MySQL Reference Manual

Table of Contents

1 General Information

The MySQL ® 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 trademark of MySQL AB.

The MySQL software is Dual Licensed. Users can choose to use the MySQL software as an Open Source/Free Software 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 Section 1.4 [Licensing and Support], page 16.

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

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

- For information about the company behind the MySQL Database Server, see Section 1.3 [What is MySQL AB], page 12.
- For a discussion about the capabilities of the MySQL Database Server, see Section 1.2.2 [Features], page 6.
- For installation instructions, see Chapter 2 [Installing], page 59.
- For tips on porting the MySQL Database Software to new architectures or operating systems, see Appendix D [Porting], page 1198.
- For information about upgrading from a Version 4.0 release, see Section 2.5.2 [Upgrading-from-4.0], page 128.
- For information about upgrading from a Version 3.23 release, see Section 2.5.3 [Upgrading-from-3.23], page 131.
- For information about upgrading from a Version 3.22 release, see Section 2.5.4 [Upgrading-from-3.22], page 135.
- For a tutorial introduction to the MySQL Database Server, see Chapter 3 [Tutorial], page 171.
- 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 C [News], page 1040.
- For a list of currently known bugs and misfeatures, see Section 1.8.7 [Bugs], page 52.
- For future plans, see Section 1.6 [TODO], page 26.
- For a list of all the contributors to this project, see Appendix B [Credits], page 1029.

Important:

Reports of errors (often called "bugs"), as well as questions and comments, should be sent to the general MySQL mailing list. See Section 1.7.1.1 [Mailing-list], page 32. See Section 1.7.1.3 [Bug reports], page 34.

The mysqlbug script should be used to generate bug reports on Unix. (Windows distributions contain a file named 'mysqlbug.txt' in the base directory that can be used as a template for a bug report.)

For source distributions, the `mysqlbug` script can be found in the '`scripts`' directory. For binary distributions, `mysqlbug` can be found in the '`bin`' directory ('`/usr/bin`' for the `MySQL-server` RPM package).

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 `MySQL` reference manual; it documents `MySQL` up to Version 5.0.0-alpha. Functional changes are always indicated with reference to the version, so this manual is also suitable if you are using an older version of the `MySQL` software (such as 3.23 or 4.0-production). There are also references for version 5.0 (development).

Being a reference manual, it does not provide general instruction on `SQL` or relational database concepts. It also will not teach you how to use your operating system or command line interpreter.

As the `MySQL Database Software` is under constant development, the manual is also updated frequently. The most recent version of this manual is available at `http://www.mysql.com/documentation/` in many different formats, including HTML, PDF, and Windows HLP 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`.

The index can assist you in finding information in the manual. For online use, you can try the searchable version of the manual available at `http://www.mysql.com/doc/`.

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 now maintained by the MySQL Documentation Team, consisting of Arjen Lentz, Paul DuBois, and Stefan Hinz. For the many other contributors, see Appendix B [Credits], page 1029.

The copyright (2004) to this manual is owned by the Swedish company `MySQL AB`. See Section 1.4.2 [Copyright], page 17.

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."

'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 by 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 is `command.com` or `cmd.exe`, typically run in a Windows console.

Note that to enter a command or statement from an example, you do not type the prompt shown in the example.

Commands to set shell variables are shown using Bourne shell syntax. If you are using `csh` or `tcsh`, you will need to issue commands somewhat differently. 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
```

For `csh` or `tcsh`, you would execute the sequence like this:

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

Database, table, and column names must often be substituted into commands. 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 `INSERT` 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] ...
```

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, that builds its business by providing services around the `MySQL` database management system. See Section 1.3 [What is MySQL AB], page 12.

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 Section 1.4.3 [MySQL licenses], page 17.

Why use the MySQL Database Server?

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 [MySQL Benchmarks], page 399.

`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.

The technical features of MySQL Server

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 a multi-threaded library that you can link into your application to get a smaller, faster, easier-to-manage product.

There is a large amount of contributed MySQL software available.

It is very likely that you will find that your favorite application or language already 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`. `Sakila` 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 Section 1.5.1 [MySQL 4.0 Nutshell], page 21.

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 [Which OS], page 60.
- Uses GNU Automake, Autoconf, and Libtool for portability.
- APIs for C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, and Tcl are available. See Chapter 20 [Clients], page 857.
- Fully multi-threaded using kernel threads. This means 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 geometry types. See Chapter 12 [Column types], page 533.
- Fixed-length and variable-length records.

Commands 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 tbl_name
    -> 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 10.6 [Reserved words], page 503.
- You can mix tables from different databases in the same query (as of Version 3.22).

Security

- A privilege and password system that is very flexible and secure, and 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 `CHAR` or `VARCHAR` column.

Connectivity

- Clients may connect to the `MySQL` server using TCP/IP sockets on any platform. On Windows systems in the NT family (NT, 2000, or XP), clients may connect using named pipes. On Unix systems, clients may connect using Unix domain socket files.
- The Connector/ODBC 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 may be run on Windows or Unix. Connector/ODBC source is available. All ODBC 2.5 functions are supported, as are many others. See Section 20.3 [ODBC], page 957.

- The Connector/JDBC interface provides `MySQL` support for Java client programs that use JDBC connections. Clients may be run on Windows or Unix. Connector/JDBC source is available. See Section 20.4 [Java], page 966.

Localization
- The server can provide error messages to clients in many languages. See Section 5.7.2 [Languages], page 341.
- Full support for several different character sets, including ISO-8859-1 (Latin1), 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 (the Swedish way 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 [MySQL Database Administration], page 218.
- 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 list, which is 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 and 4.0 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 1.8.7 [Bugs], page 52.

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 — Gamma
: 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 (in 3.23 from 3.23.49)
: 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 — Gamma
: The Berkeley DB code is very stable, but we are still improving the BDB transactional storage engine interface in MySQL Server, so it will take some time before this is as well tested as the other table types.

Full-text searches — Beta
: Full-text searching works but is not yet widely used. Important enhancements have been implemented in MySQL 4.0.

Connector/ODBC 3.51 (uses ODBC SDK 3.51) — Stable
: In wide production use. Some issues brought up appear to be application-related and independent of the ODBC driver or underlying database server.

Automatic recovery of MyISAM tables — Gamma
: This status applies only to the new code in the MyISAM storage engine that checks whether the table was closed properly on open and executes an automatic check or repair of the table if it wasn't.

Paying customers receive high-quality support directly from MySQL AB. MySQL AB also provides the MySQL mailing list as a community resource where anyone may ask questions.

Bugs are usually fixed right away with a patch. For serious bugs, there is almost always a new release.

1.2.4 How Big MySQL Tables Can Be

MySQL Version 3.22 had a 4GB (4 gigabyte) limit on table size. With the MyISAM storage engine in MySQL Version 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 now normally is 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:

Operating System	**File-size Limit**
Linux-Intel 32-bit	2GB, much more when using LFS
Linux-Alpha	8TB (?)
Solaris 2.5.1	2GB (4GB possible with patch)
Solaris 2.6	4GB (can be changed with flag)
Solaris 2.7 Intel	4GB
Solaris 2.7 UltraSPARC	512GB
NetWare w/NSS filesystem	8TB

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. Most current Linux distributions are based on kernel 2.4 and already include all the required LFS patches. However, the maximum available file size still depends on several factors, one of them being the filesystem used to store MySQL tables.

