one record.
- Parse `CONSTRAINT FOREIGN KEY` correctly. (Bug #3332 [http://bugs.mysql.com/3332])
- Fixed a memory corruption bug on Windows. The bug is present in all InnoDB versions in Windows, but it depends on how the linker places a static array in `srv0srv.c`, whether the bug shows itself. 4 bytes were overwritten with a pointer to a statically allocated string '`get windows aio return value`'.
- Fix a glitch reported by Philippe Lewicki on the general mailing list: do not print a warning to the `.err` log if `read_key` fails with a lock wait timeout error 146.
- Allow quotes to be embedded in strings in the private SQL parser of InnoDB, so that `''` can be used in InnoDB table and column names. Display quotes within identifiers properly.
- Debugging: Allow `UNIV_SYNC_DEBUG` to be disabled while `UNIV_DEBUG` is enabled.
- Debugging: Handle magic numbers in a more consistent way.

D.9.7. MySQL/InnoDB-4.0.18, February 13, 2004

- Do not allow dropping a table referenced by a `FOREIGN KEY` constraint, unless the user does `SET FOREIGN_KEY_CHECKS=0`. The error message here is somewhat misleading ``Cannot delete or update a parent row...", and must be changed in a future version 4.1.x.
- Make `InnoDB` to remember the `CONSTRAINT` name given by a user for a `FOREIGN KEY`.
- Change the print format of `FOREIGN KEY` constraints spanning multiple databases to `` `*db_name*`.`*tbl_name*` ``. But when parsing them, we must also accept `` `*db_name.tbl_name*` ``, because that was the output format in < 4.0.18.
- An optimization in locking: If `AUTOCOMMIT=1`, then we do not need to make a plain `SELECT` set shared locks even on the `SERIALIZABLE` isolation level, because we know that the transaction is read-only. A read-only transaction can always be performed on the `REPEATABLE READ` level, and that does not endanger the serializability.
- Implement an automatic downgrade from >= 4.1.1 -> 4.0.18 if the user has not created tables in `.ibd` files or used other 4.1.x features. **Consult** the manual section on **multiple tablespaces** care-

fully if you want to downgrade!

- Fixed a bug: MySQL should not allow `REPLACE` to internally perform an `UPDATE` if the table is referenced by a `FOREIGN KEY`. The MySQL manual states that `REPLACE` must resolve a duplicate-key error semantically with `DELETE`(s) + `INSERT`, and not by an `UPDATE`. In versions < 4.0.18 and < 4.1.2, MySQL could resolve a duplicate key conflict in `REPLACE` by doing an `UPDATE` on the existing row, and `FOREIGN KEY` checks could behave in a semantically wrong way. (Bug #2418 [http://bugs.mysql.com/2418])

- Fixed a bug: generate `FOREIGN KEY` constraint identifiers locally for each table, in the form *`db_name/tbl_name_ibfk_number`*. If the user gives the constraint name explicitly, then remember it. These changes should ensure that foreign key id's in a slave are the same as in the master, and `DROP FOREIGN KEY` does not break replication. (Bug #2167 [http://bugs.mysql.com/2167])

- Fixed a bug: allow quoting of identifiers in InnoDB's `FOREIGN KEY` definitions with a backtick (`) and a double quote ("). You can now use also spaces in table and column names, if you quote the identifiers. (Bug #1725 [http://bugs.mysql.com/1725], Bug #2424 [http://bugs.mysql.com/2424])

- Fixed a bug: `FOREIGN KEY ... ON UPDATE/DELETE NO ACTION` must check the foreign key constraint, not ignore it. Since we do not have deferred constraints in `InnoDB`, this bugfix makes `InnoDB` to check `NO ACTION` constraints immediately, like it checks `RESTRICT` constraints.

- Fixed a bug: `InnoDB` crashed in `RENAME TABLE` if *`db_name.tbl_name`* is shorter than 5 characters. (Bug #2689 [http://bugs.mysql.com/2689])

- Fixed a bug: in `SHOW TABLE STATUS`, `InnoDB` row count and index cardinality estimates wrapped around at 512 million in 32-bit computers. Note that unless MySQL is compiled with the `BIG_TABLES` option, they still wrap around at 4 billion.

- Fixed a bug: If there was a `UNIQUE` secondary index, and `NULL` values in that unique index, then with the `IS NULL` predicate, `InnoDB` returned only the first matching row, though there can be many. This bug was introduced in 4.0.16. (Bug #2483 [http://bugs.mysql.com/2483])

D.9.8. MySQL/InnoDB-5.0.0, December 24, 2003

- **Important note:** If you upgrade to MySQL 4.1.1 or higher, it is difficult to downgrade back to 4.0 or 4.1.0! That is because, for earlier versions, `InnoDB` is not aware of multiple tablespaces.

- `InnoDB` in 5.0.0 is essentially the same as InnoDB-4.1.1 with the bugfixes of InnoDB-4.0.17 included.

D.9.9. MySQL/InnoDB-4.0.17, December 17, 2003

- Fixed a bug: If you created a column prefix secondary index and updated it so that the last characters in the column prefix were spaces, `InnoDB` would assert in `row0upd.c`, line 713. The same assertion failed if you updated a column in an ordinary secondary index so that the new value was alphabetically equivalent, but had a different length. This could happen, for example, in the UTF8 character set if you updated a letter to its accented or umlaut form.

- Fixed a bug: `InnoDB` could think that a secondary index record was not locked though it had been updated to an alphabetically equivalent value, for example, 'abc' -> 'aBc'.

- Fixed a bug: If you updated a secondary index column to an alphabetically equivalent value, and

rolled back your update, InnoDB failed to restore the field in the secondary index to its original value.

- There are still several outstanding non-critical bugs reported in the MySQL bugs database. Their fixing has been delayed, because resources were allocated to the 4.1.1 release.

D.9.10. MySQL/InnoDB-4.1.1, December 4, 2003

- **Important note:** If you upgrade to MySQL 4.1.1 or higher, you cannot downgrade to a version lower than 4.1.1 any more! That is because, for earlier versions, InnoDB is not aware of multiple tablespaces.
- Multiple tablespaces now available for InnoDB. You can store each InnoDB type table and its indexes into a separate .ibd file into a MySQL database directory, into the same directory where the .frm file is stored.
- The MySQL query cache now works for InnoDB tables also if AUTOCOMMIT=0, or the statements are enclosed inside BEGIN ... COMMIT.
- Reduced InnoDB memory consumption by a few megabytes if one sets the buffer pool size < 8MB.
- You can use raw disk partitions also in Windows.

D.9.11. MySQL/InnoDB-4.0.16, October 22, 2003

- Fixed a bug: in contrary to what was said in the manual, in a locking read InnoDB set two record locks if a unique exact match search condition was used on a multi-column unique key. For a single column unique key it worked right.
- Fixed a bug: If you used the rename trick #sql... -> rsql... to recover a temporary table, InnoDB asserted in row_mysql_lock_data_dictionary().
- There are several outstanding non-critical bugs reported in the MySQL bugs database. Their fixing has been delayed, because resources are allocated to the upcoming 4.1.1 release.

D.9.12. MySQL/InnoDB-3.23.58, September 15, 2003

- Fixed a bug: InnoDB could make the index page directory corrupt in the first B-tree page splits after **mysqld** startup. A symptom would be an assertion failure in page0page.c, in function page_dir_find_slot().
- Fixed a bug: InnoDB could in rare cases return an extraneous row if a rollback, purge, and a SELECT coincided.
- Fixed a possible hang over the btr0sea.c latch if SELECT was used inside LOCK TABLES.
- Fixed a bug: If a single DELETE statement first managed to delete some rows and then failed in a FOREIGN KEY error or a Table is full error, MySQL did not roll back the whole SQL statement as it should.

D.9.13. MySQL/InnoDB-4.0.15, September 10, 2003

- Fixed a bug: If you updated a row so that the 8000 byte maximum length (without `BLOB` and `TEXT`) was exceeded, `InnoDB` simply removed the record from the clustered index. In a similar insert, `InnoDB` would leak reserved file space extents, which would only be freed at the next **mysqld** startup.
- Fixed a bug: If you used big `BLOB` values, and your log files were relatively small, `InnoDB` could in a big `BLOB` operation temporarily write over the log produced after the latest checkpoint. If `InnoDB` would crash at that moment, then the crash recovery would fail, because `InnoDB` would not be able to scan the log even up to the latest checkpoint. Starting from this version, `InnoDB` tries to ensure the latest checkpoint is young enough. If that is not possible, `InnoDB` prints a warning to the `.err` log of MySQL and advises you to make the log files bigger.
- Fixed a bug: setting `innodb_fast_shutdown=0` had no effect.
- Fixed a bug introduced in 4.0.13: If a `CREATE TABLE` ended in a comment, that could cause a memory overrun.
- Fixed a bug: If `InnoDB` printed `Operating system error number .. in a file operation` to the `.err` log in Windows, the error number explanation was wrong. Workaround: look at section 13.2 of http://www.innodb.com/ibman.php about Windows error numbers.
- Fixed a bug: If you created a column prefix `PRIMARY KEY` like in `t(a CHAR(200), PRIMARY KEY (a(10)))` on a fixed-length `CHAR` column, `InnoDB` would crash even in a simple `SELECT`. A `CHECK TABLE` would report the table as corrupt, also in the case where the created key was not `PRIMARY`.

D.9.14. MySQL/InnoDB-4.0.14, July 22, 2003

- `InnoDB` now supports the `SAVEPOINT` and `ROLLBACK TO SAVEPOINT` SQL statements. See http://www.innodb.com/ibman.php#Savepoints for the syntax.
- You can now create column prefix keys like in `CREATE TABLE t (a BLOB, INDEX (a(10)))`.
- You can also use `O_DIRECT` as the `innodb_flush_method` on the latest versions of Linux and FreeBSD. Beware of possible bugs in those operating systems, though.
- Fixed the checksum calculation of data pages. Previously most OS file system corruption went unnoticed. Note that if you downgrade from version >= 4.0.14 to an earlier version < 4.0.14 then in the first startup(s) `InnoDB` prints warnings:

  ```
  InnoDB: Warning: An inconsistent page in the doublewrite buffer
  InnoDB: space id 2552202359 page number 8245, 127'th page in dblwr buf.
  ```

  but that is not dangerous and can be ignored.
- Modified the buffer pool replacement algorithm so that it tries to flush modified pages if there are no replaceable pages in the last 10 % of the LRU list. This can reduce disk I/O if the workload is a mixture of reads and writes.
- The buffer pool checkpoint flush algorithm now tries to flush also close neighbors of the page at the end of the flush list. This can speed up database shutdown, and can also speed up disk writes if `InnoDB` log files are very small compared to the buffer pool size.
- In 4.0.13 we made `SHOW INNODB STATUS` to print detailed info on the latest `UNIQUE KEY` er-

ror, but storing that information could slow down `REPLACE` significantly. We no longer store or print the info.

- Fixed a bug: `SET FOREIGN_KEY_CHECKS=0` was not replicated properly in the MySQL replication. The fix will not be backported to 3.23.
- Fixed a bug: the parameter `innodb_max_dirty_pages_pct` forgot to take into account the free pages in the buffer pool. This could lead to excessive flushing even though there were lots of free pages in the buffer pool. Workaround: `SET GLOBAL innodb_max_dirty_pages_pct = 100`.
- Fixed a bug: If there were big index scans then a file read request could starve and `InnoDB` could assert because of a very long semaphore wait.
- Fixed a bug: If `AUTOCOMMIT=1` then inside `LOCK TABLES` MySQL failed to do the commit after an updating SQL statement if binary logging was not on, and for `SELECT` statements did not commit regardless of binary logging state.
- Fixed a bug: `InnoDB` could make the index page directory corrupt in the first B-tree page splits after a **mysqld** startup. A symptom would be an assertion in page0page.c, in function page_dir_find_slot().
- Fixed a bug: If in a `FOREIGN KEY` with an `UPDATE CASCADE` clause the parent column was of a different internal storage length than the child column, then a cascaded update would make the column length wrong in the child table and corrupt the child table. Because of MySQL's 'silent column specification changes' a fixed-length `CHAR` column can change internally to a `VARCHAR` and cause this error.
- Fixed a bug: If a non-`latin1` character set was used and if in a `FOREIGN KEY` the parent column was of a different internal storage length than the child column, then all inserts to the child table would fail in a foreign key error.
- Fixed a bug: `InnoDB` could complain that it cannot find the clustered index record, or in rare cases return an extraneous row if a rollback, purge, and a `SELECT` coincided.
- Fixed a possible hang over the btr0sea.c latch if `SELECT` was used inside `LOCK TABLES`.
- Fixed a bug: contrary to what the release note of 4.0.13 said, the group commit still did not work if the MySQL binary logging was on.
- Fixed a bug: os_event_wait() did not work properly in Unix, which might have caused starvation in various log operations.
- Fixed a bug: If a single `DELETE` statement first managed to delete some rows and then failed in a `FOREIGN KEY` error or a `Table is full` error, MySQL did not roll back the whole SQL statement as it should, and also wrote the failed statement to the binary log, reporting there a non-zero error_code.
- Fixed a bug: the maximum allowed number of columns in a table is 1000, but `InnoDB` did not check that limit in `CREATE TABLE`, and a subsequent `INSERT` or `SELECT` from that table could cause an assertion.

D.9.15. MySQL/InnoDB-3.23.57, June 20, 2003

- Changed the default value of `innodb_flush_log_at_trx_commit` from 0 to 1. If you have not specified it explicitly in your `my.cnf`, and your application runs much slower with this new re-

lease, it is because the value 1 causes a log flush to disk at each transaction commit.

- Fixed a bug: InnoDB forgot to call pthread_mutex_destroy() when a table was dropped. That could cause memory leakage on FreeBSD and other non-Linux Unixes.
- Fixed a bug: MySQL could erroneously return 'Empty set' if InnoDB estimated an index range size to 0 records though the range was not empty; MySQL also failed to do the next-key locking in the case of an empty index range.
- Fixed a bug: GROUP BY and DISTINCT could treat NULL values inequal.

D.9.16. MySQL/InnoDB-4.0.13, May 20, 2003

- InnoDB now supports ALTER TABLE DROP FOREIGN KEY. You have to use SHOW CREATE TABLE to find the internally generated foreign key ID when you want to drop a foreign key.
- SHOW INNODB STATUS now prints detailed information of the latest detected FOREIGN KEY and UNIQUE KEY errors. If you do not understand why InnoDB gives the error 150 from a CREATE TABLE, you can use this statement to study the reason.
- ANALYZE TABLE now works also for InnoDB type tables. It makes 10 random dives to each of the index trees and updates index cardinality estimates accordingly. Note that because these are only estimates, repeated runs of ANALYZE TABLE may produce different numbers. MySQL uses index cardinality estimates only in join optimization. If some join is not optimized in the right way, you may try using ANALYZE TABLE.
- InnoDB group commit capability now works also when MySQL binary logging is switched on. There have to be > 2 client threads for the group commit to become active.
- Changed the default value of innodb_flush_log_at_trx_commit from 0 to 1. If you have not specified it explicitly in your my.cnf, and your application runs much slower with this new release, it is because the value 1 causes a log flush to disk at each transaction commit.
- Added a new global settable MySQL system variable innodb_max_dirty_pages_pct. It is an integer in the range 0 - 100. The default is 90. The main thread in InnoDB tries to flush pages from the buffer pool so that at most this many percents are not yet flushed at any time.
- If innodb_force_recovery=6, do not let InnoDB do repair of corrupt pages based on the doublewrite buffer.
- InnoDB startup now happens faster because it does not set the memory in the buffer pool to zero.
- Fixed a bug: The InnoDB parser for FOREIGN KEY definitions was confused by the keywords 'foreign key' inside MySQL comments.
- Fixed a bug: If you dropped a table to which there was a FOREIGN KEY reference, and later created the same table with non-matching column types, InnoDB could assert in dict0load.c, in function dict_load_table().
- Fixed a bug: GROUP BY and DISTINCT could treat NULL values as not equal. MySQL also failed to do the next-key locking in the case of an empty index range.
- Fixed a bug: Do not commit the current transaction when a MyISAM table is updated; this also makes CREATE TABLE not to commit an InnoDB transaction, even when binary logging is enabled.

- Fixed a bug: We did not allow `ON DELETE SET NULL` to modify the same table where the delete was made; we can allow it because that cannot produce infinite loops in cascaded operations.
- Fixed a bug: Allow `HANDLER PREV` and `NEXT` also after positioning the cursor with a unique search on the primary key.
- Fixed a bug: If `MIN()` or `MAX()` resulted in a deadlock or a lock wait timeout, MySQL did not return an error, but returned `NULL` as the function value.
- Fixed a bug: `InnoDB` forgot to call `pthread_mutex_destroy()` when a table was dropped. That could cause memory leakage on FreeBSD and other non-Linux Unix systems.

D.9.17. MySQL/InnoDB-4.1.0, April 3, 2003

- `InnoDB` now supports up to 64GB of buffer pool memory in a Windows 32-bit Intel computer. This is possible because `InnoDB` can use the AWE extension of Windows to address memory over the 4GB limit of a 32-bit process. A new startup variable `innodb_buffer_pool_awe_mem_mb` enables AWE and sets the size of the buffer pool in megabytes.
- Reduced the size of buffer headers and the lock table. `InnoDB` uses 2 % less memory.

D.9.18. MySQL/InnoDB-3.23.56, March 17, 2003

- Fixed a major bug in `InnoDB` query optimization: queries of type SELECT ... WHERE indexcolumn < x and SELECT ... WHERE indexcolumn > x could cause a table scan even if the selectivity would have been very good.
- Fixed a potential bug if MySQL calls store_lock with TL_IGNORE in the middle of a query.

D.9.19. MySQL/InnoDB-4.0.12, March 18, 2003

- In crash recovery `InnoDB` now prints the progress in percents of a transaction rollback.
- Fixed a bug/feature: If your application program used mysql_use_result(), and used >= 2 connections to send SQL queries, it could deadlock on the adaptive hash S-latch in btr0sea.c. Now **mysqld** releases the S-latch whenever it passes data from a SELECT to the client.
- Fixed a bug: MySQL could erroneously return 'Empty set' if `InnoDB` estimated an index range size to 0 records though the range was not empty; MySQL also failed to do the next-key locking in the case of an empty index range.

D.9.20. MySQL/InnoDB-4.0.11, February 25, 2003

- Fixed a bug introduced in 4.0.10: SELECT ... FROM ... ORDER BY ... DESC could hang in an infinite loop.
- An outstanding bug: SET FOREIGN_KEY_CHECKS=0 is not replicated properly in the MySQL replication.

D.9.21. MySQL/InnoDB-4.0.10, February 4, 2003

- In INSERT INTO t1 SELECT ... FROM t2 WHERE ... MySQL previously set a table level read lock on t2. This lock is now removed.
- Increased SHOW INNODB STATUS maximum printed length to 200KB.
- Fixed a major bug in `InnoDB` query optimization: queries of type SELECT ... WHERE indexcolumn < x and SELECT ... WHERE indexcolumn > x could cause a table scan even if the selectivity would have been very good.
- Fixed a bug: purge could cause a hang in a BLOB table where the primary key index tree was of height 1. Symptom: semaphore waits caused by an X-latch set in btr_free_externally_stored_field().
- Fixed a bug: using `InnoDB` HANDLER commands on a fresh handle crashed **mysqld** in `ha_innobase::change_active_index()`.
- Fixed a bug: If MySQL estimated a query in the middle of a SELECT statement, `InnoDB` could hang on the adaptive hash index latch in btr0sea.c.
- Fixed a bug: `InnoDB` could report table corruption and assert in page_dir_find_owner_slot() if an adaptive hash index search coincided with purge or an insert.
- Fixed a bug: some filesystem snapshot tool in Windows 2000 could cause an `InnoDB` file write to fail with error 33 ERROR_LOCK_VIOLATION. In synchronous writes `InnoDB` now retries the write 100 times at 1 second intervals.
- Fixed a bug: REPLACE INTO t1 SELECT ... did not work if t1 has an auto-inc column.
- An outstanding bug: SET FOREIGN_KEY_CHECKS=0 is not replicated properly in the MySQL replication.

D.9.22. MySQL/InnoDB-3.23.55, January 24, 2003

- In INSERT INTO t1 SELECT ... FROM t2 WHERE ... MySQL previously set a table level read lock on t2. This lock is now removed.
- Fixed a bug: If the combined size of `InnoDB` log files was >= 2GB in a 32-bit computer, `InnoDB` would write log in a wrong position. That could make crash recovery and `InnoDB Hot Backup` to fail in log scan.
- Fixed a bug: index cursor restoration could theoretically fail.
- Fixed a bug: an assertion in btr0sea.c, in function btr_search_info_update_slow could theoretically fail in a race of 3 threads.
- Fixed a bug: purge could cause a hang in a BLOB table where the primary key index tree was of height 1. Symptom: semaphore waits caused by an X-latch set in btr_free_externally_stored_field().
- Fixed a bug: If MySQL estimated a query in the middle of a SELECT statement, `InnoDB` could hang on the adaptive hash index latch in btr0sea.c.
- Fixed a bug: `InnoDB` could report table corruption and assert in page_dir_find_owner_slot() if an adaptive hash index search coincided with purge or an insert.
- Fixed a bug: some filesystem snapshot tool in Windows 2000 could cause an `InnoDB` file write to

fail with error 33 ERROR_LOCK_VIOLATION. In synchronous writes `InnoDB` now retries the write 100 times at 1 second intervals.

- An outstanding bug: SET FOREIGN_KEY_CHECKS=0 is not replicated properly in the MySQL replication. The fix appears in 4.0.11 and probably will not be backported to 3.23.
- Fixed bug in `InnoDB` `page0cur.c` file in function page_cur_search_with_match which caused `InnoDB` to remain on the same page forever. This bug is evident only in tables with more than one page.

D.9.23. MySQL/InnoDB-4.0.9, January 14, 2003

- Removed the warning message: 'InnoDB: Out of memory in additional memory pool.'
- Fixed a bug: If the combined size of `InnoDB` log files was >= 2GB in a 32-bit computer, `InnoDB` would write log in a wrong position. That could make crash recovery and `InnoDB Hot Backup` to fail.
- Fixed a bug: index cursor restoration could theoretically fail.

D.9.24. MySQL/InnoDB-4.0.8, January 7, 2003

- `InnoDB` now supports also FOREIGN KEY (...) REFERENCES ...(...) [ON UPDATE CASCADE | ON UPDATE SET NULL | ON UPDATE RESTRICT | ON UPDATE NO ACTION].
- Tables and indexes now reserve 4 % less space in the tablespace. Also existing tables reserve less space. By upgrading to 4.0.8 you should see more free space in "InnoDB free" in SHOW TABLE STATUS.
- Fixed bugs: updating the PRIMARY KEY of a row would generate a foreign key error on all FOREIGN KEYs which referenced secondary keys of the row to be updated. Also, if a referencing FOREIGN KEY constraint only referenced the first columns in an index, and there were more columns in that index, updating the additional columns generated a foreign key error.
- Fixed a bug: If an index contains some column twice, and that column is updated, the table becomes corrupt. From now on `InnoDB` prevents creation of such indexes.
- Fixed a bug: removed superfluous error 149 and 150 printouts from the .err log when a locking SELECT caused a deadlock or a lock wait timeout.
- Fixed a bug: an assertion in btr0sea.c, in function btr_search_info_update_slow could theoretically fail in a race of 3 threads.
- Fixed a bug: one could not switch a session transaction isolation level back to REPEATABLE READ after setting it to something else.

D.9.25. MySQL/InnoDB-4.0.7, December 26, 2002

- `InnoDB` in 4.0.7 is essentially the same as in 4.0.6.

D.9.26. MySQL/InnoDB-4.0.6, December 19, 2002

- Since innodb_log_arch_dir has no relevance under MySQL, there is no need to specify it any more in my.cnf.
- LOAD DATA INFILE in AUTOCOMMIT=1 mode no longer does implicit commits for each 1MB of written binary log.
- Fixed a bug introduced in 4.0.4: LOCK TABLES ... READ LOCAL should not set row locks on the rows read. This caused deadlocks and lock wait timeouts in mysqldump.
- Fixed two bugs introduced in 4.0.4: in AUTO_INCREMENT, REPLACE could cause the counter to be left 1 too low. A deadlock or a lock wait timeout could cause the same problem.
- Fixed a bug: TRUNCATE on a TEMPORARY table crashed `InnoDB`.
- Fixed a bug introduced in 4.0.5: If binary logging was not switched on, INSERT INTO ... SELECT ... or CREATE TABLE ... SELECT ... could cause `InnoDB` to hang on a semaphore created in btr0sea.c, line 128. Workaround: switch binary logging on.
- Fixed a bug: in replication issuing SLAVE STOP in the middle of a multiple-statement transaction could cause that SLAVE START would only perform a part of the transaction. A similar error could occur if the slave crashed and was restarted.

D.9.27. MySQL/InnoDB-3.23.54, December 12, 2002

- Fixed a bug: the `InnoDB` range estimator greatly exaggerated the size of a short index range if the paths to the endpoints of the range in the index tree happened to branch in the root. This could cause unnecessary table scans in SQL queries.
- Fixed a bug: ORDER BY could fail if you had not created a primary key to a table, but had defined several indexes of which at least one was a UNIQUE index with all its columns declared as NOT NULL.
- Fixed a bug: a lock wait timeout in connection with ON DELETE CASCADE could cause corruption in indexes.
- Fixed a bug: If a SELECT was done with a unique key from a primary index, and the search matched to a delete-marked record, `InnoDB` could erroneously return the NEXT record.
- Fixed a bug introduced in 3.23.53: LOCK TABLES ... READ LOCAL should not set row locks on the rows read. This caused deadlocks and lock wait timeouts in mysqldump.
- Fixed a bug: If an index contains some column twice, and that column is updated, the table becomes corrupt. From now on `InnoDB` prevents creation of such indexes.

D.9.28. MySQL/InnoDB-4.0.5, November 18, 2002

- `InnoDB` now supports also transaction isolation levels READ COMMITTED and READ UNCOMMITTED. READ COMMITTED more closely emulates Oracle and makes porting of applications from Oracle to MySQL easier.
- Deadlock resolution is now selective: we try to pick as victims transactions with less modified or in-

serted rows.

- FOREIGN KEY definitions are now aware of the lower_case_table_names setting in my.cnf.
- SHOW CREATE TABLE does not output the database name to a FOREIGN KEY definition if the referred table is in the same database as the table.
- `InnoDB` does a consistency check to most index pages before writing them to a data file.
- If you set `innodb_force_recovery` > 0, `InnoDB` tries to jump over corrupt index records and pages when doing SELECT * FROM table. This helps in dumping.
- `InnoDB` now again uses asynchronous unbuffered I/O in Windows 2000 and XP; only unbuffered simulated async I/O in NT, 95/98/ME.
- Fixed a bug: the `InnoDB` range estimator greatly exaggerated the size of a short index range if the paths to the endpoints of the range in the index tree happened to branch in the root. This could cause unnecessary table scans in SQL queries. The fix is also backported to 3.23.54.
- Fixed a bug present in 3.23.52, 4.0.3, 4.0.4: `InnoDB` startup could take very long or even crash on some Windows 95/98/ME computers.
- Fixed a bug: the AUTO-INC lock was held to the end of the transaction if it was granted after a lock wait. This could cause unnecessary deadlocks.
- Fixed a bug: If SHOW INNODB STATUS, innodb_monitor, or innodb_lock_monitor had to print several hundred transactions in one report, and the output became truncated, `InnoDB` would hang, printing to the error log many waits for a mutex created at srv0srv.c, line 1621.
- Fixed a bug: SHOW INNODB STATUS on Unix always reported average file read size as 0 bytes.
- Fixed a potential bug in 4.0.4: `InnoDB` now does ORDER BY ... DESC like `MyISAM`.
- Fixed a bug: DROP TABLE could cause crash or a hang if there was a rollback concurrently running on the table. The fix will be backported to 3.23 only if this appears a real problem for users.
- Fixed a bug: ORDER BY could fail if you had not created a primary key to a table, but had defined several indexes of which at least one was a UNIQUE index with all its columns declared as NOT NULL.
- Fixed a bug: a lock wait timeout in connection with ON DELETE CASCADE could cause corruption in indexes.
- Fixed a bug: If a SELECT was done with a unique key from a primary index, and the search matched to a delete-marked record, `InnoDB` could return the NEXT record.
- Outstanding bugs: in 4.0.4 two bugs were introduced to AUTO_INCREMENT. REPLACE can cause the counter to be left 1 too low. A deadlock or a lock wait timeout can cause the same problem. These are fixed in 4.0.6.

D.9.29. MySQL/InnoDB-3.23.53, October 9, 2002

- We again use unbuffered disk I/O to data files in Windows. Windows XP and Windows 2000 read performance seems to be very poor with normal I/O.
- Tuned range estimator so that index range scans are preferred over full index scans.

- Allow dropping and creating a table even if innodb_force_recovery is set. One can use this to drop a table which would cause a crash in rollback or purge, or if a failed table import causes a runaway rollback in recovery.
- Fixed a bug present in 3.23.52, 4.0.3, 4.0.4: `InnoDB` startup could take very long or even crash on some Windows 95/98/ME computers.
- Fixed a bug: fast shutdown (which is the default) sometimes was slowed down by purge and insert buffer merge.
- Fixed a bug: doing a big SELECT from a table where no rows were visible in a consistent read could cause a very long (> 600 seconds) semaphore wait in btr0cur.c line 310.
- Fixed a bug: the AUTO-INC lock was held to the end of the transaction if it was granted after a lock wait. This could cause unnecessary deadlocks.
- Fixed a bug: If you created a temporary table inside LOCK TABLES, and used that temporary table, that caused an assertion failure in ha_innobase.cc.
- Fixed a bug: If SHOW INNODB STATUS, innodb_monitor, or innodb_lock_monitor had to print several hundred transactions in one report, and the output became truncated, `InnoDB` would hang, printing to the error log many waits for a mutex created at srv0srv.c, line 1621.
- Fixed a bug: SHOW INNODB STATUS on Unix always reported average file read size as 0 bytes.

D.9.30. MySQL/InnoDB-4.0.4, October 2, 2002

- We again use unbuffered disk I/O in Windows. Windows XP and Windows 2000 read performance seems to be very poor with normal I/O.
- Increased the maximum key length of `InnoDB` tables from 500 to 1024 bytes.
- Increased the table comment field in SHOW TABLE STATUS so that up to 16000 characters of foreign key definitions can be printed there.
- The auto-increment counter is no longer incremented if an insert of a row immediately fails in an error.
- Allow dropping and creating a table even if innodb_force_recovery is set. One can use this to drop a table which would cause a crash in rollback or purge, or if a failed table import causes a runaway rollback in recovery.
- Fixed a bug: Using ORDER BY primarykey DESC in 4.0.3 causes an assertion failure in btr0pcur.c, line 203.
- Fixed a bug: fast shutdown (which is the default) sometimes was slowed down by purge and insert buffer merge.
- Fixed a bug: doing a big SELECT from a table where no rows were visible in a consistent read could cause a very long (> 600 seconds) semaphore wait in btr0cur.c line 310.
- Fixed a bug: If the MySQL query cache was used, it did not get invalidated by a modification done by ON DELETE CASCADE or ...SET NULL.
- Fixed a bug: If you created a temporary table inside LOCK TABLES, and used that temporary table, that caused an assertion failure in ha_innodb.cc.

- Fixed a bug: If you set innodb_flush_log_at_trx_commit to 1, SHOW VARIABLES would show its value as 16 million.

D.9.31. MySQL/InnoDB-4.0.3, August 28, 2002

- Removed unnecessary deadlocks when inserts have to wait for a locking read, update, or delete to release its next-key lock.
- The MySQL `HANDLER` SQL commands now work also for `InnoDB` type tables. `InnoDB` does the `HANDLER` reads always as consistent reads. `HANDLER` is a direct access path to read individual indexes of tables. In some cases, `HANDLER` can be used as a substitute of server-side cursors.
- Fixed a bug in 4.0.2: even a simple insert could crash the AIX version.
- Fixed a bug: If you used in a table name characters whose code is > 127, in DROP TABLE `InnoDB` could assert on line 155 of pars0sym.c.
- Compilation from source now provides a working version both on HP-UX-11 and HP-UX-10.20. The source of 4.0.2 worked only on 11, and the source of 3.23.52 only on 10.20.
- Fixed a bug: If compiled on 64-bit Solaris, `InnoDB` produced a bus error at startup.

D.9.32. MySQL/InnoDB-3.23.52, August 16, 2002

- The feature set of 3.23 is frozen from this version on. New features go the 4.0 branch, and only bugfixes are made to the 3.23 branch.
- Many CPU-bound join queries now run faster. On Windows also many other CPU-bound queries run faster.
- A new SQL command SHOW INNODB STATUS returns the output of the `InnoDB` Monitor to the client. The `InnoDB` Monitor now prints detailed information on the latest detected deadlock.
- `InnoDB` made the SQL query optimizer to avoid too much index-only range scans and choose full table scans instead. This is now fixed.
- `BEGIN` and `COMMIT` are now added in the binary log around transactions. The MySQL replication now respects transaction borders: a user no longer sees half transactions in replication slaves.
- A replication slave now prints in crash recovery the last master binary log position it was able to recover to.
- A new setting `innodb_flush_log_at_trx_commit=2` makes `InnoDB` to write the log to the operating system file cache at each commit. This is almost as fast as the setting innodb_flush_log_at_trx_commit=0, and the setting 2 also has the nice feature that in a crash where the operating system does not crash, no committed transaction is lost. If the operating system crashes or there is a power outage, then the setting 2 is no safer than the setting 0.
- Added checksum fields to log blocks.
- SET FOREIGN_KEY_CHECKS=0 helps in importing tables in an arbitrary order which does not respect the foreign key rules.
- SET UNIQUE_CHECKS=0 speeds up table imports into `InnoDB` if you have UNIQUE constraints

on secondary indexes. This flag should be used only if you are certain that the input records contain no UNIQUE constraint violations.

- SHOW TABLE STATUS now lists also possible ON DELETE CASCADE or ON DELETE SET NULL in the comment field of the table.
- When CHECK TABLE is run on any `InnoDB` type table, it now checks also the adaptive hash index for all tables.
- If you defined ON DELETE CASCADE or SET NULL and updated the referenced key in the parent row, `InnoDB` deleted or updated the child row. This is now changed to conform to standard SQL: you get the error 'Cannot delete parent row'.
- Improved the auto-increment algorithm: now the first insert or SHOW TABLE STATUS initializes the auto-increment counter for the table. This removes almost all surprising deadlocks caused by SHOW TABLE STATUS.
- Aligned some buffers used in reading and writing to data files. This allows using unbuffered raw devices as data files in Linux.
- Fixed a bug: If you updated the primary key of a table so that only the case of characters changed, that could cause assertion failures, mostly in page0page.ic line 515.
- Fixed a bug: If you delete or update a row referenced in a foreign key constraint and the foreign key check has to wait for a lock, then the check may report an erroneous result. This affects also the ON DELETE... operation.
- Fixed a bug: A deadlock or a lock wait timeout error in `InnoDB` causes `InnoDB` to roll back the whole transaction, but MySQL could still write the earlier SQL statements to the binary log, even though `InnoDB` rolled them back. This could, for example, cause replicated databases to get out-of-sync.
- Fixed a bug: If the database happened to crash in the middle of a commit, then the recovery might leak tablespace pages.
- Fixed a bug: If you specified a non-latin1 character set in my.cnf, then, in contrary to what is stated in the manual, in a foreign key constraint a string type column had to have the same length specification in the referencing table and the referenced table.
- Fixed a bug: DROP TABLE or DROP DATABASE could fail if there simultaneously was a CREATE TABLE running.
- Fixed a bug: If you configured the buffer pool bigger than 2GB in a 32-bit computer, `InnoDB` would assert in buf0buf.ic line 214.
- Fixed a bug: on 64-bit computers updating rows which contained the SQL NULL in some column could cause the undo log and the ordinary log to become corrupt.
- Fixed a bug: innodb_log_monitor caused a hang if it suppressed lock prints for a page.
- Fixed a bug: in the HP-UX-10.20 version mutexes would leak and cause race conditions and crashes in any part of `InnoDB` code.
- Fixed a bug: If you ran in the AUTOCOMMIT mode, executed a SELECT, and immediately after that a RENAME TABLE, then RENAME would fail and MySQL would complain about error 1192.
- Fixed a bug: If compiled on 64-bit Solaris, `InnoDB` produced a bus error at startup.

D.9.33. MySQL/InnoDB-4.0.2, July 10, 2002

- `InnoDB` is essentially the same as InnoDB-3.23.51.
- If no innodb_data_file_path is specified, `InnoDB` at the database creation now creates a 10MB auto-extending data file ibdata1 to the datadir of MySQL. In 4.0.1 the file was 64MB and not auto-extending.

D.9.34. MySQL/InnoDB-3.23.51, June 12, 2002

- Fixed a bug: a join could result in a seg fault in copying of a BLOB or TEXT column if some of the BLOB or TEXT columns in the table contained SQL NULL values.
- Fixed a bug: If you added self-referential foreign key constraints with ON DELETE CASCADE to tables and a row deletion caused `InnoDB` to attempt the deletion of the same row twice because of a cascading delete, then you got an assertion failure.
- Fixed a bug: If you use MySQL 'user level locks' and close a connection, then `InnoDB` may assert in ha_innobase.cc, line 302.

D.9.35. MySQL/InnoDB-3.23.50, April 23, 2002

- `InnoDB` now supports an auto-extending last data file. You do not need to preallocate the whole data file at the database startup.
- Made several changes to facilitate the use of the `InnoDB Hot Backup` tool. It is a separate non-free tool you can use to take online backups of your database without shutting down the server or setting any locks.
- If you want to run the `InnoDB Hot Backup` tool on an auto-extending data file you have to upgrade it to version ibbackup-0.35.
- The log scan phase in crash recovery now runs much faster.
- Starting from this server version, the hot backup tool truncates unused ends in the backup `InnoDB` data files.
- To allow the hot backup tool to work, on Windows we no longer use unbuffered I/O or native async I/O; instead we use the same simulated async I/O as on Unix.
- You can now define the ON DELETE CASCADE or ON DELETE SET NULL clause on foreign keys.
- FOREIGN KEY constraints now survive ALTER TABLE and CREATE INDEX.
- We suppress the FOREIGN KEY check if any of the column values in the foreign key or referenced key to be checked is the SQL NULL. This is compatible with Oracle, for example.
- SHOW CREATE TABLE now lists also foreign key constraints. Also mysqldump no longer forgets about foreign keys in table definitions.
- You can now add a new foreign key constraint with ALTER TABLE ... ADD CONSTRAINT FOREIGN KEY (...) REFERENCES ... (...).

- FOREIGN KEY definitions now allow backticks around table and column names.
- MySQL command SET TRANSACTION ISOLATION LEVEL ... has now the following effect on `InnoDB` tables: If a transaction is defined as SERIALIZABLE then `InnoDB` conceptually adds LOCK IN SHARE MODE to all consistent reads. If a transaction is defined to have any other isolation level, then `InnoDB` obeys its default locking strategy which is REPEATABLE READ.
- SHOW TABLE STATUS no longer sets an x-lock at the end of an auto-increment index if the auto-increment counter has been initialized. This removes in almost all cases the surprising deadlocks caused by SHOW TABLE STATUS.
- Fixed a bug: in a CREATE TABLE statement the string 'foreign' followed by a non-space character confused the FOREIGN KEY parser and caused table creation to fail with errno 150.

D.9.36. MySQL/InnoDB-3.23.49, February 17, 2002

- Fixed a bug: If you called DROP DATABASE for a database on which there simultaneously were running queries, the MySQL server could crash or hang. Crashes fixed, but a full fix has to wait some changes in the MySQL layer of code.
- Fixed a bug: on Windows one had to put the database name in lowercase for `DROP DATABASE` to work. Fixed in 3.23.49: case no longer matters on Windows. On Unix, the database name remains case sensitive.
- Fixed a bug: If one defined a non-latin1 character set as the default character set, then definition of foreign key constraints could fail in an assertion failure in dict0crea.c, reporting an internal error 17.

D.9.37. MySQL/InnoDB-3.23.48, February 9, 2002

- Tuned the SQL optimizer to favor more often index searches over table scans.
- Fixed a performance problem when several large SELECT queries are run concurrently on a multi-processor Linux computer. Large CPU-bound SELECT queries now also generally run faster on all platforms.
- If MySQL binary logging is used, `InnoDB` now prints after crash recovery the latest MySQL binary log file name and the position in that file (= byte offset) `InnoDB` was able to recover to. This is useful, for example, when resynchronizing a master and a slave database in replication.
- Added better error messages to help in installation problems.
- One can now recover also MySQL temporary tables which have become orphaned inside the `InnoDB` tablespace.
- `InnoDB` now prevents a FOREIGN KEY declaration where the signedness is not the same in the referencing and referenced integer columns.
- Fixed a bug: calling SHOW CREATE TABLE or SHOW TABLE STATUS could cause memory corruption and make **mysqld** to crash. Especially at risk was mysqldump, because it calls frequently SHOW CREATE TABLE.
- Fixed a bug: If on Unix you did an ALTER TABLE to an `InnoDB` table and simultaneously did queries to it, **mysqld** could crash with an assertion failure in row0row.c, line 474.