For a very 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`.

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` command or with the `myisamchk -dv tbl_name`. See Section 14.5.3 [`SHOW`], page 701.

If you need a `MyISAM` table that will be 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 14.2.5 [`CREATE TABLE`], page 670. 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 14.2.2 [`ALTER TABLE`], page 663.

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`], page 449.
- Another way to get around the operating system file limit for `MyISAM` data files is by using the `RAID` options. See Section 14.2.5 [`CREATE TABLE`], page 670.
- `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 15.2 [`MERGE` tables], page 746.

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` Version 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 doesn't have any problems with dates 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 (0.00 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),
    -> ('2050-01-01','2050-01-01 00:00:00',20500101000000);
Query OK, 13 rows affected (0.01 sec)
Records: 13  Duplicates: 0  Warnings: 0

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 |
| 2050-01-01 | 2050-01-01 00:00:00 | 00000000000000 |
```

```
+------------+---------------------+----------------+
13 rows in set (0.00 sec)
```

The final TIMESTAMP column value is zero because the final year (2050) exceeds 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).

The example also shows that the DATE and DATETIME data types have no problems with the dates used. They handle dates through the year 9999.

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 12.3.4 [Y2K issues], page 547 for MySQL Server's rules for dealing with ambiguous date input data that contains two-digit year values.

1.3 Overview of MySQL AB

MySQL AB is the company of the MySQL founders and main developers. MySQL AB was originally established in Sweden by David Axmark, Allan Larsson, and Michael "Monty" Widenius.

The developers of the MySQL server are all employed by the company. We are a virtual organization with people in a dozen countries around the world. We communicate extensively over the Internet every day with one another and with our users, supporters, and partners.

We are dedicated to developing the MySQL database software and promoting it to new users. MySQL AB owns the copyright to the MySQL source code, the MySQL logo and trademark, and this manual. See Section 1.2 [What-is], page 4.

The MySQL core values show our dedication to MySQL and Open Source.

We want the MySQL Database Software to be:

- The best and the most widely used database in the world
- Available to, and affordable by all
- Easy to use
- Continuously improving while remaining fast and safe
- Fun to use and improve
- Free from bugs

MySQL AB and the people at MySQL AB:

- Promote Open Source philosophy and support the Open Source community

- Aim to be good citizens
- Prefer partners that share our values and mind-set
- Answer email and provide support
- Are a virtual company, networking with others
- Work against software patents

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

By the way, the "AB" part of the company name is the acronym for the Swedish "aktiebolag," or "stock company." It translates to "MySQL, Inc." In fact, MySQL, Inc. and MySQL GmbH are examples of MySQL AB subsidiaries. They are located in the US and Germany, respectively.

1.3.1 The Business Model and Services of MySQL AB

One of the most common questions we encounter is: "*How can you make a living from something you give away for free?*" This is how:

- `MySQL AB` makes money on support, services, commercial licenses, and royalties.
- We use these revenues to fund product development and to expand the `MySQL` business.

The company has been profitable since its inception. In October 2001, we accepted venture financing from leading Scandinavian investors and a handful of business angels. This investment is used to solidify our business model and build a basis for sustainable growth.

1.3.1.1 Support

`MySQL AB` is run and owned by the founders and main developers of the `MySQL` database. The developers are committed to providing support to customers and other users in order to stay in touch with their needs and problems. All our support is provided by qualified developers. Really tricky questions are answered by Michael `Monty` Widenius, principal author of the `MySQL Server`. See Section 1.4.1 [Support], page 16.

For more information and ordering support at various levels, see `http://www.mysql.com/support/` or contact our sales staff at `sales@mysql.com`.

1.3.1.2 Training and Certification

`MySQL AB` delivers `MySQL` and related training worldwide. We offer both open courses and in-house courses tailored to the specific needs of your company. `MySQL Training` is also available through our partners, the `Authorized MySQL Training Centers`.

Our training material uses the same sample databases used in our documentation and our sample applications, and is always updated to reflect the latest `MySQL` version. Our trainers are backed by the development team to guarantee the quality of the training and the continuous development of the course material. This also ensures that no questions raised during the courses remain unanswered.

Attending our training courses will enable you to achieve your `MySQL` application goals. You will also:

- Save time.
- Improve the performance of your applications.
- Reduce or eliminate the need for additional hardware, decreasing cost.
- Enhance security.
- Increase customers' and co-workers' satisfaction.
- Prepare yourself for MySQL Certification.

If you are interested in our training as a potential participant or as a training partner, please visit the training section at http://www.mysql.com/training/ or contact us at: training@mysql.com.

For details about the MySQL Certification Program, please see http://www.mysql.com/certification/.

1.3.1.3 Consulting

MySQL AB and its Authorized Partners offer consulting services to users of MySQL Server and to those who embed MySQL Server in their own software, all over the world.

Our consultants can help you design and tune your databases, construct efficient queries, tune your platform for optimal performance, resolve migration issues, set up replication, build robust transactional applications, and more. We also help customers embed MySQL Server in their products and applications for large-scale deployment.

Our consultants work in close collaboration with our development team, which ensures the technical quality of our professional services. Consulting assignments range from two-day power-start sessions to projects that span weeks and months. Our expertise not only covers MySQL Server—it also extends into programming and scripting languages such as PHP, Perl, and more.

If you are interested in our consulting services or want to become a consulting partner, please visit the consulting section of our Web site at http://www.mysql.com/consulting/ or contact our consulting staff at consulting@mysql.com.

1.3.1.4 Commercial Licenses

The MySQL database is released under the GNU General Public License (GPL). This means that the MySQL software can be used free of charge under the GPL. If you do not want to be bound by the GPL terms (such as the requirement that your application must also be GPL), you may purchase a commercial license for the same product from MySQL AB; see http://www.mysql.com/products/pricing.html. Since MySQL AB owns the copyright to the MySQL source code, we are able to employ Dual Licensing, which means that the same product is available under GPL and under a commercial license. This does not in any way affect the Open Source commitment of MySQL AB. For details about when a commercial license is required, please see Section 1.4.3 [MySQL licenses], page 17.

We also sell commercial licenses of third-party Open Source GPL software that adds value to MySQL Server. A good example is the InnoDB transactional storage engine that offers ACID support, row-level locking, crash recovery, multi-versioning, foreign key support, and more. See Chapter 16 [InnoDB], page 758.

1.3.1.5 Partnering

MySQL AB has a worldwide partner program that covers training courses, consulting and support, publications, plus reselling and distributing MySQL and related products. MySQL AB Partners get visibility on the http://www.mysql.com/ Web site and the right to use special versions of the MySQL trademarks to identify their products and promote their business.

If you are interested in becoming a MySQL AB Partner, please email partner@mysql.com.

The word MySQL and the MySQL dolphin logo are trademarks of MySQL AB. See Section 1.4.4 [MySQL AB Logos and Trademarks], page 19. These trademarks represent a significant value that the MySQL founders have built over the years.

The MySQL Web site (http://www.mysql.com/) is popular among developers and users. In December 2003, we served 16 million page views. Our visitors represent a group that makes purchase decisions and recommendations for both software and hardware. Twelve percent of our visitors authorize purchase decisions, and only nine percent are not involved in purchase decisions at all. More than 65% have made one or more online business purchases within the last half-year, and 70% plan to make one in the next few months.

1.3.2 Contact Information

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

For press services and inquiries not covered in our news releases (http://www.mysql.com/news/), please send email to press@mysql.com.

If you have a valid support contract with MySQL AB, you will get timely, precise answers to your technical questions about the MySQL software. For more information, see Section 1.4.1 [Support], page 16. On our Web site, see http://www.mysql.com/support/, or send an email message to sales@mysql.com.

For information about MySQL training, please visit the training section at http://www.mysql.com/training/. If you have restricted access to the Internet, please contact the MySQL AB training staff via email at training@mysql.com. See Section 1.3.1.2 [Business Services Training], page 13.

For information on the MySQL Certification Program, please see http://www.mysql.com/certification See Section 1.3.1.2 [Business Services Training], page 13.

If you're interested in consulting, please visit the consulting section of our Web site at http://www.mysql.com/consulting/. If you have restricted access to the Internet, please contact the MySQL AB consulting staff via email at consulting@mysql.com. See Section 1.3.1.3 [Business Services Consulting], page 14.

Commercial licenses may be purchased online at https://order.mysql.com/. There you will also find information on how to fax your purchase order to MySQL AB. More information about licensing can be found at http://www.mysql.com/products/licensing/. If you have questions regarding licensing or you want a quote for a high-volume license deal, please fill in the contact form on our Web site (http://www.mysql.com/) or send email to licensing@mysql.com (for licensing questions) or to sales@mysql.com (for sales inquiries). See Section 1.4.3 [MySQL licenses], page 17.

If you represent a business that is interested in partnering with MySQL AB, please send email to partner@mysql.com. See Section 1.3.1.5 [Business Services Partnering], page 15.

For more information on the MySQL trademark policy, refer to http://www.mysql.com/company/trademark or send email to trademark@mysql.com. See Section 1.4.4 [MySQL AB Logos and Trademarks], page 19.

If you are interested in any of the MySQL AB jobs listed in our jobs section (http://www.mysql.com/company/jobs/), please send email to jobs@mysql.com. Please do not send your CV as an attachment, but rather as plain text at the end of your email message.

For general discussion among our many users, please direct your attention to the appropriate mailing list. See Section 1.7.1 [Questions], page 31.

Reports of errors (often called "bugs"), as well as questions and comments, should be sent to the general MySQL mailing list. See Section 1.7.1.1 [Mailing-list], page 32. 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. See Section 1.7.1.3 [Bug reports], page 34.

If you have benchmark results that we can publish, please contact us via email at benchmarks@mysql.com.

If you have suggestions concerning additions or corrections to this manual, please send them to the manual team via email at docs@mysql.com.

For questions or comments about the workings or content of the MySQL Web site (http://www.mysql.com/), please send email to webmaster@mysql.com.

MySQL AB has a privacy policy, which can be read at http://www.mysql.com/company/privacy.html. For any queries regarding this policy, please send email to privacy@mysql.com.

For all other inquires, please send an email to info@mysql.com.

1.4 MySQL Support and Licensing

This section describes MySQL support and licensing arrangements.

1.4.1 Support Offered by MySQL AB

Technical support from MySQL AB means individualized answers to your unique problems direct from the software engineers who code the MySQL database engine.

We try to take a broad and inclusive view of technical support. Almost any problem involving MySQL software is important to us if it's important to you. Typically customers seek help on how to get different commands and utilities to work, remove performance bottlenecks, restore crashed systems, understand the impact of operating system or networking issues on MySQL, set up best practices for backup and recovery, utilize APIs, and so on. Our support covers only the MySQL server and our own utilities, not third-party products that access the MySQL server, though we try to help with these where we can.

Detailed information about our various support options is given at http://www.mysql.com/support/, where support contracts can also be ordered online. If you have restricted access to the Internet, please contact our sales staff via email at sales@mysql.com.

Technical support is like life insurance. You can live happily without it for years. However, when your hour arrives, it becomes critically important, but it's too late to buy it. If you use MySQL Server for important applications and encounter sudden difficulties, it may be too time-consuming to figure out all the answers yourself. You may need immediate access to the most experienced MySQL troubleshooters available, those employed by MySQL AB.

1.4.2 Copyrights and Licenses Used by MySQL

MySQL AB owns the copyright to the MySQL source code, the MySQL logos and trademarks, and this manual. See Section 1.3 [What is MySQL AB], page 12. Several different licenses are relevant to the MySQL distribution:

1. All the MySQL-specific source in the server, the mysqlclient library and the client, as well as the GNU readline library, are covered by the GNU General Public License. See Appendix G [GPL license], page 1216. The text of this license can be found as the file 'COPYING' in the distribution.
2. The GNU getopt library is covered by the GNU Lesser General Public License. See http://www.fsf.org/licenses/.
3. Some parts of the source (the regexp library) are covered by a Berkeley-style copyright.
4. Older versions of MySQL (3.22 and earlier) are subject to a stricter license (http://www.mysql.com/products/licensing/mypl.html). See the documentation of the specific version for information.
5. The MySQL reference manual is currently **not** distributed under a GPL-style license. Use of the manual is subject to the following terms:
   - Conversion to other formats is allowed, but the actual content may not be altered or edited in any way.
   - You may create a printed copy for your own personal use.
   - For all other uses, such as selling printed copies or using (parts of) the manual in another publication, prior written agreement from MySQL AB is required.

   Please send an email message to docs@mysql.com for more information or if you are interested in doing a translation.

For information about how the MySQL licenses work in practice, please refer to Section 1.4.3 [MySQL licenses], page 17. Also see Section 1.4.4 [MySQL AB Logos and Trademarks], page 19.

1.4.3 MySQL Licenses

The MySQL software is released under the GNU General Public License (GPL), which is probably the best known Open Source license. The formal terms of the GPL license can be found at http://www.fsf.org/licenses/. See also http://www.fsf.org/licenses/gpl-faq.html and http://www.gnu.org/philosophy/enforcing-gpl.h

Since the MySQL software is released under the GPL, it may often be used for free, but for certain uses you may want or need to buy commercial licenses from MySQL AB at https://order.mysql.com/. See http://www.mysql.com/products/licensing/ for more information.

Older versions of MySQL (3.22 and earlier) are subject to a stricter license (http://www.mysql.com/products/licensing/mypl.html). See the documentation of the specific version for information.

Please note that the use of the MySQL software under commercial license, GPL, or the old MySQL license does not automatically give you the right to use MySQL AB trademarks. See Section 1.4.4 [MySQL AB Logos and Trademarks], page 19.

1.4.3.1 Using the MySQL Software Under a Commercial License