- Fixed a bug: If inserts to several tables containing an auto-inc column were wrapped inside one LOCK TABLES, `InnoDB` asserted in lock0lock.c.
- In 3.23.47 we allowed several NULLS in a UNIQUE secondary index. But CHECK TABLE was not relaxed: it reports the table as corrupt. CHECK TABLE no longer complains in this situation.
- Fixed a bug: on Sparc and other high-endian processors SHOW VARIABLES showed innodb_flush_log_at_trx_commit and other boolean-valued startup parameters always OFF even if they were switched on.
- Fixed a bug: If you ran mysqld-max-nt as a service on Windows NT/2000, the service shutdown did not always wait long enough for the `InnoDB` shutdown to finish.

D.9.38. MySQL/InnoDB-3.23.47, December 28, 2001

- Recovery happens now faster, especially in a lightly loaded system, because background checkpointing has been made more frequent.
- `InnoDB` allows now several similar key values in a UNIQUE secondary index if those values contain SQL NULLs. Thus the convention is now the same as in `MyISAM` tables.
- `InnoDB` gives a better row count estimate for a table which contains BLOBs.
- In a FOREIGN KEY constraint `InnoDB` is now case-insensitive to column names, and in Windows also to table names.
- `InnoDB` allows a FOREIGN KEY column of CHAR type to refer to a column of VARCHAR type, and vice versa. MySQL silently changes the type of some columns between CHAR and VARCHAR, and these silent changes do not hinder FOREIGN KEY declaration any more.
- Recovery has been made more resilient to corruption of log files.
- Unnecessary statistics calculation has been removed from queries which generate a temporary table. Some ORDER BY and DISTINCT queries now run much faster.
- MySQL now knows that the table scan of an `InnoDB` table is done through the primary key. This saves a sort in some ORDER BY queries.
- The maximum key length of `InnoDB` tables is again restricted to 500 bytes. The MySQL interpreter is not able to handle longer keys.
- The default value of innodb_lock_wait_timeout was changed from infinite to 50 seconds, the default value of innodb_file_io_threads from 9 to 4.

D.9.39. MySQL/InnoDB-4.0.1, December 23, 2001

- `InnoDB` is the same as in 3.23.47.
- In 4.0.0 the MySQL interpreter did not know the syntax LOCK IN SHARE MODE. This has been fixed.
- In 4.0.0 multiple-table delete did not work for transactional tables. This has been fixed.

D.9.40. MySQL/InnoDB-3.23.46, November 30, 2001

- This is the same as 3.23.45.

D.9.41. MySQL/InnoDB-3.23.45, November 23, 2001

- This is a bugfix release.
- In versions 3.23.42-.44 when creating a table on Windows, you have to use lowercase letters in the database name to be able to access the table. Fixed in 3.23.45.
- `InnoDB` now flushes stdout and stderr every 10 seconds: If these are redirected to files, the file contents can be better viewed with an editor.
- Fixed an assertion failure in .44, in trx0trx.c, line 178 when you drop a table which has the .frm file but does not exist inside `InnoDB`.
- Fixed a bug in the insert buffer. The insert buffer tree could get into an inconsistent state, causing a crash, and also crashing the recovery. This bug could appear especially in large table imports or alterations.
- Fixed a bug in recovery: `InnoDB` could go into an infinite loop constantly printing a warning message that it cannot find free blocks from the buffer pool.
- Fixed a bug: when you created a temporary table of the `InnoDB` type, and then used ALTER TABLE to it, the MySQL server could crash.
- Prevented creation of MySQL system tables 'mysql.user', 'mysql.host', or 'mysql.db', in the `InnoDB` type.
- Fixed a bug which can cause an assertion failure in 3.23.44 in srv0srv.c, line 1728.

D.9.42. MySQL/InnoDB-3.23.44, November 2, 2001

- You can define foreign key constraints on `InnoDB` tables. An example: FOREIGN KEY (col1) REFERENCES table2(col2).
- You can create data files larger than 4GB in those filesystems that allow it.
- Improved `InnoDB` monitors, including a new innodb_table_monitor which allows you to print the contents of the `InnoDB` internal data dictionary.
- DROP DATABASE now works also for `InnoDB` tables.
- Accent characters in the default character set latin1 are ordered according to the MySQL ordering.

  NOTE: If you are using latin1 and have inserted characters whose code is > 127 to an indexed CHAR column, you should run CHECK TABLE on your table when you upgrade to 3.23.43, and drop and reimport the table if CHECK TABLE reports an error!
- `InnoDB` calculates better table cardinality estimates.
- Change in deadlock resolution: in .43 a deadlock rolls back only the SQL statement, in .44 it rolls

back the whole transaction.

- Deadlock, lock wait timeout, and foreign key constraint violations (no parent row, child rows exist) now return native MySQL error codes 1213, 1205, 1216, 1217, respectively.
- A new my.cnf parameter innodb_thread_concurrency helps in performance tuning in high concurrency environments.
- A new my.cnf option innodb_force_recovery helps you in dumping tables from a corrupted database.
- A new my.cnf option innodb_fast_shutdown speeds up shutdown. Normally `InnoDB` does a full purge and an insert buffer merge at shutdown.
- Raised maximum key length to 7000 bytes from a previous limit of 500 bytes.
- Fixed a bug in replication of auto-inc columns with multiline inserts.
- Fixed a bug when the case of letters changes in an update of an indexed secondary column.
- Fixed a hang when there are more than 24 data files.
- Fixed a crash when MAX(col) is selected from an empty table, and col is a not the first column in a multi-column index.
- Fixed a bug in purge which could cause crashes.

D.9.43. MySQL/InnoDB-3.23.43, October 4, 2001

- This is essentially the same as InnoDB-3.23.42.

D.9.44. MySQL/InnoDB-3.23.42, September 9, 2001

- Fixed a bug which corrupted the table if the primary key of a > 8000-byte row was updated.
- There are now 3 types of `InnoDB` Monitors: innodb_monitor, innodb_lock_monitor, and innodb_tablespace_monitor. innodb_monitor now prints also buffer pool hit rate and the total number of rows inserted, updated, deleted, read.
- Fixed a bug in RENAME TABLE.
- Fixed a bug in replication with an auto-increment column.

D.9.45. MySQL/InnoDB-3.23.41, August 13, 2001

- Support for < 4GB rows. The previous limit was 8000 bytes.
- Use the doublewrite file flush method.
- Raw disk partitions supported as data files.
- `InnoDB` Monitor.

- Several hang bugs fixed and an `ORDER BY` bug (``Sort aborted'') fixed.

D.9.46. MySQL/InnoDB-3.23.40, July 16, 2001

- Only a few rare bugs fixed.

D.9.47. MySQL/InnoDB-3.23.39, June 13, 2001

- `CHECK TABLE` now works for `InnoDB` tables.
- A new `my.cnf` parameter `innodb_unix_file_flush_method` introduced. It can be used to tune disk write performance.
- An auto-increment column now gets new values past the transaction mechanism. This saves CPU time and eliminates transaction deadlocks in new value assignment.
- Several bugfixes, most notably the rollback bug in 3.23.38.

D.9.48. MySQL/InnoDB-3.23.38, May 12, 2001

- The new syntax `SELECT ... LOCK IN SHARE MODE` is introduced.
- `InnoDB` now calls `fsync()` after every disk write and calculates a checksum for every database page it writes or reads, which reveals disk defects.
- Several bugfixes.

D.10. `MySQL Cluster` Change History

D.10.1. MySQL Cluster-4.1.10 (12 Feb 2005)

Functionality added or changed:

Bugs fixed:

- (Bug #8284 [http://bugs.mysql.com/8284]) Out of fragment memory in DBACC
- (Bug #8262 [http://bugs.mysql.com/8262]) Node crash due to bug in DBLQH
- (Bug #8208 [http://bugs.mysql.com/8208]) node restart fails on Aix 5.2
- (Bug #8167 [http://bugs.mysql.com/8167]) cluster shared memory and mysqld signal usage clash
- (Bug #8101 [http://bugs.mysql.com/8101]) unique index and error 4209 while selecting
- (Bug #8070 [http://bugs.mysql.com/8070]) (Bug #7937 [http://bugs.mysql.com/7937]) (Bug #6716 [http://bugs.mysql.com/6716]) various ndb_restore core dumps on HP-UX

- (Bug #8010 [http://bugs.mysql.com/8010]) 4006 forces MySQL Node Restart
- (Bug #7928 [http://bugs.mysql.com/7928]) out of connection objects
- (Bug #7898 [http://bugs.mysql.com/7898]) mysqld crash with ndb (solaris)
- (Bug #7864 [http://bugs.mysql.com/7864]) Not possible to have more than 4.5G data memory

D.10.2. MySQL Cluster-4.1.9 (13 Jan 2005)

Functionality added or changed:

- New implementation of shared memory transporter.
- Cluster automatically configures shared memory transporter if possible.
- Cluster prioritizes usage of transporters with shared memory and localhost TCP
- Added switches to control the above functions, `ndb-shm` and `ndb-optimized-node-selection`.

Bugs fixed:

- (Bug #7805 [http://bugs.mysql.com/7805]) config.ini parsing error
- (Bug #7798 [http://bugs.mysql.com/7798]) Running range scan after alter table in different thread causes node failure
- (Bug #7761 [http://bugs.mysql.com/7761]) Alter table does not autocommit
- (Bug #7725 [http://bugs.mysql.com/7725]) Indexed DATETIME Columns Return Random Results
- (Bug #7660 [http://bugs.mysql.com/7660]) START BACKUP does not increment BACKUP-ID (Big Endian machines)
- (Bug #7593 [http://bugs.mysql.com/7593]) Cannot Create A Large NDB Data Warehouse
- (Bug #7480 [http://bugs.mysql.com/7480]) Mysqld crash in ha_ndbcluster using Query Browser
- (Bug #7470 [http://bugs.mysql.com/7470]) shared memory transporter does not connect
- (Bug #7396 [http://bugs.mysql.com/7396]) Primary Key not working in NDB Mysql Clustered table (solaris)
- (Bug #7379 [http://bugs.mysql.com/7379]) ndb restore fails to handle blobs and multiple databases
- (Bug #7346 [http://bugs.mysql.com/7346]) ndb_restore enters infinite loop
- (Bug #7340 [http://bugs.mysql.com/7340]) Problem for inserting data into the Text field on utf8
- (Bug #7124 [http://bugs.mysql.com/7124]) ndb_mgmd is aborted on startup when using SHM connection

D.10.3. MySQL Cluster-4.1.8 (14 Dec 2004)

Functionality added or changed:

- Default port for **ndb_mgmd** was changed to 1186 (from 2200) as this port number was officially assigned to MySQL Cluster by IANA.
- New command in **ndb_mgm**, PURGE STALE SESSIONS, as a workaround for cases where nodes fail to allocate a node id even if it is free to use.
- New command in **ndb_mgm**, CONNECT.
- The ndb executables have been changed to make use of the regular MySQL command line option parsing features. See Section 16.6.5, “Command Options for MySQL Cluster Processes” for notes on changes.
- As bonus of the above you can now specify all command line options in `my.cnf` using the executable names as sections, i.e. `[ndbd]`, `[ndb_mgmd]`, `[ndb_mgm]`, `[ndb_restore]` etc.

  ```
  [ndbd]
  ndb-connectstring=myhost.domain.com:1234
  [ndb_mgm]
  ndb-connectstring=myhost.domain.com:1234
  ```

- Added use of section `[mysql_cluster]` in `my.cnf`. All cluster executables, including mysqld, parse this section. Convenient place to put e.g. `ndb-connectstring` so that it only needs to be specified once.
- Added cluster log info events on allocation and deallocation of nodeid's.
- Added cluster log info events on connection refuse as a result of version mismatch.
- Extended connectstring syntax to allow for leaving the port number out. E.g. `ndb-connectstring|connect-string=myhost1,myhost2,myhost3` is a valid connectstring and connect occurs on default port 1186.
- Clear text ndb error messages provided also for error codes that are mapped to corresponding mysql error codes, by executing `SHOW WARNINGS` after an error has occured which relates to the ndb storage engine.
- Significant performance improvements done for read performance, especially for blobs.
- Added some variables for performance tuning, `ndb_force_send` and `ndb_use_exact_count`. Do `show variables like 'ndb%';` in mysql client for listing. Use `set` command to alter variables.
- Added variables to set some options, `ndb_use_transactions` and `ndb_autoincrement_prefetch_sz`.

Bugs fixed:

- (Bug #7303 [http://bugs.mysql.com/7303]) ndb_mgm: Trying to set CLUSTERLOG for a specific node id core dumps
- (Bug #7193 [http://bugs.mysql.com/7193]) start backup gives false error printout

- (Bug #7153 [http://bugs.mysql.com/7153]) Cluster nodes don't report error on endianness mismatch
- (Bug #7152 [http://bugs.mysql.com/7152]) ndb_mgmd segfaults on incorrect HostName in configuration
- (Bug #7104 [http://bugs.mysql.com/7104]) clusterlog filtering and level setting broken
- (Bug #6995 [http://bugs.mysql.com/6995]) ndb_recover on varchar fields results in changing case of data
- (Bug #6919 [http://bugs.mysql.com/6919]) all status only shows 2 nodes on a 8-node cluster
- (Bug #6871 [http://bugs.mysql.com/6871]) DBD execute failed: Got error 897 'Unknown error code' from ndbcluster
- (Bug #6794 [http://bugs.mysql.com/6794]) Wrong outcome of update operation of ndb table
- (Bug #6791 [http://bugs.mysql.com/6791]) Segmentation fault when config.ini is not correctly set
- (Bug #6775 [http://bugs.mysql.com/6775]) failure in acc when running many mysql clients
- (Bug #6696 [http://bugs.mysql.com/6696]) ndb_mgm command line options inconsistent with behavior
- (Bug #6684 [http://bugs.mysql.com/6684]) ndb_restore doesn't give error messages if inproper command given
- (Bug #6677 [http://bugs.mysql.com/6677]) ndb_mgm can crash on "ALL CLUSTERLOG"
- (Bug #6538 [http://bugs.mysql.com/6538]) Error code returned when select max() on empty table with index
- (Bug #6451 [http://bugs.mysql.com/6451]) failing create table givers "ghost" tables which are impossible to remove
- (Bug #6435 [http://bugs.mysql.com/6435]) strange behavior of left join
- (Bug #6426 [http://bugs.mysql.com/6426]) update with long pk fails
- (Bug #6398 [http://bugs.mysql.com/6398]) update of primary key fails
- (Bug #6354 [http://bugs.mysql.com/6354]) mysql does not complain about --ndbcluster option when NDB is not compiled in
- (Bug #6331 [http://bugs.mysql.com/6331]) INSERT IGNORE .. SELECT breaks subsequent inserts
- (Bug #6288 [http://bugs.mysql.com/6288]) cluster nodes crash on data import
- (Bug #6031 [http://bugs.mysql.com/6031]) To drop database you have to execute DROP DATABASE command twice
- (Bug #6020 [http://bugs.mysql.com/6020]) LOCK TABLE + delete returns error 208
- (Bug #6018 [http://bugs.mysql.com/6018]) REPLACE does not work for BLOBs + NDB
- (Bug #6016 [http://bugs.mysql.com/6016]) Strange crash with blobs + different DATABASES
- (Bug #5973 [http://bugs.mysql.com/5973]) ndb table belonging to different database shows up in show tables

- (Bug #5872 [http://bugs.mysql.com/5872]) ALTER TABLE with blob from ndb table to myisam fails
- (Bug #5844 [http://bugs.mysql.com/5844]) Failing mysql-test-run leaves stray NDB processes behind
- (Bug #5824 [http://bugs.mysql.com/5824]) HELP text messed up in ndb_mgm
- (Bug #5786 [http://bugs.mysql.com/5786]) Duplicate key error after restore
- (Bug #5785 [http://bugs.mysql.com/5785]) lock timeout during concurrent update
- (Bug #5782 [http://bugs.mysql.com/5782]) Unknown error when using LIMIT with ndb table
- (Bug #5756 [http://bugs.mysql.com/5756]) RESTART node from ndb_mgm fails
- A few more not reported bugs fixed

D.10.4. MySQL Cluster-4.1.7 (23 Oct 2004)

Functionality added or changed:

- Optimization 1: Improved performance on index scans. Measured 30% performance increase on query which do large amounts of index scans.
- Optimization 2: Improved performance on primary key lookups. Around double performance for autocommitted primary key lookups.
- Optimization 3: Improved performance when using blobs by avoiding usage of exclusive locks for blobs.

Bugs fixed:

- A few bugs fixed.

D.10.5. MySQL Cluster-4.1.6 (10 Oct 2004)

Functionality added or changed:

- Limited character set support for storage engine NDBCLUSTER:

Char set	Collation
big5	big5_chinese_ci
	big5_bin
binary	binary
euckr	euckr_korean_ci
	euckr_bin
gb2312	gb2312_chinese_ci
	gb2312_bin

gbk	gbk_chinese_ci
	gbk_bin
latin1	latin1_swedish_ci
	latin1_bin
sjis	sjis_japanese_ci
	sjis_bin
tis620	tis620_bin
ucs2	ucs2_general_ci
	ucs2_bin
ujis	ujis_japanese_ci
	ujis_bin
utf8	utf8_general_ci
	utf8_bin

- The SCI Transporter has been brought up-to-date with all changes and now works and has been documented as well.
- Optimizations when several clients to a MySQL Server access ndb tables.
- Added more checks and warnings for erroneous and unappropriate cluster configurations.
- `SHOW TABLES` now directly shows ndb tables created on a different mysql server, i.e. without a prior table access.
- Enhanced support for starting MySQL Server independently of ndbd and ndb_mgmd.
- Clear text ndb error messages provided by executing `SHOW WARNINGS` after an error has occured which relates to the ndb storage engine.

Bugs fixed:

- Quite a few bugs fixed.

D.10.6. MySQL Cluster-4.1.5 (16 Sep 2004)

Functionality added or changed:

- Many queries in MySQL Cluster are executed as range scans or full table scans. All queries that don't use a unique hash index or the primary hash index use this access method. In a distributed system it is crucial that batching is properly performed.

  In previous version the batch size was fixed to 16 per storage node. In this version it is configurable per MySQL Server. So for queries using lots of large scans it is appropriate to set this parameter rather large and for queries using lots of small scans only fetching a small amount of records it is appropriate to set it low.

  The performance of queries can easily change as much as 40% based on how this variable is set.

In future versions more logic will be made to assess the batch size on per query basis. Thus the semantics of this new configuration variable `ScanBatchSize` is likely to change.

- The fixed size overhead of the ndbd process was greatly decreased. Also overhead per operation record was greatly decreased and also overhead per table and index was greatly decreased.

  A number of new configuration variables was introduced to be able to configure more buffers in the system. Configuration variables to specify the number of tables, unique hash indexes and ordered indexes was introduced as well.

  New configuration variables: `MaxNoOfOrderedIndexes`, `MaxNoOfUniqueHashIndexes`

  Configuration variables no longer used: `MaxNoOfIndexes` (split into the two above).

- In previous versions `ALTER TABLE`, `TRUNCATE TABLE`, and `LOAD DATA` were performed as one big transaction. In this version, all those statements are automatically separated into a number of transactions.

  This removes the limitation that one could not change very large tables due to the `MaxNoOfConcurrentOperations` parameter.

- The online backup feature of MySQL Cluster now also backs up the indexes such that the restore ensures that both data and indexes are restored.

- In previous versions it was not possible to use `NULL` in indexes. This is now possible in all indexes.

- Much work has been put onto making `AUTO_INCREMENT` features work as for other table handlers. Autoincrements as a partial key is still only supported by `MyISAM`.

- In previous version,s **mysqld** would crash if the cluster wasn't started and the `--ndbcluster` option was used. Now **mysqld** handles cluster crashes and starts without crashing.

- The `-i` option for initial startup has been removed from **ndbd**. Initial startup still can be specified by using the `--initial` option. The reason is to ensure that it is clearer what takes place when using the `--initial` option. This option completely removes all data from the disk and should only be used at initial start, in certain software upgrade cases, and in some cases when node restarts don't work as a workaround.

- The management client (**ndb_mgm**) now has more commands and more information is printed in some commands such as `show`.

- In previous versions, the files were called `ndb_0..` when it wasn't possible to allocate a node ID when starting the node. To ensure that files are not so easily overwritten, these files are now named `ndb_pid..`, where pid is the process ID assigned by the OS.

- The default parameters have changed for **ndb_mgmd** and **ndbd**. In particular, they are now started as daemons by default. The `-n` option was removed since it could confusion as to whether its meaning is nostart or nodaemon.