The GPL license is contagious in the sense that when a program is linked to a GPL program, all the source code for all the parts of the resulting product must also be released under the GPL. If you do not follow this GPL requirement, you break the license terms and forfeit your right to use the GPL program altogether. You also risk damages.

You need a commercial license:

- When you link a program with any GPL code from the MySQL software and don't want the resulting product to be licensed under GPL, perhaps because you want to build a commercial product or keep the added non-GPL code closed source for other reasons. When purchasing commercial licenses, you are not using the MySQL software under GPL even though it's the same code.
- When you distribute a non-GPL application that **only** works with the MySQL software and ship it with the MySQL software. This type of solution is considered to be linking even if it's done over a network.
- When you distribute copies of the MySQL software without providing the source code as required under the GPL license.
- When you want to support the further development of the MySQL database even if you don't formally need a commercial license. Purchasing support directly from MySQL AB is another good way of contributing to the development of the MySQL software, with immediate advantages for you. See Section 1.4.1 [Support], page 16.

If you require a license, you will need one for each installation of the MySQL software. This covers any number of CPUs on a machine, and there is no artificial limit on the number of clients that connect to the server in any way.

For commercial licenses, please visit our Web site at http://www.mysql.com/products/licensing/. For support contracts, see http://www.mysql.com/support/. If you have special needs or you have restricted access to the Internet, please contact our sales staff via email at sales@mysql.com.

1.4.3.2 Using the MySQL Software for Free Under GPL

You can use the MySQL software for free under the GPL if you adhere to the conditions of the GPL. For additional details, including answers to common questions about the GPL, see the generic FAQ from the Free Software Foundation at http://www.fsf.org/licenses/gpl-faq.html. Common uses of the GPL include:

- When you distribute both your own application and the MySQL source code under the GPL with your product.

- When you distribute the MySQL source code bundled with other programs that are not linked to or dependent on the MySQL system for their functionality even if you sell the distribution commercially. This is called "mere aggregation" in the GPL license.
- When you are not distributing **any** part of the MySQL system, you can use it for free.
- When you are an Internet Service Provider (ISP), offering Web hosting with MySQL servers for your customers. We encourage people to use ISPs that have MySQL support, because doing so will give them the confidence that their ISP will, in fact, have the resources to solve any problems they may experience with the MySQL installation. Even if an ISP does not have a commercial license for MySQL Server, their customers should at least be given read access to the source of the MySQL installation so that the customers can verify that it is correctly patched.
- When you use the MySQL database software in conjunction with a Web server, you do not need a commercial license (so long as it is not a product you distribute). This is true even if you run a commercial Web server that uses MySQL Server, because you are not distributing any part of the MySQL system. However, in this case we would like you to purchase MySQL support because the MySQL software is helping your enterprise.

If your use of MySQL database software does not require a commercial license, we encourage you to purchase support from MySQL AB anyway. This way you contribute toward MySQL development and also gain immediate advantages for yourself. See Section 1.4.1 [Support], page 16.

If you use the MySQL database software in a commercial context such that you profit by its use, we ask that you further the development of the MySQL software by purchasing some level of support. We feel that if the MySQL database helps your business, it is reasonable to ask that you help MySQL AB. (Otherwise, if you ask us support questions, you are not only using for free something into which we've put a lot a work, you're asking us to provide free support, too.)

1.4.4 MySQL AB Logos and Trademarks

Many users of the MySQL database want to display the MySQL AB dolphin logo on their Web sites, books, or boxed products. We welcome and encourage this, although it should be noted that the word MySQL and the MySQL dolphin logo are trademarks of MySQL AB and may only be used as stated in our trademark policy at `http://www.mysql.com/company/trademark.html`.

1.4.4.1 The Original MySQL Logo

The MySQL dolphin logo was designed by the Finnish advertising agency Priority in 2001. The dolphin was chosen as a suitable symbol for the MySQL database management system, which is like a smart, fast, and lean animal, effortlessly navigating oceans of data. We also happen to like dolphins.

The original MySQL logo may only be used by representatives of MySQL AB and by those having a written agreement allowing them to do so.

1.4.4.2 MySQL Logos That May Be Used Without Written Permission

We have designed a set of special *Conditional Use* logos that may be downloaded from our Web site at `http://www.mysql.com/press/logos.html` and used on third-party Web sites without written permission from `MySQL AB`. The use of these logos is not entirely unrestricted but, as the name implies, subject to our trademark policy that is also available on our Web site. You should read through the trademark policy if you plan to use them. The requirements are basically as follows:

- Use the logo you need as displayed on the `http://www.mysql.com/` site. You may scale it to fit your needs, but may not change colors or design, or alter the graphics in any way.
- Make it evident that you, and not `MySQL AB`, are the creator and owner of the site that displays the `MySQL` trademark.
- Don't use the trademark in a way that is detrimental to `MySQL AB` or to the value of `MySQL AB` trademarks. We reserve the right to revoke the right to use the `MySQL AB` trademark.
- If you use the trademark on a Web site, make it clickable, leading directly to `http://www.mysql.com/`.
- If you use the `MySQL` database under `GPL` in an application, your application must be `Open Source` and must be able to connect to a `MySQL` server.

Contact us via email at `trademark@mysql.com` to inquire about special arrangements to fit your needs.

1.4.4.3 When You Need Written Permission to Use MySQL Logos

You need written permission from `MySQL AB` before using `MySQL` logos in the following cases:

- When displaying any `MySQL AB` logo anywhere except on your Web site.
- When displaying any `MySQL AB` logo except the *Conditional Use* logos (mentioned previously) on Web sites or elsewhere.

Due to legal and commercial reasons, we monitor the use of MySQL trademarks on products, books, and other items. We usually require a fee for displaying `MySQL AB` logos on commercial products, since we think it is reasonable that some of the revenue is returned to fund further development of the `MySQL` database.

1.4.4.4 MySQL AB Partnership Logos

`MySQL` partnership logos may be used only by companies and persons having a written partnership agreement with `MySQL AB`. Partnerships include certification as a `MySQL` trainer or consultant. For more information, please see Section 1.3.1.5 [Partnering], page 15.

1.4.4.5 Using the Word `MySQL` in Printed Text or Presentations

`MySQL AB` welcomes references to the `MySQL` database, but it should be noted that the word `MySQL` is a trademark of `MySQL AB`. Because of this, you must append the trademark symbol

(TM) to the first or most prominent use of the word MySQL in a text and, where appropriate, state that MySQL is a trademark of MySQL AB. For more information, please refer to our trademark policy at http://www.mysql.com/company/trademark.html.

1.4.4.6 Using the Word MySQL in Company and Product Names

Use of the word MySQL in product or company names or in Internet domain names is not allowed without written permission from MySQL AB.

1.5 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 production release series is MySQL 4.0, which was declared stable for production use as of Version 4.0.12, released in March 2003. This means that future 4.0 development will be limited only to making bugfixes. For the older MySQL 3.23 series, only critical bugfixes will be made.

Active MySQL development currently is taking place in the MySQL 4.1 and 5.0 release series. This means that new features are being added to MySQL 4.1 and MySQL 5.0. Both 4.1 and 5.0 are available now in alpha status.

Before upgrading from one release series to the next, please see the notes at Section 2.5 [Upgrade], page 126.

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

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

1.5.1 MySQL 4.0 in a Nutshell

Long awaited by our users, MySQL Server 4.0 is now available in production status.

MySQL 4.0 is available for download from http://www.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.

1.5.1.1 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 5.10 [Query Cache], page 358.
- 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 1.5.1.2 [Nutshell Embedded MySQL], page 23.

`InnoDB` storage engine as standard
- The `InnoDB` storage engine is now 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 now standard features. See Chapter 16 [`InnoDB`], page 758.

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 13.6 [Fulltext Search], page 597.

Standards compliance, portability, and migration
- Many users will also be happy to learn that MySQL Server now supports the `UNION` statement, a long-awaited standard SQL feature.
- MySQL now runs natively on the Novell NetWare 6.0 platform. See Section 2.2.4 [NetWare installation], page 93.
- Features to simplify migration from other database systems to MySQL Server include `TRUNCATE TABLE` (as in Oracle).

Internationalization
- Our German, Austrian, and Swiss users will note that `MySQL` now 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 now be set without taking down the server. This is a convenient feature for database administrators (DBAs). See Section 14.5.3.1 [SET OPTION], page 702.
- 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 now 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 C.3 [News-4.0.x], page 1059.

1.5.1.2 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 will 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. The embedded MySQL library uses the same interface as the normal client library, so it is convenient and easy to use. See Section 20.2.15 [libmysqld], page 951.

1.5.2 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.

With these additions, critics of the MySQL Database Server have to be more imaginative than ever in pointing out deficiencies in the MySQL database management system. Already well-known for its stability, speed, and ease of use, MySQL Server will be able to fulfill the requirement checklists of very demanding buyers.

1.5.2.1 Features Available in MySQL 4.1

The features listed in this section are implemented in MySQL 4.1. A few other features are still planned for MySQL 4.1. See Section 1.6.1 [TODO MySQL 4.1], page 26.

Most new features being coded are or will be available in MySQL 5.0. See Section 1.6.2 [TODO MySQL 5.0], page 26.

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 14.1.8 [Subqueries], page 652.

Speed enhancements
- Faster binary client/server protocol with support for prepared statements and parameter binding. See Section 20.2.4 [C API Prepared statements], page 911.
- `BTREE` indexing is now 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 now supports OpenGIS spatial types for storing geographical data. See Chapter 18 [Spatial extensions in MySQL], page 816.
- 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 operations such as bulk data loading much easier to track.
- `SHOW WARNINGS` shows warnings for the last command. See Section 14.5.3.20 [`SHOW WARNINGS`], page 719.

Internationalization
- To support applications that require the use of local languages, the MySQL software now offers extensive Unicode support through the `utf8` and `ucs2` character sets.
- Character sets can now 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 11 [Charset], page 507.

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 20.2.8 [C API multiple queries], page 942.

- 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` key (index). See Section 14.1.4 [INSERT], page 630.
- 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 13.9 [Group by functions and modifiers], page 618.

The news section of this manual includes a more in-depth list of features. See Section C.2 [News-4.1.x], page 1043.

1.5.2.2 Stepwise Rollout

New features are being added to MySQL 4.1. The alpha version is already available for download. See Section 1.5.2.3 [Nutshell Ready for Immediate Use], page 25.

The set of features that are being added to version 4.1 is mostly fixed. Additional development is already ongoing for version 5.0. MySQL 4.1 will go through the steps of *Alpha* (during which time new features might still be added/changed), *Beta* (when we have feature freeze and only bug corrections will be done), and *Gamma* (indicating that a production release is just weeks ahead). At the end of this process, MySQL 4.1 will become the new production release.

1.5.2.3 Ready for Immediate Development Use

MySQL 4.1 is currently in the alpha stage, and binaries are available for download at `http://www.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 C.2 [News-4.1.x], page 1043.

For those wishing to use the most recent development source for MySQL 4.1, we make our 4.1 BitKeeper repository publicly available. See Section 2.3.3 [Installing source tree], page 104.

1.5.3 MySQL 5.0: The Next Development Release

New development for MySQL is focused on the 5.0 release, featuring stored procedures and other new features. See Section 1.6.2 [TODO MySQL 5.0], page 26.

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 2.3.3 [Installing source tree], page 104. As of December 2003, binary builds of version 5.0 are also available.

1.6 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.

1.6.1 New Features Planned for 4.1

The following features are not yet implemented in MySQL 4.1, but are planned for implementation before MySQL 4.1 moves into its beta phase. For a list what is already done in MySQL 4.1, see Section 1.5.2.1 [Nutshell 4.1 features], page 23.

- Stable OpenSSL support (MySQL 4.0 supports rudimentary, not 100% tested, support for OpenSSL).
- More testing of prepared statements.
- More testing of multiple character sets for one table.

1.6.2 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 are complete and are included in MySQL 5.0 alpha, which is available now. 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 already done in MySQL 4.1, see Section 1.5.2.1 [Nutshell 4.1 features], page 23.

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 2.3.3 [Installing source tree], page 104. As of December 2003, binary builds of version 5.0 are also available.

Stored Procedures

- Stored procedures are currently implemented, based on the SQL:2003 standard. See Chapter 19 [Stored Procedures], page 845.

New functionality

- Elementary cursor support. See Section 19.1.8 [Cursors], page 852.
- 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 `HEAP` tables.

Standards compliance, portability and migration

- Add true `VARCHAR` support (column lengths longer than 255, and no stripping of trailing whitespace). (There is already support for this in the `MyISAM` storage engine, but it is not yet available at the user level.)

Speed enhancements

- `SHOW COLUMNS FROM tbl_name` (used by `mysql` client to allow expansions of column names) should not open the table, only the definition file. This will require less memory and 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` (`HEAP`) 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 C.1 [News-5.0.x], page 1040.

1.6.3 New Features Planned for 5.1

New functionality

- `FOREIGN KEY` support for all table types, not just `InnoDB`.
- Column-level 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 will enable us to do faster queries of table structures and do more efficient foreign key support.
- Optimize the `BIT` type to take one bit. (`BIT` now takes one byte; it is treated as a synonym for `TINYINT`.)

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:
  - Create the new database.
  - For every table, do a rename of the table to another database, as we do with the `RENAME` command.
  - Drop the old database.
- New internal file interface change. This will make all file handling much more general and make it easier to add extensions like RAID.

1.6.4 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.
- Add `SUM(DISTINCT)`.
- `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 TABLE 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 example: `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` fields.
- 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:

  ```
  LOAD DATA INFILE 'file_name.txt' INTO TABLE tbl_name
            TEXT_FIELDS (text_field1, text_field2, text_field3)
            SET table_field1=CONCAT(text_field1, text_field2),
                table_field3=23
            IGNORE text_field3
  ```

  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 the `tbl_name1` table.
- `DELETE` and `REPLACE` options to the `UPDATE` statement (this will delete 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

- Don't add automatic `DEFAULT` values to columns. Produce an error for any `INSERT` statement that is missing a value for a column that has no `DEFAULT`.
- 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 0 values 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 ... 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 parse out 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.

Internationalization

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 SQL 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. Records reads and updates. Selects on a single table and selects with joins. Mean number of tables in select. 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->string->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.

1.6.5 New Features Planned for the Mid-Term Future

- Implement function: get_changed_tables(timeout,table1,table2,...).
- Change reading through tables to use memmap when possible. Now only compressed tables use memmap.
- Make the automatic timestamp code nicer. Add timestamps to the update log with SET TIMESTAMP=#;.
- Use read/write mutex in some places to get more speed.
- Simple views (implemented in stepwise fashion up to full functionality). See Section 1.8.5.6 [ANSI diff Views], page 49.
- 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 = expression comparisons. At the moment, only key = field 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 [in 4.0] and LEFT|RIGHT OUTER JOIN are supported.)
- Allow SQL_OPTION MAX_SELECT_TIME=#, 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 now).
- 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.

1.6.6 New Features We Don't Plan to Implement

We aim toward full compliance with ANSI/ISO SQL, so there are no features we plan not to implement.

1.7 MySQL Information Sources

1.7.1 MySQL Mailing Lists

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

1.7.1.1 The MySQL Mailing Lists

To subscribe to or unsubscribe from any of the mailing lists described in this section, visit `http://lists.mysql.com/`. 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.

`mysql-digest`
This is the `mysql` list in digest form. Subscribing to this list means you will get all list messages, sent as one large mail message once a day.

`bugs` This list will be of interest to you if you want to stay informed about issues reported since the last release of `MySQL` or if you want to be actively involved in the process of bug hunting and fixing. See Section 1.7.1.3 [Bug reports], page 34.

`bugs-digest`
This is the `bugs` list in digest form.

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

`internals-digest`
This is the `internals` list in digest form.

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

`mysqldoc-digest`
This is the `mysqldoc` list in digest form.

`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.

`benchmarks-digest`
This is the `benchmarks` list in digest form.

`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.

`packagers-digest`
This is the `packagers` list in digest form.

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

`java-digest`
This is the `java` list in digest form.

`win32` This list is for all topics concerning the MySQL software on Microsoft operating systems, such as Windows 9x, Me, NT, 2000, and XP.

`win32-digest`
This is the `win32` list in digest form.

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

`myodbc-digest`
This is the `myodbc` list in digest form.

`mysqlcc` This list is for all topics concerning the `MySQL Control Center` graphical client.

`mysqlcc-digest`
This is the `mysqlcc` list in digest form.

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

`plusplus-digest`
This is the `plusplus` list in digest form.

`msql-mysql-modules`
This list is for all topics concerning the Perl support for MySQL with `msql-mysql-modules`, which is now named `DBD::mysql`.

`msql-mysql-modules-digest`
This is the `msql-mysql-modules` list in digest form.

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 will put you in direct contact with MySQL developers. See Section 1.4.1 [Support], page 16.

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.

1.7.1.2 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://www.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 appendix (`http://www.mysql.com/doc/en/News.html`) can be particularly useful since it is quite possible that a newer version already contains a solution to your problem.
- Search in the bugs database at `http://bugs.mysql.com/` to see whether the bug has already 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.

1.7.1.3 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 will also be able to 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 will help 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` 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/` will be corrected or documented in the next MySQL release. If only minor code changes are needed to correct a problem, we will also post a patch that fixes the problem.

If you have found a sensitive security bug in MySQL, please send an 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 already 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 D.1.6 [Reproduceable test case], page 1204.

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 case should be observed.) You should never try to remember 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 MyODBC trace file and send it with your report. See Section 20.3.7 [MyODBC bug report], page 965.

Please remember that many of the people who will read your report will 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 Version 4.0.12). You can find out which version you are running by executing `mysqladmin version`. `mysqladmin` 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 logging enabled. See Section D.1.5 [Using log files], page 1203.
- If a database table is related to the problem, include the output from `mysqldump --no-data db_name tbl_name1 tbl_name2 ...`. This is very easy to do and is a powerful way to get information about any table in a database. The information will help us create a situation matching the one you have.
- 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 will reproduce 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 `http://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://support.mysql.com/pub/mysql/secret/`. 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 you don't want to show to others, you can use `ftp` to transfer it to `ftp://support.mysql.com/pub/mysql/secret/`. If the data 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 will 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 will help us here. Show that the patch will handle 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://www.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 C [News], page 1040.
- 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 [MySQL Database Administration], page 218.

  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 5.8.1 [Error log], page 345. 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 [What is crashing], page 998.

- 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 [Which version], page 62.

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 20.3.4 [ODBC Problems], page 960.

For solutions to some common problems, see Appendix A [Problems], page 998.

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.

1.7.1.4 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.

1.7.2 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.
- **EFnet** (see `http://www.efnet.org/` for servers)
  - `#mysql` MySQL questions.

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.

1.8 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 will find out what they are and how to use them. You will also find information about functionality missing from MySQL Server, and how to work around some differences.

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 14.1.3 [`HANDLER`], page 628.)

We will 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.

We are currently not targeting realtime support, although the MySQL replication capabilities already offer significant functionality.

Database cluster support is planned through integration of our acquired NDB Cluster technology into a new storage engine, available early 2004.

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

1.8.1 What Standards MySQL Follows

ODBC levels 0-3.51.

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

1.8.2 Selecting SQL Modes

The MySQL server can operate in different SQL modes, and can apply these modes differentially for different clients. This allows applications to tailor server operation to their 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 [Server SQL mode], page 237.

1.8.3 Running MySQL in ANSI Mode

You can tell mysqld to use the ANSI mode with the --ansi startup option. See Section 5.2.1 [Server options], page 228.

Running the server in ANSI mode is the same as starting it with these options:

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

In MySQL 4.1, you can achieve the same effect with these two statements:

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

See Section 1.8.2 [SQL mode], page 40.

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

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

In this case, the value of the sql_mode variable will be set to all options that are relevant for ANSI mode. You can check the result by doing:

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

1.8.4 MySQL Extensions to the SQL Standard

MySQL Server includes some extensions that you probably will not find in other SQL databases. Be warned that if you use them, your code will not 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 will parse 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 will be 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 will use 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 10.2.2 [Name case sensitivity], page 497.
- Database, table, index, column, or alias names may begin with a digit (but may not consist solely of digits).
- 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.

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 14.2.3 [CREATE DATABASE], page 668.
- The `DO` statement.
- `EXPLAIN SELECT` to get a description of how tables are joined.
- The `FLUSH` and `RESET` statements.
- The `SET` statement. See Section 14.5.3.1 [SET], page 702.
- The `SHOW` statement. See Section 14.5.3 [SHOW], page 701.
- Use of `LOAD DATA INFILE`. In many cases, this syntax is compatible with Oracle's `LOAD DATA INFILE`. See Section 14.1.5 [LOAD DATA], page 635.
- Use of `RENAME TABLE`. See Section 14.2.9 [RENAME TABLE], page 683.
- Use of `REPLACE` instead of `DELETE` + `INSERT`. See Section 14.1.6 [REPLACE], page 642.
- 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 14.2.2 [ALTER TABLE], page 663.
- Use of index names, indexes on a prefix of a field, and use of `INDEX` or `KEY` in a `CREATE TABLE` statement. See Section 14.2.5 [CREATE TABLE], page 670.
- 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 14.1.7 [SELECT], page 643.
- 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 13.9 [Group by functions and modifiers], page 618.
- 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. See Section 13.8.3 [Information functions], page 611.
- 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(), or 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 13.9 [Group by functions and modifiers], page 618.

For a prioritized list indicating when new extensions will be added to MySQL Server, you should consult the online MySQL TODO list at http://www.mysql.com/doc/en/TODO.html. That is the latest version of the TODO list in this manual. See Section 1.6 [TODO], page 26.

1.8.5 MySQL Differences Compared to 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. See Section 1.8.7 [Bugs], page 52.
- In some cases, CHAR columns are silently converted to VARCHAR columns when you define a table or alter its structure. See Section 14.2.5.1 [Silent column changes], page 681.
- 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 14.5.1.2 [GRANT], page 689.

1.8.5.1 Subqueries

MySQL Version 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 14.1.8 [Subqueries], page 652.

For MySQL versions older than 4.1, most subqueries can be rewritten using joins or other methods. See Section 14.1.8.11 [Rewriting subqueries], page 660 for examples that show how to do this.

1.8.5.2 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 14.1.4.1 [INSERT SELECT], page 633.