- In the configuration file, you can now use `[NDBD]` as an alias for `[DB]`, `[MYSQLD]` as an alias for `[API]`, and `[NDB_MGMD]` as an alias for `[MGM]`. **Note**: `[NDBD]`, `[MYSQLD]`, and `[NDB_MGMD]` are now the preferred designations, although the older ones will continue to be supported for some time in order to provide backwards compatibility.

- Many more checks of the consistency of the configuration have been introduced to provide quicker feedback on configuration errors.

- In the connect string, it is now possible to use both '`;`' and '`,`' as the separator between entries. So

"nodeid=2,host=localhost:2200" is equivalent to "nodeid=2;host=localhost:2200".

In the configuration, it is possible to use `':'` or `'='` as the assignment symbol. Thus `MaxNoOfOrderedIndexes : 128` and `MaxNoOfOrderedIndexes = 128` are equivalent.

- The configuration variable names are now case insensitive so `MaxNoOfOrderedIndexes: 128` is equivalent to `MAXNOOFORDEREDINDEXES = 128`.
- It is possible now to set the backup directory separately from the `FileSystemPath` by using the `BackupDir` config variable.

  Log files and trace files can now be put in any directory by setting the `DataDir` configuration variable.

  `FileSystemPath` is no longer mandatory and defaults to `DataDir`.
- It is now supported to perform queries involving tables from different databases in MySQL.
- It is now possible to update the primary key.
- The performance of the ordered index has been greatly improved, particularly the maintenance of the index at updates, inserts and deletes.

Bugs fixed:

- Quite a few bugs fixed.

D.10.7. MySQL Cluster-4.1.4 (31 Aug 2004)

Functionality added or changed:

- The names of the log files and trace files created by the **ndbd** and **ndb_mgmd** processes have changed.
- Support for the many `BLOB` data types was introduced in this version.

Bugs fixed:

- Quite a few bugs were fixed in the 4.1.4 release.

D.10.8. MySQL Cluster-5.0.1 (27 Jul 2004)

Functionality added or changed:

- This was the first MySQL Cluster release in 5.0. Actually almost all attention was on getting 4.1 stable so it is not recommended to use MySQL 5.0.1 for MySQL Cluster usage.

Bugs fixed:

•

D.10.9. MySQL Cluster-4.1.3 (28 Jun 2004)

Functionality added or changed:

- This was the first MySQL Cluster release so all the functionality was new.

Bugs fixed:

- Various bugs fixed in the development process leading up to 4.1.3.

D.11. Changes in MyODBC

D.11.1. Changes in MyODBC 3.51.11

Functionality added or changed:

•

Bugs fixed:

- `mysql_list_dbcolumns()` and `insert_fields()` were retrieving all rows from a table. Fixed the queries generated by these functions to return no rows. (Bug #8198 [http://bugs.mysql.com/8198])
- `SQLGetTypoInfo()` returned `tinyblob` for `SQL_VARBINARY` and nothing for `SQL_BINARY`. Fixed to return `varbinary` for `SQL_VARBINARY`, `binary` for `SQL_BINARY`, and `longblob` for `SQL_LONGVARBINARY`. (Bug #8138 [http://bugs.mysql.com/8138])

Appendix E. Porting to Other Systems

This appendix helps you port MySQL to other operating systems. Do check the list of currently supported operating systems first. See Section 2.1.1, “Operating Systems Supported by MySQL”. If you have created a new port of MySQL, please let us know so that we can list it here and on our Web site (http://www.mysql.com/), recommending it to other users.

Note: If you create a new port of MySQL, you are free to copy and distribute it under the GPL license, but it does not make you a copyright holder of MySQL.

A working POSIX thread library is needed for the server. On Solaris 2.5 we use Sun PThreads (the native thread support in 2.4 and earlier versions is not good enough), on Linux we use LinuxThreads by Xavier Leroy, `<Xavier.Leroy@inria.fr>`.

The hard part of porting to a new Unix variant without good native thread support is probably to port MIT-pthreads. See `mit-pthreads/README` and Programming POSIX Threads (http://www.humanfactor.com/pthreads/).

Up to MySQL 4.0.2, the MySQL distribution included a patched version of Chris Provenzano's Pthreads from MIT (see the MIT Pthreads Web page at http://www.mit.edu/afs/sipb/project/pthreads/ and a programming introduction at http://www.mit.edu:8001/people/proven/IAP_2000/). These can be used for some operating systems that do not have POSIX threads. See Section 2.8.5, “MIT-pthreads Notes”.

It is also possible to use another user level thread package named FSU Pthreads (see http://moss.csc.ncsu.edu/~mueller/pthreads/). This implementation is being used for the SCO port.

See the `thr_lock.c` and `thr_alarm.c` programs in the `mysys` directory for some tests/examples of these problems.

Both the server and the client need a working C++ compiler. We use **gcc** on many platforms. Other compilers that are known to work are SPARCworks, Sun Forte, Irix **cc**, HP-UX **aCC**, IBM AIX **xlC_r**), Intel **ecc/icc** and Compaq **cxx**).

To compile only the client use **./configure --without-server**.

There is currently no support for only compiling the server, nor is it likely to be added unless someone has a good reason for it.

If you want/need to change any `Makefile` or the configure script you also need GNU Automake and Autoconf. See Section 2.8.3, “Installing from the Development Source Tree”.

All steps needed to remake everything from the most basic files.

```
/bin/rm */.deps/*.P
/bin/rm -f config.cache
aclocal
autoheader
aclocal
automake
autoconf
./configure --with-debug=full --prefix='your installation directory'

# The makefiles generated above need GNU make 3.75 or newer.
# (called gmake below)
gmake clean all install init-db
```

If you run into problems with a new port, you may have to do some debugging of MySQL! See Sec-

tion E.1, “Debugging a MySQL Server”.

Note: Before you start debugging **mysqld**, first get the test programs `mysys/thr_alarm` and `mysys/thr_lock` to work. This ensures that your thread installation has even a remote chance to work!

E.1. Debugging a MySQL Server

If you are using some functionality that is very new in MySQL, you can try to run **mysqld** with the `--skip-new` (which disables all new, potentially unsafe functionality) or with `--safe-mode` which disables a lot of optimization that may cause problems. See Section A.4.2, “What to Do If MySQL Keeps Crashing”.

If **mysqld** doesn't want to start, you should verify that you don't have any `my.cnf` files that interfere with your setup! You can check your `my.cnf` arguments with **mysqld --print-defaults** and avoid using them by starting with **mysqld --no-defaults ...**.

If **mysqld** starts to eat up CPU or memory or if it ``hangs," you can use **mysqladmin processlist status** to find out if someone is executing a query that takes a long time. It may be a good idea to run **mysqladmin -i10 processlist status** in some window if you are experiencing performance problems or problems when new clients can't connect.

The command **mysqladmin debug** dumps some information about locks in use, used memory and query usage to the MySQL log file. This may help solve some problems. This command also provides some useful information even if you haven't compiled MySQL for debugging!

If the problem is that some tables are getting slower and slower you should try to optimize the table with `OPTIMIZE TABLE` or **myisamchk**. See Chapter 5, *Database Administration*. You should also check the slow queries with `EXPLAIN`.

You should also read the OS-specific section in this manual for problems that may be unique to your environment. See Section 2.12, “Operating System-Specific Notes”.

E.1.1. Compiling MySQL for Debugging

If you have some very specific problem, you can always try to debug MySQL. To do this you must configure MySQL with the `--with-debug` or the `--with-debug=full` option. You can check whether MySQL was compiled with debugging by doing: **mysqld --help**. If the `--debug` flag is listed with the options then you have debugging enabled. **mysqladmin ver** also lists the **mysqld** version as **mysql ... --debug** in this case.

If you are using **gcc** or **egcs**, the recommended **configure** line is:

```
CC=gcc CFLAGS="-O2" CXX=gcc CXXFLAGS="-O2 -felide-constructors \
   -fno-exceptions -fno-rtti" ./configure --prefix=/usr/local/mysql \
   --with-debug --with-extra-charsets=complex
```

This avoids problems with the `libstdc++` library and with C++ exceptions (many compilers have problems with C++ exceptions in threaded code) and compile a MySQL version with support for all character sets.

If you suspect a memory overrun error, you can configure MySQL with `--with-debug=full`, which installs a memory allocation (`SAFEMALLOC`) checker. However, running with `SAFEMALLOC` is quite slow, so if you get performance problems you should start **mysqld** with the `--skip-safemalloc` option. This disables the memory overrun checks for each call to `malloc()` and `free()`.

If **mysqld** stops crashing when you compile it with `--with-debug`, you probably have found a compiler bug or a timing bug within MySQL. In this case, you can try to add `-g` to the `CFLAGS` and `CXXFLAGS` variables above and not use `--with-debug`. If **mysqld** dies, you can at least attach to it with **gdb** or use **gdb** on the core file to find out what happened.

When you configure MySQL for debugging you automatically enable a lot of extra safety check functions that monitor the health of **mysqld**. If they find something ``unexpected,'' an entry is written to `stderr`, which **safe_mysqld** directs to the error log! This also means that if you are having some unexpected problems with MySQL and are using a source distribution, the first thing you should do is to configure MySQL for debugging! (The second thing is to send mail to a MySQL mailing list and ask for help. See Section 1.4.1.1, “The MySQL Mailing Lists”. Please use the **mysqlbug** script for all bug reports or questions regarding the MySQL version you are using!

In the Windows MySQL distribution, `mysqld.exe` is by default compiled with support for trace files.

E.1.2. Creating Trace Files

If the **mysqld** server doesn't start or if you can cause it to crash quickly, you can try to create a trace file to find the problem.

To do this, you must have a **mysqld** that has been compiled with debugging support. You can check this by executing `mysqld -V`. If the version number ends with `-debug`, it's compiled with support for trace files.

Start the **mysqld** server with a trace log in `/tmp/mysqld.trace` on Unix or `C:\mysqld.trace` on Windows:

```
shell> mysqld --debug
```

On Windows, you should also use the `--standalone` flag to not start **mysqld** as a service. In a console window, use this command:

```
C:\> mysqld --debug --standalone
```

After this, you can use the `mysql.exe` command-line tool in a second console window to reproduce the problem. You can stop the **mysqld** server with **mysqladmin shutdown**.

Note that the trace file become **very big**! If you want to generate a smaller trace file, you can use debugging options something like this:

mysqld --debug=d,info,error,query,general,where:O,/tmp/mysqld.trace

This only prints information with the most interesting tags to the trace file.

If you make a bug report about this, please only send the lines from the trace file to the appropriate mailing list where something seems to go wrong! If you can't locate the wrong place, you can ftp the trace file, together with a full bug report, to ftp://ftp.mysql.com/pub/mysql/upload/ so that a MySQL developer can take a look a this.

The trace file is made with the **DBUG** package by Fred Fish. See Section E.3, “The DBUG Package”.

E.1.3. Debugging mysqld under gdb

On most systems you can also start **mysqld** from **gdb** to get more information if **mysqld** crashes.

With some older **gdb** versions on Linux you must use `run --one-thread` if you want to be able to

debug **mysqld** threads. In this case, you can only have one thread active at a time. We recommend you to upgrade to gdb 5.1 ASAP as thread debugging works much better with this version!

NTPL threads (the new thread library on Linux) may cause problems while running **mysqld** under **gdb**. Some symptoms are:

- **mysqld** hangs during startup (before it writes `ready for connections`).
- **mysqld** crashes during a `pthread_mutex_lock()` or `pthread_mutex_unlock()` call.

In this case you should set the following environment variable in the shell before starting **gdb**:

```
LD_ASSUME_KERNEL=2.4.1
export LD_ASSUME_KERNEL
```

When running **mysqld** under **gdb**, you should disable the stack trace with `--skip-stack-trace` to be able to catch segfaults within **gdb**.

In MySQL 4.0.14 and above you should use the `--gdb` option to mysqld. This installs an interrupt handler for `SIGINT` (needed to stop **mysqld** with `^C` to set breakpoints) and disable stack tracing and core file handling.

It's very hard to debug MySQL under **gdb** if you do a lot of new connections the whole time as **gdb** doesn't free the memory for old threads. You can avoid this problem by starting **mysqld** with `-O thread_cache_size= 'max_connections +1'`. In most cases just using `-O thread_cache_size=5'` helps a lot!

If you want to get a core dump on Linux if **mysqld** dies with a SIGSEGV signal, you can start **mysqld** with the `--core-file` option. This core file can be used to make a backtrace that may help you find out why **mysqld** died:

```
shell> gdb mysqld core
gdb>   backtrace full
gdb>   exit
```

See Section A.4.2, “What to Do If MySQL Keeps Crashing”.

If you are using **gdb** 4.17.x or above on Linux, you should install a `.gdb` file, with the following information, in your current directory:

```
set print sevenbit off
handle SIGUSR1 nostop noprint
handle SIGUSR2 nostop noprint
handle SIGWAITING nostop noprint
handle SIGLWP nostop noprint
handle SIGPIPE nostop
handle SIGALRM nostop
handle SIGHUP nostop
handle SIGTERM nostop noprint
```

If you have problems debugging threads with **gdb**, you should download gdb 5.x and try this instead. The new **gdb** version has very improved thread handling!

Here is an example how to debug mysqld:

```
shell> gdb /usr/local/libexec/mysqld
gdb> run
```

```
...
backtrace full # Do this when mysqld crashes
```

Include the above output in a mail generated with **mysqlbug** and mail this to the general MySQL mailing list. See Section 1.4.1.1, “The MySQL Mailing Lists”.

If **mysqld** hangs you can try to use some system tools like `strace` or `/usr/proc/bin/pstack` to examine where **mysqld** has hung.

```
strace /tmp/log libexec/mysqld
```

If you are using the Perl `DBI` interface, you can turn on debugging information by using the `trace` method or by setting the `DBI_TRACE` environment variable.

E.1.4. Using a Stack Trace

On some operating systems, the error log contains a stack trace if **mysqld** dies unexpectedly. You can use this to find out where (and maybe why) **mysqld** died. See Section 5.9.1, “The Error Log”. To get a stack trace, you must not compile **mysqld** with the `-fomit-frame-pointer` option to gcc. See Section E.1.1, “Compiling MySQL for Debugging”.

If the error file contains something like the following:

```
mysqld got signal 11;
The manual section 'Debugging a MySQL server' tells you how to use a
stack trace and/or the core file to produce a readable backtrace that may
help in finding out why mysqld died
Attempting backtrace. You can use the following information to find out
where mysqld died.  If you see no messages after this, something went
terribly wrong...
stack range sanity check, ok, backtrace follows
0x40077552
0x81281a0
0x8128f47
0x8127be0
0x8127995
0x8104947
0x80ff28f
0x810131b
0x80ee4bc
0x80c3c91
0x80c6b43
0x80c1fd9
0x80c1686
```

you can find where **mysqld** died by doing the following:

1. Copy the preceding numbers to a file, for example `mysqld.stack`.
2. Make a symbol file for the **mysqld** server:

   ```
   nm -n libexec/mysqld > /tmp/mysqld.sym
   ```

   Note that most MySQL binary distributions (except for the "debug" packages, where this information is included inside of the binaries themselves) ship with the above file, named `mysqld.sym.gz`. In this case, you can simply unpack it by doing:

```
gunzip < bin/mysqld.sym.gz > /tmp/mysqld.sym
```

3. Execute `resolve_stack_dump -s /tmp/mysqld.sym -n mysqld.stack`.

   This prints out where **mysqld** died. If this doesn't help you find out why **mysqld** died, you should make a bug report and include the output from the above command with the bug report.

   Note however that in most cases it does not help us to just have a stack trace to find the reason for the problem. To be able to locate the bug or provide a workaround, we would in most cases need to know the query that killed **mysqld** and preferable a test case so that we can repeat the problem! See Section 1.4.1.3, "How to Report Bugs or Problems".

E.1.5. Using Log Files to Find Cause of Errors in mysqld

Note that before starting **mysqld** with `--log` you should check all your tables with **myisamchk**. See Chapter 5, *Database Administration*.

If **mysqld** dies or hangs, you should start **mysqld** with `--log`. When **mysqld** dies again, you can examine the end of the log file for the query that killed **mysqld**.

If you are using `--log` without a file name, the log is stored in the database directory as *`host_name`*`.log` In most cases it is the last query in the log file that killed **mysqld**, but if possible you should verify this by restarting **mysqld** and executing the found query from the **mysql** command-line tools. If this works, you should also test all complicated queries that didn't complete.

You can also try the command `EXPLAIN` on all `SELECT` statements that takes a long time to ensure that **mysqld** is using indexes properly. See Section 7.2.1, "`EXPLAIN` Syntax (Get Information About a `SELECT`)".

You can find the queries that take a long time to execute by starting **mysqld** with `--log-slow-queries`. See Section 5.9.5, "The Slow Query Log".

If you find the text `mysqld restarted` in the error log file (normally named `hostname.err`) you probably have found a query that causes **mysqld** to fail. If this happens, you should check all your tables with **myisamchk** (see Chapter 5, *Database Administration*), and test the queries in the MySQL log files to see if one doesn't work. If you find such a query, try first upgrading to the newest MySQL version. If this doesn't help and you can't find anything in the `mysql` mail archive, you should report the bug to a MySQL mailing list. The mailing lists are described at http://lists.mysql.com/, which also has links to online list archives.

If you have started **mysqld** with `myisam-recover`, MySQL automatically checks and tries to repair `MyISAM` tables if they are marked as 'not closed properly' or 'crashed'. If this happens, MySQL writes an entry in the `hostname.err` file `'Warning: Checking table ...'` which is followed by `Warning: Repairing table` if the table needs to be repaired. If you get a lot of these errors, without **mysqld** having died unexpectedly just before, then something is wrong and needs to be investigated further. See Section 5.2.1, "**mysqld** Command-Line Options".

It's not a good sign if **mysqld** did die unexpectedly, but in this case one shouldn't investigate the `Checking table...` messages but instead try to find out why **mysqld** died.

E.1.6. Making a Test Case If You Experience Table Corruption

If you get corrupted tables or if **mysqld** always fails after some update commands, you can test whether

this bug is reproducible by doing the following:

- Take down the MySQL daemon (with **mysqladmin shutdown**).
- Make a backup of the tables (to guard against the very unlikely case that the repair does something bad).
- Check all tables with **myisamchk -s database/*.MYI**. Repair any wrong tables with **myisamchk -r database/*table*.MYI**.
- Make a second backup of the tables.
- Remove (or move away) any old log files from the MySQL data directory if you need more space.
- Start **mysqld** with `--log-bin`. See Section 5.9.4, “The Binary Log”. If you want to find a query that crashes **mysqld**, you should use `--log --log-bin`.
- When you have gotten a crashed table, stop the `mysqld server`.
- Restore the backup.
- Restart the **mysqld** server **without** `--log-bin`
- Re-execute the commands with **mysqlbinlog update-log-file | mysql**. The update log is saved in the MySQL database directory with the name `hostname-bin.#`.
- If the tables are corrupted again or you can get **mysqld** to die with the above command, you have found reproducible bug that should be easy to fix! FTP the tables and the binary log to ftp://ftp.mysql.com/pub/mysql/upload/ and enter it into our bugs system at http://bugs.mysql.com/. If you are a support customer, you can use the MySQL Customer Support Center https://support.mysql.com/ to alert the MySQL team about the problem and have it fixed as soon as possible.

You can also use the script `mysql_find_rows` to just execute some of the update statements if you want to narrow down the problem.

E.2. Debugging a MySQL Client

To be able to debug a MySQL client with the integrated debug package, you should configure MySQL with `--with-debug` or `--with-debug=full`. See Section 2.8.2, “Typical **configure** Options”.

Before running a client, you should set the `MYSQL_DEBUG` environment variable:

```
shell> MYSQL_DEBUG=d:t:O,/tmp/client.trace
shell> export MYSQL_DEBUG
```

This causes clients to generate a trace file in `/tmp/client.trace`.

If you have problems with your own client code, you should attempt to connect to the server and run your query using a client that is known to work. Do this by running **mysql** in debugging mode (assuming that you have compiled MySQL with debugging on):

```
shell> mysql --debug=d:t:O,/tmp/client.trace
```

This provides useful information in case you mail a bug report. See Section 1.4.1.3, “How to Report Bugs or Problems”.

If your client crashes at some 'legal' looking code, you should check that your `mysql.h` include file matches your MySQL library file. A very common mistake is to use an old `mysql.h` file from an old MySQL installation with new MySQL library.

E.3. The DBUG Package

The MySQL server and most MySQL clients are compiled with the DBUG package originally created by Fred Fish. When you have configured MySQL for debugging, this package makes it possible to get a trace file of what the program is debugging. See Section E.1.2, “Creating Trace Files”.