```
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 19.1.6.3 [SELECT INTO Statement], page 850.

1.8.5.3 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 15 [Table types], page 737.

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 you never will get an automatic abort 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 will stall until any 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 new inserted records will 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 14.1.4.2 [INSERT DELAYED], page 633.

"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 will never be an automatic rollback (which can happen with transactional tables if you are not very careful). MySQL Server also guarantees that there will not be any 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 will be 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 fields relative to their current value.
  - Update only those fields 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_he_owes_us=money_he_owes_us-125
  WHERE
    customer_id=id AND address='old address' AND phone='old phone';
```

As you can see, this is very efficient and works even if another client has changed the values in the `pay_back` or `money_he_owes_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 SQL function `LAST_INSERT_ID()` or the C API function `mysql_insert_id()`. See Section 13.8.3 [Information functions], page 611. See Section 20.2.3.32 [`mysql_insert_id()`], page 886.

  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 already 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;
  ```

1.8.5.4 Stored Procedures and Triggers

Stored procedures are implemented in MySQL version 5.0. See Chapter 19 [Stored Procedures], page 845.

Triggers are scheduled for implementation in MySQL version 5.1. A "trigger" is effectively a type of stored procedure, one that is invoked when a particular event occurs. For example, you could set up a stored procedure that is triggered each time a record is deleted from a transactional table, and that stored procedure automatically deletes the corresponding customer from a customer table when all their transactions are deleted.

1.8.5.5 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 16.7.4 [InnoDB foreign key constraints], page 772.

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], page 423.)

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 14.1.1 [DELETE], page 626.
- 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 16.7.4 [InnoDB foreign key constraints], page 772. 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 table1,table2 WHERE table1.id = table2.id;
```

See Section 14.1.7.1 [JOIN], page 648. See Section 3.6.6 [example-Foreign keys], page 201.

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

1.8.5.6 Views

Views are currently being implemented, and will appear in the 5.0 or 5.1 version of MySQL Server. Unnamed views (*derived tables*, a subquery in the `FROM` clause of a `SELECT`) are already implemented in version 4.1.

Historically, MySQL Server has been most used in applications and on Web systems where the application writer has full control over database usage. Usage has shifted over time, and so we find that an increasing number of users now regard views as an important feature.

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). One does not require views to restrict access to columns, because MySQL Server has a sophisticated privilege system. See Section 5.4 [Privilege system], page 275.

Many DBMS don't allow updates to a view. Instead, you have to perform the updates on the individual tables. In designing an implementation of views, our goal, as much as is possible within the confines of SQL, is full compliance with "**Codd's Rule #6**" for relational database systems: All views that are theoretically updatable, should in practice also be updatable.

1.8.5.7 '--' 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 10.5 [Comments], page 503.

MySQL Server Version 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 tbl_name SET credit=credit-!payment!
```

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

```
UPDATE tbl_name 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 tbl_name 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 Version 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
```

1.8.6 How MySQL Deals with Constraints

MySQL allows you to work with both transactional tables that allow rollback and non-transactional tables that do not, so constraint handling is a bit different in MySQL than in other databases.

We have to handle the case when you have updated a lot of rows in a non-transactional table that cannot roll back when an error occurs.

The basic philosophy is to try to give an error for anything that we can detect at compile time but try to recover from any errors we get at runtime. We do this in most cases, but not yet for all. See Section 1.6.4 [TODO future], page 28.

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.

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

1.8.6.1 Constraint PRIMARY KEY / UNIQUE

Normally you will get an error 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 will automatically roll back the transaction. If you are using a non-transactional storage engine, MySQL will stop at the incorrect row and leave any remaining rows unprocessed.

To make life easier, MySQL supports an IGNORE keyword for most commands that can cause a key violation (such as INSERT IGNORE and UPDATE IGNORE). In this case, MySQL will ignore any key violation and continue with processing the next row. You can get information about what MySQL did with the mysql_info() API function. See Section 20.2.3.30 [mysql_info()], page 885. In MySQL 4.1 and up, you also can use the SHOW WARNINGS statement. See Section 14.5.3.20 [SHOW WARNINGS], page 719.

Note that, for the moment, only InnoDB tables support foreign keys. See Section 16.7.4 [InnoDB foreign key constraints], page 772. Foreign key support in MyISAM tables is scheduled for implementation in MySQL 5.1.

1.8.6.2 Constraint NOT NULL and DEFAULT Values

To be able to support easy handling of non-transactional tables, all columns in MySQL have default values.

If you insert an "incorrect" value in a column, such as a NULL in a NOT NULL column or a too-large numerical value in a numerical column, MySQL sets the column to the "best possible value" instead of producing 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 in the column.

This means that if you try to store NULL into a column that doesn't take NULL values, MySQL Server instead stores 0 or '' (the empty string). This last behavior can, for single-row inserts, be changed with the -DDONT_USE_DEFAULT_FIELDS compile option.) See Section 2.3.2 [configure options], page 101. This causes INSERT statements to generate an error unless you explicitly specify values for all columns that require a non-NULL value.

The reason for the preceding rules is that we can't check these conditions until the query has begun executing. We can't just roll back if we encounter a problem after updating a few rows, because the table type 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.

This means that you should generally not use MySQL to check column content. Instead, the application should ensure that is passes only legal values to MySQL.

In MySQL 5.0, we plan to improve this by providing warnings when automatic column conversions occur, plus an option to let you roll back statements that attempt to perform a disallowed column value assignment, as long as the statement uses only transactional tables.

1.8.6.3 Constraint ENUM and SET

In MySQL 4.x, ENUM is not a real constraint, but is a more efficient way to define columns that can contain only a given set of values. This is because of the same reasons NOT NULL is not honored. See Section 1.8.6.2 [constraint NOT NULL], page 51.

If you insert an incorrect value into an ENUM column, it will be set to the reserved enumeration value 0, which will be displayed as an empty string in string context. See Section 12.4.3 [ENUM], page 550.

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 12.4.4 [SET], page 551.

1.8.7 Known Errors and Design Deficiencies in MySQL

1.8.7.1 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 can get a deadlock 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 in 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 in MySQL 4.0.11.
- DELETE FROM heap_table without a WHERE clause doesn't work on a locked HEAP table.

1.8.7.2 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 a UNION, the first SELECT determines the type, max_length, and NULL properties for the resulting columns. This issue is resolved in 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 in 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.