This section summaries the argument values that you can specify in debug options on the command line for MySQL programs that have been built with debugging support. For more information about programming with the DBUG package, see the DBUG manual in the `dbug` directory of MySQL source distributions. It's best to use a recent distribution for MySQL 5.0 to get the most updated DBUG manual.

You use the debug package by invoking a program with the `--debug="..."` or the `-#...` option.

Most MySQL programs have a default debug string that is used if you don't specify an option to `--debug`. The default trace file is usually `/tmp/program_name.trace` on Unix and `\program_name.trace` on Windows.

The debug control string is a sequence of colon-separated fields as follows:

```
<field_1>:<field_2>:...:<field_N>
```

Each field consists of a mandatory flag character followed by an optional `,` and comma-separated list of modifiers:

```
flag[,modifier,modifier,...,modifier]
```

The currently recognized flag characters are:

Flag	Description
`d`	Enable output from DBUG_<N> macros for the current state. May be followed by a list of keywords which selects output only for the DBUG macros with that keyword. An empty list of keywords implies output for all macros.
`D`	Delay after each debugger output line. The argument is the number of tenths of seconds to delay, subject to machine capabilities. For example, `-#D,20` specifies a delay of two seconds.
`f`	Limit debugging and/or tracing, and profiling to the list of named functions. Note that a null list disables all functions. The appropriate `d` or `t` flags must still be given; this flag only limits their actions if they are enabled.
`F`	Identify the source file name for each line of debug or trace output.
`i`	Identify the process with the PID or thread ID for each line of debug or trace output.
`g`	Enable profiling. Create a file called `dbugmon.out` containing information that can be used to profile the program. May be followed by a list of keywords that select profiling only for the functions in that list. A null list implies that all functions are considered.
`L`	Identify the source file line number for each line of debug or trace output.
`n`	Print the current function nesting depth for each line of debug or trace output.
`N`	Number each line of debug output.
`o`	Redirect the debugger output stream to the specified file. The default output is `stderr`.

O	Like o, but the file is really flushed between each write. When needed, the file is closed and re-opened between each write.
p	Limit debugger actions to specified processes. A process must be identified with the DBUG_PROCESS macro and match one in the list for debugger actions to occur.
P	Print the current process name for each line of debug or trace output.
r	When pushing a new state, do not inherit the previous state's function nesting level. Useful when the output is to start at the left margin.
S	Do function _sanity(_file_,_line_) at each debugged function until _sanity() returns something that differs from 0. (Mostly used with safemalloc to find memory leaks)
t	Enable function call/exit trace lines. May be followed by a list (containing only one modifier) giving a numeric maximum trace level, beyond which no output occurs for either debugging or tracing macros. The default is a compile time option.

Some examples of debug control strings that might appear on a shell command line (the -# is typically used to introduce a control string to an application program) are:

```
-#d:t
-#d:f,main,subr1:F:L:t,20
-#d,input,output,files:n
-#d:t:i:O,\\mysqld.trace
```

In MySQL, common tags to print (with the d option) are enter, exit, error, warning, info, and loop.

E.4. Comments about RTS Threads

I have tried to use the RTS thread packages with MySQL but stumbled on the following problems:

They use old versions of many POSIX calls and it is very tedious to make wrappers for all functions. I am inclined to think that it would be easier to change the thread libraries to the newest POSIX specification.

Some wrappers are currently written. See mysys/my_pthread.c for more info.

At least the following should be changed:

pthread_get_specific should use one argument. sigwait should take two arguments. A lot of functions (at least pthread_cond_wait, pthread_cond_timedwait()) should return the error code on error. Now they return -1 and set errno.

Another problem is that user-level threads use the ALRM signal and this aborts a lot of functions (read, write, open...). MySQL should do a retry on interrupt on all of these but it is not that easy to verify it.

The biggest unsolved problem is the following:

To get thread-level alarms I changed mysys/thr_alarm.c to wait between alarms with pthread_cond_timedwait(), but this aborts with error EINTR. I tried to debug the thread library as to why this happens, but couldn't find any easy solution.

If someone wants to try MySQL with RTS threads I suggest the following:

- Change functions MySQL uses from the thread library to POSIX. This shouldn't take that long.

- Compile all libraries with the `-DHAVE_rts_threads`.
- Compile `thr_alarm`.
- If there are some small differences in the implementation, they may be fixed by changing `my_pthread.h` and `my_pthread.c`.
- Run `thr_alarm`. If it runs without any ``warning," ``error," or aborted messages, you are on the right track. Here is a successful run on Solaris:

```
Main thread: 1
Thread 0 (5) started
Thread: 5  Waiting
process_alarm
Thread 1 (6) started
Thread: 6  Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6  Slept for 1 (1) sec
Thread: 6  Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6  Slept for 2 (2) sec
Thread: 6  Simulation of no alarm needed
Thread: 6  Slept for 0 (3) sec
Thread: 6  Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6  Slept for 4 (4) sec
Thread: 6  Waiting
process_alarm
thread_alarm
Thread: 5  Slept for 10 (10) sec
Thread: 5  Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6  Slept for 5 (5) sec
Thread: 6  Waiting
process_alarm
process_alarm

...
thread_alarm
Thread: 5  Slept for 0 (1) sec
end
```

E.5. Differences Between Thread Packages

MySQL is very dependent on the thread package used. So when choosing a good platform for MySQL, the thread package is very important.

There are at least three types of thread packages:

- User threads in a single process. Thread switching is managed with alarms and the threads library

manages all non-thread-safe functions with locks. Read, write and select operations are usually managed with a thread-specific select that switches to another thread if the running threads have to wait for data. If the user thread packages are integrated in the standard libs (FreeBSD and BSDI threads) the thread package requires less overhead than thread packages that have to map all unsafe calls (MIT-pthreads, FSU Pthreads and RTS threads). In some environments (for example, SCO), all system calls are thread-safe so the mapping can be done very easily (FSU Pthreads on SCO). Downside: All mapped calls take a little time and it's quite tricky to be able to handle all situations. There are usually also some system calls that are not handled by the thread package (like MIT-pthreads and sockets). Thread scheduling isn't always optimal.

- User threads in separate processes. Thread switching is done by the kernel and all data are shared between threads. The thread package manages the standard thread calls to allow sharing data between threads. LinuxThreads is using this method. Downside: Lots of processes. Thread creating is slow. If one thread dies the rest are usually left hanging and you must kill them all before restarting. Thread switching is somewhat expensive.

- Kernel threads. Thread switching is handled by the thread library or the kernel and is very fast. Everything is done in one process, but on some systems, **ps** may show the different threads. If one thread aborts, the whole process aborts. Most system calls are thread-safe and should require very little overhead. Solaris, HP-UX, AIX and OSF/1 have kernel threads.

In some systems kernel threads are managed by integrating user level threads in the system libraries. In such cases, the thread switching can only be done by the thread library and the kernel isn't really ``thread aware."

Appendix F. Environment Variables

This appendix lists all the environment variables that are used directly or indirectly by MySQL. Most of these can also be found in other places in this manual.

Note that any options on the command line take precedence over values specified in option files and environment variables, and values in option files take precedence over values in environment variables.

In many cases, it's preferable to use an option file instead of environment variables to modify the behavior of MySQL. See Section 4.3.2, “Using Option Files”.

Variable	Description
`CXX`	The name of your C++ compiler (for running **configure**).
`CC`	The name of your C compiler (for running **configure**).
`CFLAGS`	Flags for your C compiler (for running **configure**).
`CXXFLAGS`	Flags for your C++ compiler (for running **configure**).
`DBI_USER`	The default username for Perl DBI.
`DBI_TRACE`	Trace options for Perl DBI.
`HOME`	The default path for the **mysql** history file is `$HOME/.mysql_history`.
`LD_RUN_PATH`	Used to specify where your `libmysqlclient.so` is located.
`MYSQL_DEBUG`	Debug trace options when debugging.
`MYSQL_HISTFILE`	The path to the **mysql** history file. If this variable is set, its value overrides the default of `$HOME/.mysql_history`.
`MYSQL_HOST`	The default hostname used by the **mysql** command-line client.
`MYSQL_PS1`	The command prompt to use in the **mysql** command-line client.
`MYSQL_PWD`	The default password when connecting to **mysqld**. Note that use of this is insecure! See Section 5.6.6, “Keeping Your Password Secure”.
`MYSQL_TCP_PORT`	The default TCP/IP port number.
`MYSQL_UNIX_PORT`	The default Unix socket filename; used for connections to `localhost`.
`PATH`	Used by the shell to find MySQL programs.
`TMPDIR`	The directory where temporary files are created.
`TZ`	This should be set to your local time zone. See Section A.4.6, “Time Zone Problems”.
`UMASK_DIR`	The user-directory creation mask when creating directories. Note that this is ANDed with `UMASK`!
`UMASK`	The user-file creation mask when creating files.
`USER`	The default username on Windows and NetWare to use when connecting to **mysqld**.

Appendix G. MySQL Regular Expressions

A regular expression is a powerful way of specifying a pattern for a complex search.

MySQL uses Henry Spencer's implementation of regular expressions, which is aimed at conformance with POSIX 1003.2. See Appendix B, *Credits*. MySQL uses the extended version to support pattern-matching operations performed with the `REGEXP` operator in SQL statements. See Section 3.3.4.7, “Pattern Matching”.

This appendix is a summary, with examples, of the special characters and constructs that can be used in MySQL for `REGEXP` operations. It does not contain all the details that can be found in Henry Spencer's `regex(7)` manual page. That manual page is included in MySQL source distributions, in the `regex.7` file under the `regex` directory.

A regular expression describes a set of strings. The simplest regular expression is one that has no special characters in it. For example, the regular expression `hello` matches `hello` and nothing else.

Non-trivial regular expressions use certain special constructs so that they can match more than one string. For example, the regular expression `hello|word` matches either the string `hello` or the string `word`.

As a more complex example, the regular expression `B[an]*s` matches any of the strings `Bananas`, `Baaaaas`, `Bs`, and any other string starting with a `B`, ending with an `s`, and containing any number of `a` or `n` characters in between.

A regular expression for the `REGEXP` operator may use any of the following special characters and constructs:

- `^`

  Match the beginning of a string.

  ```
  mysql> SELECT 'fo\nfo' REGEXP '^fo$';                    -> 0
  mysql> SELECT 'fofo' REGEXP '^fo';                       -> 1
  ```

- `$`

  Match the end of a string.

  ```
  mysql> SELECT 'fo\no' REGEXP '^fo\no$';                  -> 1
  mysql> SELECT 'fo\no' REGEXP '^fo$';                     -> 0
  ```

- `.`

  Match any character (including carriage return and newline).

  ```
  mysql> SELECT 'fofo' REGEXP '^f.*$';                     -> 1
  mysql> SELECT 'fo\r\nfo' REGEXP '^f.*$';                 -> 1
  ```

- `a*`

  Match any sequence of zero or more `a` characters.

```
mysql> SELECT 'Ban' REGEXP '^Ba*n';                    -> 1
mysql> SELECT 'Baaan' REGEXP '^Ba*n';                  -> 1
mysql> SELECT 'Bn' REGEXP '^Ba*n';                     -> 1
```

- `a+`

  Match any sequence of one or more `a` characters.

```
mysql> SELECT 'Ban' REGEXP '^Ba+n';                    -> 1
mysql> SELECT 'Bn' REGEXP '^Ba+n';                     -> 0
```

- `a?`

  Match either zero or one `a` character.

```
mysql> SELECT 'Bn' REGEXP '^Ba?n';                     -> 1
mysql> SELECT 'Ban' REGEXP '^Ba?n';                    -> 1
mysql> SELECT 'Baan' REGEXP '^Ba?n';                   -> 0
```

- `de|abc`

  Match either of the sequences `de` or `abc`.

```
mysql> SELECT 'pi' REGEXP 'pi|apa';                    -> 1
mysql> SELECT 'axe' REGEXP 'pi|apa';                   -> 0
mysql> SELECT 'apa' REGEXP 'pi|apa';                   -> 1
mysql> SELECT 'apa' REGEXP '^(pi|apa)$';               -> 1
mysql> SELECT 'pi' REGEXP '^(pi|apa)$';                -> 1
mysql> SELECT 'pix' REGEXP '^(pi|apa)$';               -> 0
```

- `(abc)*`

  Match zero or more instances of the sequence `abc`.

```
mysql> SELECT 'pi' REGEXP '^(pi)*$';                   -> 1
mysql> SELECT 'pip' REGEXP '^(pi)*$';                  -> 0
mysql> SELECT 'pipi' REGEXP '^(pi)*$';                 -> 1
```

- `{1},{2,3}`

  `{n}` or `{m,n}` notation provides a more general way of writing regular expressions that match many occurrences of the previous atom (or ``piece'') of the pattern. `m` and `n` are integers.

  - `a*`

    Can be written as `a{0,}`.

  - `a+`

    Can be written as `a{1,}`.

  - `a?`

    Can be written as `a{0,1}`.

  To be more precise, `a{n}` matches exactly `n` instances of `a`. `a{n,}` matches `n` or more instances of

a. a{m,n} matches m through n instances of a, inclusive.

m and n must be in the range from 0 to RE_DUP_MAX (default 255), inclusive. If both m and n are given, m must be less than or equal to n.

```
mysql> SELECT 'abcde' REGEXP 'a[bcd]{2}e';              -> 0
mysql> SELECT 'abcde' REGEXP 'a[bcd]{3}e';              -> 1
mysql> SELECT 'abcde' REGEXP 'a[bcd]{1,10}e';           -> 1
```

- [a-dX], [^a-dX]

  Matches any character that is (or is not, if ^ is used) either a, b, c, d or X. A - character between two other characters forms a range that matches all characters from the first character to the second. For example, [0-9] matches any decimal digit. To include a literal] character, it must immediately follow the opening bracket [. To include a literal - character, it must be written first or last. Any character that does not have a defined special meaning inside a [] pair matches only itself.

```
mysql> SELECT 'aXbc' REGEXP '[a-dXYZ]';                 -> 1
mysql> SELECT 'aXbc' REGEXP '^[a-dXYZ]$';               -> 0
mysql> SELECT 'aXbc' REGEXP '^[a-dXYZ]+$';              -> 1
mysql> SELECT 'aXbc' REGEXP '^[^a-dXYZ]+$';             -> 0
mysql> SELECT 'gheis' REGEXP '^[^a-dXYZ]+$';            -> 1
mysql> SELECT 'gheisa' REGEXP '^[^a-dXYZ]+$';           -> 0
```

- [.characters.]

  Within a bracket expression (written using [and]), matches the sequence of characters of that collating element. characters is either a single character or a character name like newline. You can find the full list of character names in the regexp/cname.h file.

```
mysql> SELECT '~' REGEXP '[[.~.]]';                     -> 1
mysql> SELECT '~' REGEXP '[[.tilde.]]';                 -> 1
```

- [=character_class=]

  Within a bracket expression (written using [and]), [=character_class=] represents an equivalence class. It matches all characters with the same collation value, including itself. For example, if o and (+) are the members of an equivalence class, then [[=o=]], [[=(+)=]], and [o(+)] are all synonymous. An equivalence class may not be used as an endpoint of a range.

- [:character_class:]

  Within a bracket expression (written using [and]), [:character_class:] represents a character class that matches all characters belonging to that class. The standard class names are:

alnum	Alphanumeric characters
alpha	Alphabetic characters
blank	Whitespace characters
cntrl	Control characters

digit	Digit characters
graph	Graphic characters
lower	Lowercase alphabetic characters
print	Graphic or space characters
punct	Punctuation characters
space	Space, tab, newline, and carriage return
upper	Uppercase alphabetic characters
xdigit	Hexadecimal digit characters

These stand for the character classes defined in the `ctype(3)` manual page. A particular locale may provide other class names. A character class may not be used as an endpoint of a range.

```
mysql> SELECT 'justalnums' REGEXP '[[:alnum:]]+';       -> 1
mysql> SELECT '!!' REGEXP '[[:alnum:]]+';               -> 0
```

- `[[:<:]]`, `[[:>:]]`

  These markers stand for word boundaries. They match the beginning and end of words, respectively. A word is a sequence of word characters that is not preceded by or followed by word characters. A word character is an alphanumeric character in the `alnum` class or an underscore (_).

```
mysql> SELECT 'a word a' REGEXP '[[:<:]]word[[:>:]]';   -> 1
mysql> SELECT 'a xword a' REGEXP '[[:<:]]word[[:>:]]';  -> 0
```

To use a literal instance of a special character in a regular expression, precede it by two backslash (\) characters. The MySQL parser interprets one of the backslashes, and the regular expression library interprets the other. For example, to match the string `1+2` that contains the special `+` character, only the last of the following regular expressions is the correct one:

```
mysql> SELECT '1+2' REGEXP '1+2';                       -> 0
mysql> SELECT '1+2' REGEXP '1\+2';                      -> 0
mysql> SELECT '1+2' REGEXP '1\\+2';                     -> 1
```

Appendix H. GNU General Public License

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software---to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

1. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The ``Program",

below, refers to any such program or work, and a ``work based on the Program'' means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term ``modification''.) Each licensee is addressed as ``you''.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

2. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

   b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

   c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

   In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

4. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b. Accompany it with a written offer, valid for at least three years, to give any third-party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

5. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

6. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

7. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

8. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other

circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

9. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

10. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and ``any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

11. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

    NO WARRANTY

12. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM ``AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

13. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the ``copyright'' line and a pointer to where the full notice is found.

```
one line to give the program's name and a brief idea of what it does.
Copyright (C) yyyy  name of author

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA  02111-1307, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
This is free software, and you are welcome to redistribute it
under certain conditions; type `show c' for details.
```

The hypothetical commands '`show w`' and '`show c`' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than '`show w`' and '`show c`'; they could even be mouse-clicks or menu items---whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a ``copyright disclaimer'' for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
`Gnomovision' (which makes passes at compilers) written by James Hacker.

signature of Ty Coon, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Appendix I. MySQL FLOSS License Exception

Version 0.3, 10 February 2005

The MySQL AB Exception for Free/Libre and Open Source Software-only Applications Using MySQL Client Libraries (the ``FLOSS Exception'').

Exception Intent

We want specified Free/Libre and Open Source Software (``FLOSS'') applications to be able to use specified GPL-licensed MySQL client libraries (the ``Program'') despite the fact that not all FLOSS licenses are compatible with version 2 of the GNU General Public License (the ``GPL'').

Legal Terms and Conditions

As a special exception to the terms and conditions of version 2.0 of the GPL:

1. You are free to distribute a Derivative Work that is formed entirely from the Program and one or more works (each, a ``FLOSS Work'') licensed under one or more of the licenses listed below in section 1, as long as:

   a. You obey the GPL in all respects for the Program and the Derivative Work, except for identifiable sections of the Derivative Work which are not derived from the Program, and which can reasonably be considered independent and separate works in themselves,

   b. all identifiable sections of the Derivative Work which are not derived from the Program, and which can reasonably be considered independent and separate works in themselves,

      - i

        are distributed subject to one of the FLOSS licenses listed below, and

      - ii

        the object code or executable form of those sections are accompanied by the complete corresponding machine-readable source code for those sections on the same medium and under the same FLOSS license as the corresponding object code or executable forms of those sections, and

   c. any works which are aggregated with the Program or with a Derivative Work on a volume of a storage or distribution medium in accordance with the GPL, can reasonably be considered independent and separate works in themselves which are not derivatives of either the Program, a Derivative Work or a FLOSS Work.

   If the above conditions are not met, then the Program may only be copied, modified, distributed or used under the terms and conditions of the GPL or another valid licensing option from MySQL AB.

2. FLOSS License List

License name	Version(s)/Copyright Date
Academic Free License	2.0
Apache Software License	1.0/1.1/2.0
Apple Public Source License	2.0

Artistic license	From Perl 5.8.0
BSD license	"July 22 1999"
Common Public License	1.0
GNU Library or "Lesser" General Public License (LGPL)	2.0/2.1
Jabber Open Source License	1.0
MIT license	-
Mozilla Public License (MPL)	1.0/1.1
Open Software License	2.0
OpenSSL license (with original SSLeay license)	"2003" ("1998")
PHP License	3.0
Python license (CNRI Python License)	-
Python Software Foundation License	2.1.1
Sleepycat License	"1999"
W3C License	"2001"
X11 License	"2001"
Zlib/libpng License	-
Zope Public License	2.0

Due to the many variants of some of the above licenses, we require that any version follow the 2003 version of the Free Software Foundation's Free Software Definition (http://www.gnu.org/philosophy/free-sw.html) or version 1.9 of the Open Source Definition by the Open Source Initiative (http://www.opensource.org/docs/definition.php).

3. Definitions

   a. Terms used, but not defined, herein shall have the meaning provided in the GPL.

   b. Derivative Work means a derivative work under copyright law.

4. Applicability This FLOSS Exception applies to all Programs that contain a notice placed by MySQL AB saying that the Program may be distributed under the terms of this FLOSS Exception. If you create or distribute a work which is a Derivative Work of both the Program and any other work licensed under the GPL, then this FLOSS Exception is not available for that work; thus, you must remove the FLOSS Exception notice from that work and comply with the GPL in all respects, including by retaining all GPL notices. You may choose to redistribute a copy of the Program exclusively under the terms of the GPL by removing the FLOSS Exception notice from that copy of the Program, provided that the copy has never been modified by you or any third party.

Index

Symbols

! (logical NOT), 620
!= (not equal), 615
", 542
% (modulo), 645
% (wildcard character), 539
& (bitwise AND), 681
&& (logical AND), 621
() (parentheses), 614
(Control-Z) \Z, 539
* (multiplication), 640
+ (addition), 640
- (subtraction), 640
- (unary minus), 640
--password option, 334
--with-raid link errors, 105
-p option, 334
.my.cnf file, 221, 223, 310, 318, 335, 387
.mysql_history file, 502
.pid (process ID) file, 360
/ (division), 641
/etc/passwd, 299, 727
< (less than), 616
<<, 211
<< (left shift), 682
<= (less than or equal), 616
<=> (equal to), 615
<> (not equal), 615
= (equal), 615
> (greater than), 616
>= (greater than or equal), 616
>> (right shift), 682
\" (double quote), 538
\' (single quote), 538
\0 (ASCII 0), 538
\b (backspace), 538
\n (linefeed), 539
\n (newline), 539
\r (carriage return), 539
\t (tab), 539
\Z (Control-Z) ASCII 26, 539
\\ (escape), 539
^ (bitwise XOR), 681
_ (wildcard character), 539
`, 542
| (bitwise OR), 681
|| (logical OR), 621
~, 682

A

aborted clients, 1278
aborted connection, 1278
ABS(), 641
access control, 310
access denied errors, 1271
access privileges, 301
Access program, 1186
account privileges
    adding, 328
accounts
    anonymous user, 125
    root, 125
ACID, 29, 860
ACLs, 301
ACOS(), 641
ActiveState Perl, 177
ADDDATE(), 650
adding
    character sets, 369
    native functions, 1267
    new account privileges, 328
    new functions, 1256
    new user privileges, 328
    new users, 94, 98
    procedures, 1268
    user-defined functions, 1258
addition (+), 640
ADDTIME(), 650
administration
    server, 508
ADO program, 1187
AES_DECRYPT(), 683
AES_ENCRYPT(), 683
age
    calculating, 194
alias, 1294
alias names
    case sensitivity, 544
aliases
    for expressions, 703
    for tables, 725
    in GROUP BY clauses, 703
    in ORDER BY clauses, 703
    names, 542
    on expressions, 724
ALL, 728
ALLOW_INVALID_DATES SQL mode, 250
ALTER COLUMN, 748
ALTER DATABASE, 746
ALTER FUNCTION, 1031
ALTER PROCEDURE, 1031
ALTER SCHEMA, 746
ALTER TABLE, 746, 748, 1299
ALTER VIEW, 751
altering
    database, 746
    schema, 746
ANALYZE TABLE, 789
AND
    bitwise, 681
    logical, 621

anonymous user, 125, 125, 311, 314
ANSI mode
    running, 24
ANSI SQL mode, 250, 254
ANSI_QUOTES SQL mode, 251
answering questions
    etiquette, 22
Apache, 215
API's
    list of, 1314
APIs, 1060
    Perl, 1154
ARCHIVE storage engine, 834, 857
ARCHIVE table type, 834, 857
Area(), 1017, 1018
argument processing, 1262
arithmetic expressions, 640
arithmetic functions, 681
AS, 724, 729
AsBinary(), 1011
ASCII(), 624
ASIN(), 642
AsText(), 1011
ATAN(), 642
ATAN2(), 642
attackers
    security against, 297
AUTO-INCREMENT
    ODBC, 1199
AUTO_INCREMENT, 211
    and NULL values, 1294
AVG(), 696
AVG(DISTINCT), 696

B

backing up
    databases, 523, 530
backslash
    escape character, 538
backspace (\b), 538
BACKUP TABLE, 789
backups, 343
    database, 789
batch
    mysql option, 498
batch mode, 204
BDB storage engine, 834, 848
BDB table type, 834, 848
BDB tables, 29
BdMPolyFromText(), 1006
BdMPolyFromWKB(), 1007
BdPolyFromText(), 1006
BdPolyFromWKB(), 1007
BEGIN, 774, 1033
benchmark suite, 428
BENCHMARK(), 686
benchmarks, 429
BerkeleyDB storage engine, 834, 848
BerkeleyDB table type, 834, 848
BETWEEN ... AND, 617
Big5 Chinese character encoding, 1292
BIGINT, 585
BIN(), 624
BINARY, 590, 678
binary distributions, 50
    installing, 91
    on Linux, 148
binary log, 376
BIT, 584
BitKeeper tree, 101
BIT_AND(), 696
BIT_COUNT, 211
BIT_COUNT(), 682
bit_functions
    example, 211
BIT_LENGTH(), 625
BIT_OR, 211
BIT_OR(), 696
BIT_XOR(), 697
BLOB, 591, 605
    inserting binary data, 540
    size, 611
BLOB columns
    default values, 606
    indexing, 466, 758
blocking_queries
    mysqlcc option, 518
BOOL, 585
BOOLEAN, 585
Borland Builder 4 program, 1187
Borland C++ compiler, 1155
Boundary(), 1013
brackets
    square, 584
buffer sizes
    client, 1060
    mysqld server, 478
Buffer(), 1020
bug reports
    criteria for, 20
bugs
    known, 36
    reporting, 18
bugs database, 18
bugs.mysql.com, 18
building
    client programs, 1147

C

C API
    data types, 1062
    functions, 1065
    linking problems, 1147
C Prepared statement API
    functions, 1115
C++ APIs, 1154

C++ Builder, 1190
C++ compiler
  gcc, 99
C++ compiler cannot create executables, 105
C:\my.cnf file, 387
CACHE INDEX, 816
caches
  clearing, 817
calculating
  dates, 194
CALL, 1032
calling sequences for aggregate functions
  UDF, 1261
calling sequences for simple functions
  UDF, 1259
can't create/write to file, 1280
carriage return (\r), 539
CASE, 622, 1037
case sensitivity
  in identifiers, 544
  in names, 544
  in searches, 1292
  in string comparisons, 637
case-sensitivity
  in access checking, 305
  of database names, 25
  of table names, 25
CAST, 679
cast functions, 678
cast operators, 678
casts, 614, 678
CC environment variable, 99, 99, 106, 1573
cc1plus problems, 105
CEILING(), 642
Centroid(), 1018
CFLAGS environment variable, 99, 106, 1573
CHANGE MASTER TO, 823
ChangeLog, 1323
changes
  log, 1323
  version 3.19, 1527
  version 3.20, 1520
  version 3.21, 1504
  version 3.22, 1489
  version 3.23, 1441
  version 4.0, 1386
  version 4.1, 1342
  version 5.0, 1323
changes to privileges, 316
changing
  column, 748
  column order, 1300
  field, 748
  table, 746, 748, 1299
changing socket location, 99, 122, 1291
CHAR, 589, 603
CHAR VARYING, 590
CHAR(), 625
CHARACTER, 589
character sets, 100, 367
  adding, 369
Character sets, 554
CHARACTER VARYING, 590
character-sets-dir
  mysql option, 498
characters
  multi-byte, 372
CHARACTER_LENGTH(), 625
CHARACTER_SETS
  INFORMATION_SCHEMA table, 1052
CHARSET(), 687
CHAR_LENGTH(), 625
check options
  myisamchk, 352
CHECK TABLE, 790
checking
  tables for errors, 357
checksum errors, 155
CHECKSUM TABLE, 791
Chinese, 1292
choosing
  a MySQL version, 46
choosing types, 611
clearing
  caches, 817
client programs
  building, 1147
client tools, 1060
clients
  debugging, 1568
  threaded, 1147
CLOSE, 1036
closing
  tables, 475
COALESCE(), 618
COERCIBILITY(), 687
ColdFusion program, 1187
collating
  strings, 371
COLLATION(), 688
COLLATIONS
  INFORMATION_SCHEMA table, 1053
COLLATION_CHARACTER_SET_APPLICABILITY
  INFORMATION_SCHEMA table, 1053
column
  changing, 748
column comments, 757
column names
  case sensitivity, 544
columns
  changing, 1300
  displaying, 534
  indexes, 466
  names, 542
  other types, 611
  selecting, 191

storage requirements, 610
types, 584
COLUMNS
INFORMATION_SCHEMA table, 1048
COLUMN_PRIVILEGES
INFORMATION_SCHEMA table, 1051
command syntax, 4
command-line history
mysql, 502
command-line options, 239
mysql, 498
mysqladmin, 512
mysqlcc, 518
command-line tool, 497
commands
for binary distribution, 92
replication masters, 821
replication slaves, 823
commands out of sync, 1280
Comment syntax, 549
comments
adding, 549
starting, 33
COMMIT, 29, 774
comparison operators, 614
compatibility
between MySQL versions, 129, 131, 137, 141, 142
with mSQL, 639
with ODBC, 544, 586, 614, 617, 729, 756, 1512
with Oracle, 26, 699, 773
with PostgreSQL, 27
with standard SQL, 23
with Sybase, 774
compiler
C++ gcc, 99
compiling
on Windows, 113
optimizing, 477
problems, 104
speed, 481
statically, 99
user-defined functions, 1264
compliance
Y2K, 10
compress
mysql option, 498
mysqlcc option, 518
COMPRESS(), 625
compressed tables, 491, 841
CONCAT(), 626
CONCAT_WS(), 626
concurrent inserts, 462
Conditions, 1034
config-file
mysqld_multi option, 235
config.cache, 104
config.cache file, 104
configuration files, 318
configuration options, 98
configure
running after prior invocation, 104
configure option
--with-charset, 100
--with-collation, 100
--with-extra-charsets, 100
--with-low-memory, 105
configure script, 98
connecting
remotely with SSH, 342
to the server, 181, 309
verification, 310
connection
aborted, 1278
CONNECTION_ID(), 688
connection_name
mysqlcc option, 518
Connector/J, 1219
Connector/JDBC, 1156
Connector/ODBC, 1156, 1156
reporting problems, 1158
Connectors
MySQL, 1156
connect_timeout variable, 502, 513, 519
constant table, 432, 441
constraints, 34
CONSTRAINTS
INFORMATION_SCHEMA table, 1054
Contains(), 1022
contributing companies
list of, 1315
contributors
list of, 1307
control access, 310
control flow functions, 622
CONV(), 626
conventions
typographical, 2
CONVERT, 679
CONVERT TO, 749
CONVERT_TZ(), 650
ConvexHull(), 1020
copying databases, 145
copying tables, 762
COS(), 643
COT(), 643
COUNT(), 697
COUNT(DISTINCT), 697
counting
table rows, 199
crash, 1563
recovery, 356
repeated, 1287
crash-me, 429
crash-me program, 427, 428
CRC32(), 643
CREATE DATABASE, 751

CREATE FUNCTION, 1029, 1257
CREATE INDEX, 752
CREATE PROCEDURE, 1029
CREATE SCHEMA, 751
CREATE TABLE, 753
CREATE USER, 779
CREATE VIEW, 765
creating
  bug reports, 18
  database, 751
  databases, 185
  default startup options, 220
  schema, 751
  tables, 187
creating user accounts, 779
CROSS JOIN, 729
Crosses(), 1022
CR_SERVER_GONE_ERROR, 1276
CR_SERVER_LOST_ERROR, 1276
CSV storage engine, 834, 857
CSV table type, 834, 857
CURDATE(), 651
CURRENT_DATE, 651
CURRENT_TIME, 651
CURRENT_TIMESTAMP, 652
CURRENT_USER(), 688
Cursors, 1035
CURTIME(), 651
customer support
  mailing address, 22
customers
  of MySQL, 428
CXX environment variable, 99, 99, 105, 105, 106, 1573
CXXFLAGS environment variable, 99, 106, 1573

D

data
  character sets, 367
  importing, 532
  loading into tables, 188
  retrieving, 189
  size, 465
data types, 584
  C API, 1062
database
  altering, 746
  creating, 751
  deleting, 770
  mysql option, 498
  mysqlcc option, 518
database design, 465
Database information
  obtaining, 799
database metadata, 1044
database names
  case sensitivity, 544
  case-sensitivity, 25
DATABASE(), 689
databases
  backups, 343
  copying, 145
  creating, 185
  defined, 4
  displaying, 534
  dumping, 523, 530
  information about, 203
  names, 542
  replicating, 393
  selecting, 186
  symbolic links, 485
  using, 185
DataJunction, 1188
DATE, 587, 595, 1292
date and time functions, 649
Date and Time types, 594
date calculations, 194
DATE columns
  problems, 1292
date functions
  Y2K compliance, 10
date types, 610
  Y2K issues, 603
date values
  problems, 597
DATE(), 652
DATEDIFF(), 652
DATETIME, 587, 595
DATE_ADD(), 652
DATE_FORMAT(), 654
DATE_SUB(), 652
DAY(), 656
DAYNAME(), 656
DAYOFMONTH(), 656
DAYOFWEEK(), 656
DAYOFYEAR(), 656
db table
  sorting, 314
DB2 SQL mode, 254
DBI interface, 1154
DBI->quote, 540
DBI->trace, 1566
DBI/DBD interface, 1154
DBI_TRACE environment variable, 1566, 1573
DBI_USER environment variable, 1573
DBUG package, 1569
DEALLOCATE PREPARE, 831
debug
  mysql option, 498
debug-info
  mysql option, 498
debugging
  client, 1568
  server, 1563
DEC, 587
DECIMAL, 587
decimal point, 584

DECLARE, 1033
DECODE(), 683
decode_bits myisamchk variable, 351
DEFAULT
  constraint, 35
default
  privileges, 125
default hostname, 309
default installation location, 60
default options, 220
DEFAULT value clause, 756
default values, 426, 709, 756
  BLOB and TEXT columns, 606
  implicit, 756
  suppression, 35
DEFAULT(), 693
default-character-set
  mysql option, 498
defaults
  embedded, 1150
DEGREES(), 643
DELAYED, 713
delayed_insert_limit, 714
DELETE, 704
deleting
  database, 770
  foreign key, 749, 877
  function, 1257
  index, 748, 771
  primary key, 748
  rows, 1295
  schema, 770
  table, 771
  user, 331, 779
  users, 331, 779
deletion
  mysql.sock, 1291
Delphi program, 1189
derived tables, 738
DESC, 772
DESCRIBE, 203, 772
design
  choices, 465
  issues, 36
  limitations, 426
DES_DECRYPT(), 683
DES_ENCRYPT(), 684
developers
  list of, 1302
development source tree, 101
Difference(), 1020
digits, 584
Dimension(), 1012
directory structure
  default, 60
DISCARD TABLESPACE, 750, 881
disconnecting
  from the server, 181
Disjoint(), 1022
disk full, 1289
disk issues, 484
disks
  splitting data across, 487
display size, 584
display width, 584
displaying
  database information, 534
  information
    Cardinality, 804
    Collation, 804
    SHOW, 799, 800, 804, 813
  table status, 811
Distance(), 1022
DISTINCT, 192, 449, 696, 697, 699, 699, 728
DISTINCTROW, 728
DIV, 641
division (/), 641
DNS, 483
DO, 707
Documenters
  list of, 1311
DOUBLE, 586
DOUBLE PRECISION, 586
double quote (\"), 538
DOUBLE(M
  D), 586
downgrading, 146
downloading, 57
DROP DATABASE, 770
DROP FOREIGN KEY, 749, 877
DROP FUNCTION, 1031, 1257
DROP INDEX, 748, 771
DROP PRIMARY KEY, 748
DROP PROCEDURE, 1031
DROP SCHEMA, 770
DROP TABLE, 771
DROP USER, 779
DROP VIEW, 771
dropping
  user, 331, 779
DUMPFILE, 728
dumping
  databases, 523, 530
dynamic table characteristics, 840

E

Eiffel Wrapper, 1155
ELT(), 627
email lists, 16
embedded MySQL server library, 1149
ENCODE(), 683
ENCRYPT(), 685
encryption functions, 683
END, 1033
EndPoint(), 1014
entering

queries, 182
ENUM, 591, 607
size, 611
Envelope(), 1012
environment variable
CC, 99, 99, 106
CFLAGS, 99, 106
CXX, 99, 99, 106
CXXFLAGS, 99, 106
HOME, 502
LD_RUN_PATH, 150, 157
MYSQL_DEBUG, 490
MYSQL_HISTFILE, 502
MYSQL_HOST, 310
MYSQL_PWD, 310, 490
MYSQL_TCP_PORT, 387, 387, 490
MYSQL_UNIX_PORT, 387, 387, 490
PATH, 93, 218
USER, 310
Environment variable
CC, 1573
CFLAGS, 1573
CXX, 105, 1573
CXXFLAGS, 1573
DBI_TRACE, 1566, 1573
DBI_USER, 1573
HOME, 1573
LD_LIBRARY_PATH, 178
LD_RUN_PATH, 178, 1573
MYSQL_DEBUG, 1568, 1573
MYSQL_HISTFILE, 1573
MYSQL_HOST, 1573
MYSQL_PS1, 1573
MYSQL_PWD, 1573
MYSQL_TCP_PORT, 1573
MYSQL_UNIX_PORT, 120, 1573
PATH, 1573
TMPDIR, 120, 1573
TZ, 1291, 1573
UMASK, 1285, 1573
UMASK_DIR, 1285, 1573
USER, 1573
Environment variables
CXX, 105
environment variables, 224, 318, 490
list of, 1573
equal (=), 615
Equals(), 1022
Errcode, 536
errno, 536
error messages
can't find file, 1285
displaying, 536
languages, 369
errors
access denied, 1271
checking tables for, 357
common, 1270
directory checksum, 155
handling for UDFs, 1263
known, 36
linking, 1283
list of, 1271
reporting, 1, 16, 18
ERROR_FOR_DIVISION_BY_ZERO SQL mode, 251
escape (\\), 539
escape characters, 538
estimating
query performance, 439
example
mysqld_multi option, 236
EXAMPLE storage engine, 834, 854
EXAMPLE table type, 834, 854
examples
compressed tables, 492
myisamchk output, 362
queries, 205
Excel, 1188
execute
mysql option, 498
EXECUTE, 831
EXP(), 644
EXPLAIN, 430
EXPORT_SET(), 627
expression aliases, 703, 724
expressions
extended, 197
extensions
to standard SQL, 23
ExteriorRing(), 1017
EXTRACT(), 656
extracting
dates, 194

F

FALSE, 540
testing for, 617
fatal signal 11, 105
features of MySQL, 5
FEDERATED storage engine, 834, 854
FEDERATED table type, 834, 854
FETCH, 1036
field
changing, 748
FIELD(), 627
FILE, 629
files
binary log, 376
config.cache, 104
error messages, 369
log, 98, 380
my.cnf, 405
not found message, 1285
permissions, 1285
query log, 375
repairing, 353

script, 204
size limits, 9
slow query log, 379
text, 532
tmp, 119
update log, 375
filesort optimization, 452
FIND_IN_SET(), 628
FIXED, 587
FLOAT, 586, 586
FLOAT(M
D), 586
FLOAT(p), 586, 586, 586
floating-point number, 586
floats, 540
FLOOR(), 644
FLUSH, 817
flush tables, 511
FOR UPDATE, 728
force
mysql option, 498
FORCE INDEX, 724, 730, 1299
FORCE KEY, 724, 730
foreign key
constraint, 34
deleting, 749, 877
foreign keys, 31, 209, 749
FORMAT(), 693
Forums, 23
FOUND_ROWS(), 689
FreeBSD troubleshooting, 106
FROM, 724
FROM_DAYS(), 657
FROM_UNIXTIME(), 657
ft_max_word_len myisamchk variable, 351
ft_min_word_len myisamchk variable, 351
ft_stopword_file myisamchk variable, 351
full disk, 1289
full-text search, 670
FULLTEXT, 670
function
deleting, 1257
functions, 613
arithmetic, 681
bit, 681
C API, 1065
C Prepared statement API, 1115
cast, 678
control flow, 622
date and time, 649
encryption, 683
GROUP BY, 696
grouping, 614
information, 686
mathematical, 641
miscellaneous, 693
native
adding, 1267
new, 1256
string, 624
string comparison, 637
user-defined, 1256
adding, 1258
Functions
user-defined, 1257
functions for SELECT and WHERE clauses, 613

G

gcc, 99
gdb
using, 1564
general information, 1
General Public License, 4
geographic feature, 995
GeomCollFromText(), 1005
GeomCollFromWKB(), 1006
geometry, 995
GEOMETRY, 1004
GEOMETRYCOLLECTION, 1004
GeometryCollection(), 1007
GeometryCollectionFromText(), 1005
GeometryCollectionFromWKB(), 1006
GeometryFromText(), 1005
GeometryFromWKB(), 1006
GeometryN(), 1018
GeometryType(), 1012
GeomFromText(), 1005, 1011
GeomFromWKB(), 1006, 1011
geospatial feature, 995
getting MySQL, 57
GET_FORMAT(), 658
GET_LOCK(), 693
GIS, 995, 995
GLength(), 1014, 1016
global privileges, 780
goals of MySQL, 5
GPL
General Public License, 1578
GNU General Public License, 1578
MySQL FLOSS License Exception, 1583
GRANT, 780
GRANT statement, 328, 340
grant tables, 316
re-creating, 115
sorting, 313, 314
upgrading, 144
granting
privileges, 780
GRANTS, 803
graphical tool, 517
greater than (>), 616
greater than or equal (>=), 616
GREATEST(), 618
GROUP BY, 453
aliases in, 703
extensions to standard SQL, 703, 725

GROUP BY functions, 696
grouping
  expressions, 614
GROUP_CONCAT(), 698
GUI tool, 517

H

HANDLER, 707
Handlers, 1034
handling
  errors, 1263
HAVING, 726
HEAP storage engine, 834, 846
HEAP table type, 834, 846
help
  mysql option, 498
  mysqlcc option, 518
  mysqld_multi option, 236
HEX(), 628
hexadecimal values, 540
HIGH_NOT_PRECEDENCE SQL mode, 251
HIGH_PRIORITY, 728
hints, 25, 728, 730, 730
  index, 724, 730
history of MySQL, 5
history_size
  mysqlcc option, 518
HOME environment variable, 502, 1573
host
  mysql option, 499
  mysqlcc option, 518
host table, 316
  sorting, 314
host.frm
  problems finding, 116
hostname
  default, 309
hostname caching, 483
HOUR(), 659
html
  mysql option, 499

I

ID
  unique, 1146
identifiers, 542
  case sensitivity, 544
  quoting, 542
IF, 1037
IF(), 622
IFNULL(), 623
IGNORE INDEX, 724, 730
IGNORE KEY, 724, 730
ignore-space
  mysql option, 499
IGNORE_SPACE SQL mode, 251
implicit default values, 756
IMPORT TABLESPACE, 750, 881
importing
  data, 532
IN, 618
increasing
  performance, 420
increasing with replication
  speed, 393
index
  deleting, 748, 771
index hints, 724, 730
indexes, 752
  and BLOB columns, 466, 758
  and IS NULL, 469
  and LIKE, 469
  and NULL values, 758
  and TEXT columns, 466, 758
  assigning to key cache, 816
  block size, 266
  columns, 466
  leftmost prefix of, 469
  multi-column, 467
  multiple-part, 752
  names, 542
  use of, 468
INET_ATON(), 694
INET_NTOA(), 694
information functions, 686
INFORMATION_SCHEMA, 1044, 1045
INNER JOIN, 729
InnoDB, 860
InnoDB storage engine, 834, 860
InnoDB table type, 834, 860
InnoDB tables, 29
INSERT, 456, 708
INSERT ... SELECT, 712
INSERT DELAYED, 713, 713
INSERT statement
  grant privileges, 329
INSERT(), 628
inserting
  speed of, 456
installation layouts, 60
installation overview, 94
installing
  binary distribution, 91
  Linux RPM packages, 85
  Mac OS X PKG packages, 87
  overview, 43
  Perl, 176
  Perl on Windows, 177
  source distribution, 94
  user-defined functions, 1264
INSTR(), 629
INT, 585
INTEGER, 585
integers, 540
InteriorRingN(), 1017
internal compiler errors, 105

internal locking, 461
internals, 1253
Internet Relay Chat, 23
Intersection(), 1020
Intersects(), 1023
INTERVAL(), 619
introducer
  string literal, 538, 562
invalid data
  constraint, 35
IRC, 23
IS boolean_value, 617
IS NOT boolean_value, 617
IS NOT NULL, 617
IS NULL, 448, 617
  and indexes, 469
ISAM storage engine, 834, 858
ISAM table type, 834, 858
IsClosed(), 1015, 1016
IsEmpty(), 1013
ISNULL(), 619
ISOLATION LEVEL, 779
IsRing(), 1016
IsSimple(), 1013
IS_FREE_LOCK(), 694
IS_USED_LOCK(), 695
ITERATE, 1038

J

Java connectivity, 1219
JDBC, 1219
JOIN, 729

K

Key cache
  MyISAM, 470
key cache
  assigning indexes to, 816
key space
  MyISAM, 839
keys, 466
  foreign, 31, 209
  multi-column, 467
  searching on two, 210
keywords, 550
key_buffer_size myisamchk variable, 351
KEY_COLUMN_USAGE
  INFORMATION_SCHEMA table, 1054
KILL, 819
known errors, 36

L

language support, 369
last row
  unique ID, 1146
LAST_DAY(), 659
LAST_INSERT_ID(), 31, 711
LAST_INSERT_ID([expr]), 690
layout of installation, 60
LCASE(), 629
LD_LIBRARY_PATH environment variable, 178
LD_RUN_PATH environment variable, 150, 157, 178, 1573
LEAST(), 619
LEAVE, 1038
LEFT JOIN, 450, 729
LEFT OUTER JOIN, 729
LEFT(), 629
leftmost prefix of indexes, 469
legal names, 542
LENGTH(), 629
less than (<), 616
less than or equal (<=), 616
libmysqld, 1149
libraries
  list of, 1313
library
  mysqlclient, 1060
  mysqld, 1060
License, 1583
LIKE, 637
  and indexes, 469
  and wildcards, 469
LIMIT, 455, 689, 726
limitations
  design, 426
  replication, 405
limits
  file-size, 9
linefeed (\n), 539
LineFromText(), 1005
LineFromWKB(), 1006
LINESTRING, 1004
LineString(), 1008
LineStringFromText(), 1005
LineStringFromWKB(), 1006
linking, 1147
  errors, 1283
  problems, 1147
  speed, 481
links
  symbolic, 485
Linux
  binary distribution, 148
  source distribution, 149
literals, 538
LN(), 644
LOAD DATA FROM MASTER, 825
LOAD DATA INFILE, 714, 1294
LOAD TABLE FROM MASTER, 826
loading
  tables, 188
LOAD_FILE(), 629
local-infile
  mysql option, 499

mysqlcc option, 518
LOCALTIME, 659
LOCALTIMESTAMP, 659
LOCATE(), 630
LOCK IN SHARE MODE, 728
LOCK TABLES, 776
locking, 477
page-level, 461
row-level, 31, 461
table-level, 461
locking methods, 461
log
changes, 1323
mysqld_multi option, 236
log files, 98
maintaining, 380
names, 343
Log files, 374
LOG(), 644
LOG10(), 645
LOG2(), 645
logical operators, 620
LONG, 605
LONGBLOB, 591
LONGTEXT, 591
LOOP, 1037
LOWER(), 630
LPAD(), 630
LTRIM(), 631

M

Mac OS X
installation, 87
mailing address
for customer support, 22
mailing list address, 1
mailing lists, 16
archive location, 18
guidelines, 22
main features of MySQL, 5
maintaining
log files, 380
tables, 360
MAKEDATE(), 659
MAKETIME(), 660
make_binary_distribution, 228
MAKE_SET(), 631
manual
available formats, 2
online location, 1
typographical conventions, 2
master/slave setup, 393
MASTER_POS_WAIT(), 695, 826
MATCH ... AGAINST(), 670
matching
patterns, 197
mathematical functions, 641
MAX(), 699
MAX(DISTINCT), 699
MAXDB SQL mode, 254
maximum memory used, 511
max_allowed_packet variable, 502, 519
MAX_CONNECTIONS_PER_HOUR, 331
max_join_size variable, 502, 519
MAX_QUERIES_PER_HOUR, 331
MAX_UPDATES_PER_HOUR, 331
MAX_USER_CONNECTIONS, 331
MBR, 1020
MBRContains(), 1020
MBRDisjoint(), 1021
MBREqual(), 1021
MBRIntersects(), 1021
MBROverlaps(), 1021
MBRTouches(), 1021
MBRWithin(), 1021
MD5(), 685
MEDIUMBLOB, 591
MEDIUMINT, 585
MEDIUMTEXT, 591
MEMORY storage engine, 834, 846
MEMORY table type, 834, 846
memory usage
myisamchk, 355
memory use, 482, 511
MERGE storage engine, 834, 843
MERGE table type, 834, 843
MERGE tables
defined, 843
messages
languages, 369
metadata
database, 1044
methods
locking, 461
MICROSECOND(), 660
MID(), 631
MIN(), 699
MIN(DISTINCT), 699
Minimum Bounding Rectangle, 1020
minus
unary (-), 640
MINUTE(), 660
mirror sites, 57
miscellaneous functions, 693
MIT-pthreads, 107
MLineFromText(), 1005
MLineFromWKB(), 1006
MOD (modulo), 645
MOD(), 645
modes
batch, 204
modules
list of, 8
modulo (%), 645
modulo (MOD), 645
monitor

terminal, 181
MONTH(), 660
MONTHNAME(), 661
MPointFromText(), 1005
MPointFromWKB(), 1007
MPolyFromText(), 1005
MPolyFromWKB(), 1007
mSQL compatibility, 639
msql2mysql, 1060
MSSQL SQL mode, 254
multi mysqld, 235
multi-byte character sets, 1282
multi-byte characters, 372
multi-column indexes, 467
MULTILINESTRING, 1004
MultiLineString(), 1008
MultiLineStringFromText(), 1005
MultiLineStringFromWKB(), 1006
multiple servers, 381
multiple-part index, 752
multiplication (*), 640
MULTIPOINT, 1004
MultiPoint(), 1008
MultiPointFromText(), 1005
MultiPointFromWKB(), 1007
MULTIPOLYGON, 1004
MultiPolygon(), 1008
MultiPolygonFromText(), 1005
MultiPolygonFromWKB(), 1007
My
derivation, 5
my.cnf file, 405
MyISAM
compressed tables, 491, 841
size, 610
MyISAM key cache, 470
MyISAM storage engine, 834, 836
MyISAM table type, 834, 836
myisamchk, 101, 228
example output, 362
options, 350
myisampack, 489, 491, 765, 841
myisam_block_size myisamchk variable, 351
MyODBC, 1156
reporting problems, 1158
MySQL
defined, 4
introduction, 4
pronunciation, 5
mysql, 489, 497
MySQL binary distribution, 46
MYSQL C type, 1062
MySQL C type, 1113
mysql command-line options, 498
mysql commands
list of, 503
MySQL Dolphin name, 5
MySQL history, 5
mysql history file, 502
MySQL mailing lists, 16
MySQL name, 5
mysql prompt command, 504
MySQL source distribution, 46
mysql status command, 503
MySQL storage engines, 834
MySQL table types, 834
MySQL version, 57
MySQL++, 1154
mysql.server, 227
mysql.sock
changing location of, 99
protection, 1291
MYSQL323 SQL mode, 254
MYSQL40 SQL mode, 254
mysqlaccess, 489
mysqladmin, 489, 508, 752, 771, 810, 813, 817, 819
mysqld_multi option, 236
mysqladmin command-line options, 512
mysqlbinlog, 489, 513
mysqlbug, 228
mysqlbug script, 18
location, 1
mysqlcc, 489, 517
mysqlcc command-line options, 518
mysqlcheck, 490
mysqlclient library, 1060
mysqld, 227
mysqld_multi option, 236
starting, 1284
mysqld library, 1060
mysqld options, 239, 478
mysqld server
buffer sizes, 478
mysqld-max, 227, 228
mysqldump, 145, 490, 523
mysqld_multi, 227, 235
mysqld_safe, 227, 231
mysqlhotcopy, 490
mysqlimport, 145, 490, 532, 715
mysqlshow, 490
mysqltest
MySQL Test Suite, 1253
mysql_affected_rows(), 1069
mysql_autocommit()., 1110
MYSQL_BIND C type, 1112
mysql_change_user(), 1070
mysql_character_set_name(), 1071
mysql_close(), 1072
mysql_commit()., 1109
mysql_config, 1060
mysql_connect(), 1072
mysql_create_db(), 1073
mysql_data_seek(), 1073
MYSQL_DEBUG environment variable, 490, 1568, 1573
mysql_debug(), 1074

mysql_drop_db(), 1074
mysql_dump_debug_info(), 1075
mysql_eof(), 1075
mysql_errno(), 1076
mysql_error(), 1077
mysql_escape_string(), 1077
mysql_fetch_field(), 1078
mysql_fetch_fields(), 1078
mysql_fetch_field_direct(), 1079
mysql_fetch_lengths(), 1079
mysql_fetch_row(), 1080
MYSQL_FIELD C type, 1063
mysql_field_count(), 1081, 1091
MYSQL_FIELD_OFFSET C type, 1063
mysql_field_seek(), 1082
mysql_field_tell(), 1082
mysql_fix_privilege_tables, 228, 317
mysql_free_result(), 1083
mysql_get_client_info(), 1083
mysql_get_client_version(), 1083
mysql_get_host_info(), 1084
mysql_get_proto_info(), 1084
mysql_get_server_info(), 1084
mysql_get_server_version(), 1084
mysql_hex_string(), 1085
MYSQL_HISTFILE environment variable, 502, 1573
MYSQL_HOST environment variable, 310, 1573
mysql_info(), 712, 722, 745, 750, 1086
mysql_init(), 1087
mysql_insert_id(), 31, 711, 1087
mysql_install_db, 228
mysql_install_db script, 118
mysql_kill(), 1088
mysql_library_end(), 1089
mysql_library_init(), 1088
mysql_list_dbs(), 1089
mysql_list_fields(), 1089
mysql_list_processes(), 1090
mysql_list_tables(), 1091
mysql_more_results()., 1110
mysql_next_result()., 1110
mysql_num_fields(), 1091
mysql_num_rows(), 1092
mysql_options(), 1093
mysql_ping(), 1095
MYSQL_PS1 environment variable, 1573
MYSQL_PWD environment variable, 310, 490, 1573
mysql_query(), 1096, 1145
mysql_real_connect(), 1097
mysql_real_escape_string(), 540, 1099
mysql_real_query(), 1100
mysql_reload(), 1101
MYSQL_RES C type, 1062
mysql_rollback()., 1109
MYSQL_ROW C type, 1062
mysql_row_seek(), 1102
mysql_row_tell(), 1102
mysql_select_db(), 1102
mysql_server_end(), 1145
mysql_server_init(), 1144
mysql_set_sever_option(), 1103
mysql_shutdown(), 1104
mysql_sqlstate(), 1104
mysql_ssl_set(), 1105
mysql_stat(), 1105
MYSQL_STMT C type, 1112
mysql_stmt_affected_rows(), 1118
mysql_stmt_attr_get(), 1118
mysql_stmt_attr_set(), 1119
mysql_stmt_bind_param(), 1119
mysql_stmt_bind_result(), 1120
mysql_stmt_close(), 1121
mysql_stmt_data_seek(), 1122
mysql_stmt_errno(), 1122
mysql_stmt_error()., 1122
mysql_stmt_execute(), 1123
mysql_stmt_fetch(), 1127
mysql_stmt_fetch_column(), 1131
mysql_stmt_field_count(), 1132
mysql_stmt_free_result(), 1126
mysql_stmt_init(), 1132
mysql_stmt_insert_id(), 1127
mysql_stmt_num_rows(), 1133
mysql_stmt_param_count(), 1133
mysql_stmt_param_metadata(), 1133
mysql_stmt_prepare(), 1134
mysql_stmt_reset(), 1135
mysql_stmt_result_metadata., 1135
mysql_stmt_row_seek(), 1136
mysql_stmt_row_tell(), 1137
mysql_stmt_send_long_data()., 1137
mysql_stmt_sqlstate(), 1139
mysql_stmt_store_result(), 1139
mysql_store_result(), 1106, 1145
MYSQL_TCP_PORT environment variable, 387, 387, 490, 1573
mysql_thread_end(), 1143
mysql_thread_id(), 1107
mysql_thread_init(), 1143
mysql_thread_safe(), 1143
MYSQL_UNIX_PORT environment variable, 120, 387, 387, 490, 1573
mysql_use_result(), 1108
mysql_warning_count()., 1109
my_init(), 1143
my_ulonglong C type, 1063
my_ulonglong values
  printing, 1063

N

named pipes, 74, 79
named-commands
  mysql option, 499
names, 542
  case sensitivity, 544
  variables, 545

naming
  releases of MySQL, 46
NATIONAL CHAR, 589
native functions
  adding, 1267
native thread support, 44
NATURAL LEFT JOIN, 729
NATURAL LEFT OUTER JOIN, 729
NATURAL RIGHT JOIN, 729
NATURAL RIGHT OUTER JOIN, 729
NCHAR, 589
negative values, 540
nested queries, 732
nested query, 732
nesting queries, 732
net etiquette, 18, 22
netmask notation
  in mysql.user table, 311
NetWare, 89
net_buffer_length variable, 502, 519
new procedures
  adding, 1268
new users
  adding, 94, 98
newline (\n), 539
no matching rows, 1296
no-auto-rehash
  mysql option, 499
no-beep
  mysql option, 499
no-log
  mysqld_multi option, 236
no-named-commands
  mysql option, 499
no-pager
  mysql option, 499
no-tee
  mysql option, 499
non-delimited strings, 596
Non-transactional tables, 1295
NOT
  logical, 620
NOT BETWEEN, 618
not equal (!=), 615
not equal (<>), 615
NOT IN, 619
NOT LIKE, 638
NOT NULL
  constraint, 35
NOT REGEXP, 638
Novell NetWare, 89
NOW(), 661
NO_AUTO_CREATE_USER SQL mode, 251
NO_AUTO_VALUE_ON_ZERO SQL mode, 252
NO_BACKSLASH_ESCAPES SQL mode, 252
NO_DIR_IN_CREATE SQL mode, 252
NO_FIELD_OPTIONS SQL mode, 252
NO_KEY_OPTIONS SQL mode, 252
NO_TABLE_OPTIONS SQL mode, 252
NO_UNSIGNED_SUBTRACTION SQL mode, 252
NO_ZERO_DATE SQL mode, 252
NO_ZERO_IN_DATE SQL mode, 253
NUL, 538
NULL, 196, 1293
  testing for null, 615, 617, 618, 623
NULL value, 196, 541
NULL values
  and AUTO_INCREMENT columns, 1294
  and indexes, 758
  and TIMESTAMP columns, 1294
  vs. empty values, 1293
NULLIF(), 624
numbers, 540
NUMERIC, 587
numeric types, 610
NumGeometries(), 1019
NumInteriorRings(), 1018
NumPoints(), 1015

O

OCT(), 631
OCTET_LENGTH(), 631
ODBC, 1156
ODBC compatibility, 544, 586, 614, 617, 729, 756, 1512
odbcadmin program, 1189
OLAP, 700
OLD_PASSWORD(), 685
ON DUPLICATE KEY, 708, 1342
one-database
  mysql option, 500
online location of manual, 1
ONLY_FULL_GROUP_BY SQL mode, 253
OPEN, 1036
Open Source
  defined, 4
open tables, 475, 511
OpenGIS, 995
opening
  tables, 475
opens, 511
OpenSSL, 335
open_files_limit variable, 516
operating systems
  file-size limits, 9
  supported, 44
  Windows versus Unix, 82
operations
  arithmetic, 640
operators, 613
  cast, 639, 678
  logical, 620
optimization
  tips, 458
optimizations, 440, 446
OPTIMIZE TABLE, 792

optimizer
  controlling, 480
optimizing
  DISTINCT, 449
  filesort, 452
  GROUP BY, 453
  LEFT JOIN, 450
  LIMIT, 455
  tables, 360
option files, 220, 318
options
  command-line, 239
    mysql, 498
    mysqladmin, 512
    mysqlcc, 518
  configure, 98
  myisamchk, 350
  provided by MySQL, 181
  replication, 405
OR, 210, 446
  bitwise, 681
  logical, 621
OR Index Merge optimization, 446
Oracle compatibility, 26, 699, 773
ORACLE SQL mode, 254
ORD(), 632
ORDER BY, 192, 725, 748
  aliases in, 703
OUTFILE, 727
Overlaps(), 1023
overview, 1

P

packages
  list of, 1314
pack_isam, 491
page-level locking, 461
pager
  mysql option, 500
parameters
  server, 478
parentheses (and), 614
password
  mysql option, 500
  mysqlcc option, 518
  mysqld_multi option, 236
  root user, 125
password encryption
  reversibility of, 686
PASSWORD(), 312, 333, 686, 1281
passwords
  for users, 327
  forgotten, 1285
  lost, 1285
  resetting, 1285
  security, 301
  setting, 333, 784, 788
PATH environment variable, 93, 218, 1573
pattern matching, 197
performance
  benchmarks, 429
  disk issues, 484
  estimating, 439
  improving, 420, 465
PERIOD_ADD(), 661
PERIOD_DIFF(), 661
Perl
  installing, 176
  installing on Windows, 177
Perl API, 1154
Perl DBI/DBD
  installation problems, 178
permission checks
  effect on speed, 430
perror, 490, 536
PHP API, 1153
PI(), 646
PIPES_AS_CONCAT SQL mode, 253
plugins_path
  mysqlcc option, 518
POINT, 1004
Point(), 1008
PointFromText(), 1005
PointFromWKB(), 1007
PointN(), 1015
PointOnSurface(), 1018
PolyFromText(), 1006
PolyFromWKB(), 1007
POLYGON, 1004
Polygon(), 1008
PolygonFromText(), 1006
PolygonFromWKB(), 1007
port
  mysql option, 500
  mysqlcc option, 518
portability, 427
  types, 611
porting
  to other systems, 1562
POSITION(), 632
post-install
  multiple servers, 381
post-installation
  setup and testing, 113
PostgreSQL compatibility, 27
POSTGRESQL SQL mode, 255
POW(), 646
POWER(), 646
PREPARE, 831
PRIMARY KEY, 748, 757
  constraint, 34
primary key
  deleting, 748
privilege
  changes, 316
privilege information

location, 306
privilege system, 301
described, 301
privileges
access, 301
adding, 328
default, 125
deleting, 331, 779
display, 803
dropping, 331, 779
granting, 780
revoking, 780
problems
access denied errors, 1271
common errors, 1270
compiling, 104
DATE columns, 1292
date values, 597
installing on IBM-AIX, 164
installing on Solaris, 155
installing Perl, 178
linking, 1283
ODBC, 1158
reporting, 18
starting the server, 122
table locking, 463
time zone, 1291
PROCEDURE, 728
procedures
adding, 1268
stored, 31, 1028
process support, 44
processes
display, 808
processing
arguments, 1262
PROCESSLIST, 808
program variables
setting, 225
programs
client, 1147
crash-me, 427
list of, 227
prompt
mysql option, 500
prompts
meanings, 184
pronunciation
MySQL, 5
protocol
mysql option, 500
Protocol mismatch, 143
PURGE MASTER LOGS, 821
Python API, 1155

Q

QUARTER(), 662
queries
entering, 182
estimating performance, 439
examples, 205
speed of, 430
Twin Studies project, 212
query
mysqlcc option, 519
Query Cache, 388
query log, 375
questions, 511
answering, 22
quick
mysql option, 500
QUOTE(), 632
quotes
in strings, 539
quoting, 540
quoting binary data, 540
quoting of identifiers, 542

R

RADIANS(), 646
RAID
compile errors, 105
table type, 762
RAND(), 647
raw
mysql option, 500
re-creating
grant tables, 115
read_buffer_size myisamchk variable, 351
REAL, 586
REAL_AS_FLOAT SQL mode, 253
reconfiguring, 104, 104
reconnect
mysql option, 500
recovery
from crash, 356
reducing
data size, 465
references, 749
ref_or_null, 448
regex, 1574
REGEXP, 639
register
mysqlcc option, 519
regular expression syntax
described, 1574
Related(), 1023
relational databases
defined, 4
release numbers, 46
releases
naming scheme, 46
testing, 47
updating, 49
RELEASE_LOCK(), 695
RENAME TABLE, 772

RENAME USER, 788
renaming user accounts, 788
reordering
  columns, 1300
repair options
  myisamchk, 353
REPAIR TABLE, 792
repairing
  tables, 357
REPEAT, 1038
REPEAT(), 632
replace, 490
REPLACE, 722
REPLACE ... SELECT, 712
replace utility, 537
REPLACE(), 633
replication, 393
replication limitations, 405
replication masters
  commands, 821
replication options, 405
replication slaves
  commands, 823
reporting
  bugs, 18
  Connector/ODBC problems, 1158
  errors, 1, 16
  MyODBC problems, 1158
REQUIRE GRANT option, 340, 786
reserved words
  exceptions, 550
RESET MASTER, 822
RESET SLAVE, 826
restarting
  the server, 117
RESTORE TABLE, 793
retrieving
  data from tables, 189
return (\r), 539
return values
  UDFs, 1263
REVERSE(), 633
REVOKE, 780
revoking
  privileges, 780
RIGHT JOIN, 729
RIGHT OUTER JOIN, 729
RIGHT(), 633
RLIKE, 639
ROLLBACK, 29, 774
ROLLBACK TO SAVEPOINT, 776
ROLLUP, 700
root password, 125
root user
  password resetting, 1285
ROUND(), 647
rounding errors, 585, 649
ROUTINES
  INFORMATION_SCHEMA table, 1055
row-level locking, 461
rows
  counting, 199
  deleting, 1295
  locking, 31
  matching problems, 1296
  selecting, 190
  sorting, 192
ROW_COUNT(), 691
RPAD(), 633
RPM file, 85
RPM Package Manager, 85
RTRIM(), 634
RTS-threads, 1570
running
  ANSI mode, 24
  batch mode, 204
  multiple servers, 381
  queries, 182
running configure after prior invocation, 104

S

safe-updates
  mysql option, 500
safe-updates option, 507
safe_mysqld, 231
Sakila, 5
SAVEPOINT, 776
schema
  altering, 746
  creating, 751
  deleting, 770
SCHEMATA
  INFORMATION_SCHEMA table, 1046
SCHEMA_PRIVILEGES
  INFORMATION_SCHEMA table, 1051
script files, 204
scripts, 231, 235
  mysqlbug, 18
  mysql_install_db, 118
  SQL, 497
searching
  and case sensitivity, 1292
  full-text, 670
  MySQL Web pages, 18
  two keys, 210
SECOND(), 662
security
  against attackers, 297
security system, 301
SEC_TO_TIME(), 662
SELECT
  LIMIT, 723
  optimizing, 430
  Query Cache, 388
SELECT INTO, 1034
SELECT INTO TABLE, 28

SELECT speed, 440
selecting
  databases, 186
select_limit variable, 502, 519
SEQUENCE, 211
sequence emulation, 691
sequences, 211
server
  connecting, 181, 309
  debugging, 1563
  disconnecting, 181
  mysqlcc option, 519
  restart, 117
  shutdown, 117
  starting, 114
  starting and stopping, 120
  starting problems, 122
server administration, 508
server variables, 255, 546, 813
servers
  multiple, 381
SESSION_USER(), 692
SET, 591, 608, 794, 1033
  size, 611
SET GLOBAL SQL_SLAVE_SKIP_COUNTER, 826
SET OPTION, 794
SET PASSWORD, 788
SET PASSWORD statement, 333
SET SQL_LOG_BIN, 822
SET TRANSACTION, 779
setting
  passwords, 333
setting passwords, 788
setting program variables, 225
setup
  post-installation, 113
SHA(), 686
SHA1(), 686
shell syntax, 4
SHOW BINARY LOGS, 822
SHOW BINLOG EVENTS, 799, 822
SHOW CHARACTER SET, 799
SHOW COLLATION, 800
SHOW COLUMNS, 799, 800
SHOW CREATE DATABASE, 799, 801
SHOW CREATE FUNCTION, 1032
SHOW CREATE PROCEDURE, 1032
SHOW CREATE SCHEMA, 799, 801
SHOW CREATE TABLE, 799, 801
SHOW CREATE VIEW, 801
SHOW DATABASES, 799, 801
SHOW ENGINES, 799, 799, 802
SHOW ERRORS, 799, 803
SHOW extensions, 1058
SHOW FIELDS, 799
SHOW FUNCTION STATUS, 1032
SHOW GRANTS, 799, 803
SHOW INDEX, 799, 804
SHOW INNODB STATUS, 799
SHOW KEYS, 799, 804
SHOW MASTER LOGS, 799, 822
SHOW MASTER STATUS, 799, 822
SHOW PRIVILEGES, 799, 806
SHOW PROCEDURE STATUS, 1032
SHOW PROCESSLIST, 799, 808
SHOW SCHEMAS, 799, 801
SHOW SLAVE HOSTS, 799, 822
SHOW SLAVE STATUS, 799, 827
SHOW STATUS, 799
SHOW STORAGE ENGINES, 802
SHOW TABLE STATUS, 799
SHOW TABLE TYPES, 799, 802
SHOW TABLES, 799, 813
SHOW VARIABLES, 799
SHOW WARNINGS, 799, 814
SHOW with WHERE, 1044, 1058
showing
  database information, 534
shutdown_timeout variable, 513
shutting down
  the server, 117
sigint-ignore
  mysql option, 501
SIGN(), 648
silent
  mysql option, 501
  mysqld_multi option, 236
silent column changes, 764
SIN(), 648
single quote (\'), 538
size of tables, 9
sizes
  display, 584
skip-column-names
  mysql option, 501
skip-line-numbers
  mysql option, 501
slow queries, 511
slow query log, 379
SMALLINT, 585
socket
  mysql option, 501
  mysqlcc option, 519
socket location
  changing, 99
Solaris installation problems, 155
Solaris troubleshooting, 106
sorting
  character sets, 367
  data, 192
  grant tables, 313, 314
  table rows, 192
sort_buffer_size myisamchk variable, 351
sort_key_blocks myisamchk variable, 351
SOUNDEX(), 634
SOUNDS LIKE, 634

source distribution
installing, 94
source distributions
on Linux, 149
SPACE(), 634
Spatial Extensions in MySQL, 995
speed
compiling, 481
increasing with replication, 393
inserting, 456
linking, 481
of queries, 430, 440
SQL
defined, 4
SQL commands
replication masters, 821
replication slaves, 823
SQL scripts, 497
SQL-92
extensions to, 23
SQL_BIG_RESULT, 728
SQL_BUFFER_RESULT, 728
SQL_CACHE, 390, 729
SQL_CALC_FOUND_ROWS, 729
SQL_NO_CACHE, 390, 729
SQL_SMALL_RESULT, 728
sql_yacc.cc problems, 105
SQRT(), 648
square brackets, 584
SRID(), 1013
SSH, 342
SSL and X509 Basics, 335
SSL command-line options, 341
SSL related options, 340, 786
stability, 8
standard SQL
extensions to, 23
Standard SQL
differences from, 787
standards compatibility, 23
START SLAVE, 830
START TRANSACTION, 774
starting
comments, 33
mysqld, 1284
the server, 114
the server automatically, 120
Starting many servers, 381
StartPoint(), 1016
startup options
default, 220
startup parameters, 478
mysql, 498
mysqladmin, 512
mysqlcc, 518
tuning, 477
statements
GRANT, 328
INSERT, 329
statically
compiling, 99
STATISTICS
INFORMATION_SCHEMA table, 1049
status
tables, 811
status command
results, 511
status variables, 283, 810
STD(), 699
STDDEV(), 699
STDDEV_POP(), 699
STDDEV_SAMP(), 699
STOP SLAVE, 831
stopping
the server, 120
storage engines
choosing, 834
storage of data, 465
storage requirements
column type, 610
storage space
minimising, 465
stored procedures, 1028
stored procedures and triggers
defined, 31
STRAIGHT_JOIN, 728, 729
STRCMP(), 639
STRICT SQL mode, 250
STRICT_ALL_TABLES SQL mode, 253
STRICT_TRANS_TABLES SQL mode, 250, 253
string collating, 371
string comparison functions, 637
string comparisons
case sensitivity, 637
string functions, 624
string literal introducer, 538, 562
string replacement
replace utility, 537
string types, 603
strings
defined, 538
escaping characters, 538
non-delimited, 596
striping
defined, 484
STR_TO_DATE(), 662
SUBDATE(), 663
subqueries, 732
subquery, 732
subselects, 732
SUBSTRING(), 634
SUBSTRING_INDEX(), 635
SUBTIME(), 663
subtraction (-), 640
SUM(), 699
SUM(DISTINCT), 699

superuser, 125
support
  for operating systems, 44
  mailing address, 22
suppression
  default values, 35
Sybase compatibility, 774
symbolic links, 485, 487
SymDifference(), 1020
syntax
  mysqlcc option, 519
  regular expression, 1574
syntax_file
  mysqlcc option, 519
SYSDATE(), 664
system
  privilege, 301
  security, 295
system optimization, 477
system table, 432
system variables, 255, 546, 813
SYSTEM_USER(), 692

T

tab (\t), 539
table
  changing, 746, 748, 1299
  deleting, 771
  mysql option, 501
table aliases, 725
table cache, 475
table is full, 796, 1279
table names
  case sensitivity, 544
  case-sensitivity, 25
Table scans
  avoiding, 455
table types
  choosing, 834
table-level locking, 461
tables
  ARCHIVE, 857
  BDB, 848
  Berkeley DB, 848
  changing column order, 1300
  checking, 352
  closing, 475
  compressed, 491
  compressed format, 841
  constant, 432, 441
  copying, 762
  counting rows, 199
  creating, 187
  CSV, 857
  defragment, 361, 840
  defragmenting, 792
  deleting rows, 1295
  displaying, 534
  displaying status, 811
  dumping, 523, 530
  dynamic, 840
  error checking, 357
  EXAMPLE, 854
  FEDERATED, 854
  flush, 511
  fragmentation, 792
  grant, 316
  HEAP, 846
  host, 316
  improving performance, 465
  information, 361
  information about, 203
  InnoDB, 860
  ISAM, 858
  loading data, 188
  maintenance regimen, 360
  maximum size, 9
  MEMORY, 846
  MERGE, 843
  merging, 843
  multiple, 201
  MyISAM, 836
  names, 542
  open, 475
  opening, 475
  optimizing, 360
  partitioning, 843
  RAID, 762
  repairing, 357
  retrieving data, 189
  selecting columns, 191
  selecting rows, 190
  sorting rows, 192
  symbolic links, 486
  system, 432
  too many, 477
  unique ID for last row, 1146
  updating, 29
TABLES
  INFORMATION_SCHEMA table, 1046
table_cache, 475
TABLE_PRIVILEGES
  INFORMATION_SCHEMA table, 1051
TAN(), 648
tar
  problems on Solaris, 155
Tcl API, 1155
tcp-ip
  mysqld_multi option, 236
TCP/IP, 74, 79
technical support
  mailing address, 22
tee
  mysql option, 501
temporary file
  write access, 119

temporary tables
  problems, 1300
terminal monitor
  defined, 181
testing
  connection to the server, 310
  installation, 114
  of MySQL releases, 47
  post-installation, 113
testing mysqld
  mysqltest, 1253
Texinfo, 2
TEXT, 591, 605
  size, 611
TEXT columns
  default values, 606
  indexing, 466, 758
text files
  importing, 532
thread packages
  differences between, 1571
thread support, 44
  non-native, 107
threaded clients, 1147
threads, 511, 808, 1253
  display, 808
  RTS, 1570
TIME, 588, 602
time types, 610
time zone problems, 1291
TIME(), 664
TIMEDIFF(), 664
timeout, 262, 693, 714
  connect_timeout variable, 502, 513, 519
  shutdown_timeout variable, 513
TIMESTAMP, 587, 595
  and NULL values, 1294
TIMESTAMP(), 664
TIMESTAMPADD(), 665
TIMESTAMPDIFF(), 665
TIME_FORMAT(), 665
TIME_TO_SEC(), 666
TINYBLOB, 590
TINYINT, 585
TINYTEXT, 590
tips
  optimization, 458
TMPDIR environment variable, 120, 1573
TODO
  embedded server, 1150
  symlinks, 487
ToDo list for MySQL, 1316
tools
  command-line, 497
  graphical, 517
  GUI, 517
  list of, 1314
  mysqld_multi, 235
  mysqld_safe, 231
  safe_mysqld, 231
Touches(), 1023
TO_DAYS(), 666
trace DBI method, 1566
TRADITIONAL SQL mode, 250, 255
transaction-safe tables, 29, 860
transactions
  support, 29, 860
translations_path
  mysqlcc option, 519
Translators
  list of, 1311
triggers, 31, 1040
TRIM(), 635
troubleshooting
  FreeBSD, 106
  Solaris, 106
TRUE, 540
  testing for, 617
TRUNCATE, 743
TRUNCATE(), 649
tutorial, 181
Twin Studies
  queries, 212
type conversions, 614
types
  columns, 584, 611
  data, 584
  date, 610
  Date and Time, 594
  numeric, 610
  of tables, 834
  portability, 611
  strings, 603
  time, 610
Types, 584
typographical conventions, 2
TZ environment variable, 1291, 1573

U

UCASE(), 635
UCS-2, 554
UDFs, 1257
  compiling, 1264
  defined, 1256
  return values, 1263
ulimit, 1282
UMASK environment variable, 1285, 1573
UMASK_DIR environment variable, 1285, 1573
unary minus (-), 640
unbuffered
  mysql option, 501
UNCOMPRESS(), 636
UNCOMPRESSED_LENGTH(), 636
UNHEX(), 636
Unicode, 554
Unicode Collation Algorithm, 577

UNION, 210, 731
Union(), 1020
UNIQUE, 748
  constraint, 34
unique ID, 1146
UNIX_TIMESTAMP(), 666
UNKNOWN
  testing for, 617
unloading
  tables, 189
UNLOCK TABLES, 776
unnamed views, 738
UNTIL, 1038
UPDATE, 744
update log, 375
updating
  releases of MySQL, 49
  tables, 29
upgrading, 128
  3.20 to 3.21, 143
  3.21 to 3.22, 142
  3.22 to 3.23, 141
  3.23 to 4.0, 137
  4.0 to 4.1, 131
  different architecture, 145
  grant tables, 144
  to 5.0, 129
UPPER(), 636
uptime, 511
URLs for downloading MySQL, 57
USE, 773
USE INDEX, 724, 730
USE KEY, 724, 730
user
  mysql option, 501
  mysqlcc option, 519
  mysqld_multi option, 236
user accounts
  creating, 779
  renaming, 788
USER environment variable, 310, 1573
user privileges
  adding, 328
  deleting, 331, 779
  dropping, 331, 779
user table
  sorting, 313
user variables, 545
USER(), 692
user-defined functions
  adding, 1256, 1258
User-defined functions, 1257
usernames
  and passwords, 327
users
  adding, 94, 98
  deleting, 331, 779
  root, 125
USER_PRIVILEGES
  INFORMATION_SCHEMA table, 1050
uses
  of MySQL, 428
using multiple disks to start data, 487
UTC_DATE(), 667
UTC_TIME(), 667
UTC_TIMESTAMP(), 667
UTF-8, 554
UTF8, 554
UUID(), 695

V

valid numbers
  examples, 540
VARBINARY, 590
VARCHAR, 590, 603
  size, 611
VARCHARACTER, 590
variables
  mysqld, 478
  server, 255, 813
  status, 283, 810
  system, 255, 813
  System, 546
  user, 545
  values, 259
VARIANCE(), 700
VAR_POP(), 699
VAR_SAMP(), 700
verbose
  mysql option, 501
  mysqld_multi option, 236
version
  choosing, 46
  latest, 57
  mysql option, 501
  mysqlcc option, 519
  mysqld_multi option, 237
VERSION(), 692
vertical
  mysql option, 502
views, 33, 765
  updatable, 33, 765
VIEWS
  INFORMATION_SCHEMA table, 1057
virtual memory
  problems while compiling, 105
Visual Basic, 1190

W

wait
  mysql option, 502
WEEK(), 667
WEEKDAY(), 669
WEEKOFYEAR(), 669
Well-Known Binary format, 1003
Well-Known Text format, 1002

What is an X509/Certificate?, 336
What is encryption, 335
WHERE, 440
    with SHOW, 1044, 1058
WHILE, 1039
widths
    display, 584
Wildcard character (%), 539
Wildcard character (_), 539
wildcards
    and LIKE, 469
    in mysql.columns_priv table, 315
    in mysql.db table, 314
    in mysql.host table, 314
    in mysql.tables_priv table, 315
    in mysql.user table, 311
Windows, 1156
    compiling on, 113
    open issues, 84
    upgrading, 81
    versus Unix, 82
Within(), 1023
without-server option, 98
WKB format, 1003
WKT format, 1002
Word program, 1188
wrappers
    Eiffel, 1155
write access
    tmp, 119
write_buffer_size myisamchk variable, 351

X

X(), 1014
xml
    mysql option, 502
XOR
    bitwise, 681
    logical, 621

Y

Y(), 1014
YEAR, 588, 603
Year 2000 compliance, 10
Year 2000 issues, 603
YEAR(), 669
YEARWEEK(), 669