1.8.7.3 Open Bugs and Design Deficiencies in MySQL

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

- Even if you are using `lower_case_table_names=2` (which enables MySQL to remember the used case for databases and table names) MySQL will not on case insensitive systems remember the used case for database names for the function `DATABASE()` or in various logs.
- 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 `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.
- `GROUP_CONCAT()` doesn't work with `BLOB`/`TEXT` columns when you use `DISTINCT` or `ORDER BY` inside `GROUP_CONCAT()`. To work around this limitation, use `MID(expr, 1, 255)` instead.
- 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 in 5.0 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. It is planned to be fixed in 4.1.
- `FLUSH TABLES WITH READ LOCK` does not block `CREATE TABLE` or `COMMIT`, which may cause a problem with the binary log position when doing a full backup of tables and the binary log.
- `ANALYZE TABLE` on a `BDB` table may in some cases make the table unusable until you restart `mysqld`. If this happens, you will see errors of the following form in the MySQL error file:

  ```
  001207 22:07:56  bdb:  log_flush: LSN past current end-of-log
  ```
- MySQL accepts parentheses in the `FROM` part of a `SELECT` statement, but silently ignores them. The reason for not giving an error is that many clients that automatically generate queries add parentheses in the `FROM` part even where they are not needed.
- Concatenating many `RIGHT JOINS` or combining `LEFT` and `RIGHT` join in the same query may not give a correct answer because MySQL only generates `NULL` rows for the table preceding a `LEFT` or before a `RIGHT` join. This will be fixed in 5.0 at the same time we add support for parentheses in the `FROM` part.
- Don't execute `ALTER TABLE` on a `BDB` table on which you are running multiple-statement transactions until all those transactions complete. (The transaction will probably 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 will take 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. Although we have never heard of it actually occurring, it is theoretically 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 which 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 will result 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 will be 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 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 files are based on hostnames (if you don't specify a file name with the startup option). For now 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. See Section 5.2.1 [Server options], page 228.

- `mysqlbinlog` will not delete temporary files left after a `LOAD DATA INFILE` command. See Section 8.5 [`mysqlbinlog`], page 470.
- `RENAME` doesn't work with `TEMPORARY` tables or tables used in a `MERGE` table.
- When using the `RPAD()` function in a query that has to be resolved by using a temporary table, all resulting strings will have rightmost spaces removed. This is an example of such a query:

```
SELECT RPAD(t1.column1, 50, ' ') AS f2, RPAD(t2.column2, 50, ' ') AS f1
FROM table1 as t1 LEFT JOIN table2 AS t2 ON t1.record=t2.joinID
ORDER BY t2.record;
```

  The final result of this bug is that you will not be able to get spaces on the right side of the resulting values. The problem also occurs for any other string function that adds spaces to the right.

  The reason for this is due to the fact that `HEAP` tables, which are used first for temporary tables, are not capable of handling `VARCHAR` columns.

  This behavior exists in all versions of MySQL. It will be fixed in one of the 4.1 series releases.

- 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 with a number stored in different formats (+01.00, 1.00, 01.00), `GROUP BY` may regard each value as a different value.
- `DELETE FROM merge_table` used without a `WHERE` clause will clear only the mapping for the table, not delete everything in the mapped tables.
- 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 2.3.5 [MIT-pthreads], page 110.
- `BLOB` 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. A workaround for most cases is to use a substring. For example: `SELECT DISTINCT LEFT(blob,2048) FROM tbl_name`.
- Numeric calculations are done with `BIGINT` or `DOUBLE` (both are normally 64 bits long). It depends on the function which precision one gets. 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 only have 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 '''hostname'.err' instead of '''hostname'.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 will get the updated value instead of the original value. For example:

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

  This will increment KEY by 2, not 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 temporary_table, temporary_table AS t2;
  ```

- The optimizer may handle DISTINCT differently if you are using "hidden" columns in a join or 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 may 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 will actually give 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 will, instead of doing a rollback, store the `best possible value` in the column:

- If you try to store a value outside the range in a numerical column, MySQL Server instead stores the smallest or largest possible value in the column.
- If you try to store a string that doesn't start with a number into a numerical column, MySQL Server stores 0.
- If you try to store `NULL` into a column that doesn't allow `NULL` values, MySQL Server stores 0 or `''` (the empty string) in it instead. (This behavior can, however, be changed with the `-DDONT_USE_DEFAULT_FIELDS` compile option.)
- MySQL allows you to store some wrong date values into `DATE` and `DATETIME` columns (like `'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 you set an `ENUM` column to an unsupported value, it is set to the error value `empty string`, with numeric value 0.
- If you set a `SET` column to an unsupported value, the value is ignored.

- If you execute a `PROCEDURE` on a query that returns an empty set, in some cases the `PROCEDURE` will not transform the columns.
- Creation of a table of type `MERGE` doesn't check whether the underlying tables are of compatible types.
- MySQL Server can't yet handle `NaN`, `-Inf`, and `Inf` values in `DOUBLE` columns. Using these will cause problems when trying to export and import data. We should, as an intermediate solution, change `NaN` to `NULL` (if possible) and `-Inf` and `Inf` to the minimum respective maximum possible `double` value.
- If you use `ALTER TABLE` to first add a `UNIQUE` index to a table used in a `MERGE` table and then use `ALTER TABLE` to add a normal index on the `MERGE` table, the key order will be 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:

- You can get a hung thread if you do a `DROP TABLE` on a table that is one among many tables that is locked with `LOCK TABLES`.
- 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 Version 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:

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

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

- Before MySQL Server Version 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 platform-specific bugs, see the sections about compiling and porting. See Section 2.3 [Installing source], page 98. See Appendix D [Porting], page 1198.

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 [Which OS], page 60 for details.
2. **Choose a 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 [Which version], page 62.
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 [Getting MySQL], page 73. You can verify the integrity of the distribution using the instructions in Section 2.1.4 [Verifying Package Integrity], page 73.
4. **Install the distribution.** For binary distributions, use the instructions in in Section 2.2.5 [Installing binary], page 95. For source distributions, use the instructions in Section 2.3 [Installing source], page 98. Additional installation procedures include the following:
   - For post-installation procedures, see Section 2.4 [Post-installation], page 115. These procedures apply whether you install MySQL using a binary or source distribution.
   - If you plan to upgrade an existing version of MySQL to a newer version rather than installing MySQL for the first time, see Section 2.5 [Upgrade], page 126 for information about upgrade procedures and about issues that you should consider before upgrading.
   - If you want to run the MySQL benchmark scripts, Perl support for MySQL must be available. See Section 2.7 [Perl support], page 166.

The last part of the chapter provides information on system-specific problems you may run into.

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 2.6.5.3 [IBM-AIX], page 158.
- Amiga.
- BSDI 2.x with the MIT-pthreads package. See Section 2.6.4.5 [BSDI], page 155.
- BSDI 3.0, 3.1 and 4.x with native threads. See Section 2.6.4.5 [BSDI], page 155.
- DEC UNIX 4.x with native threads. See Section 2.6.5.5 [Alpha-DEC-UNIX], page 160.
- FreeBSD 2.x with the MIT-pthreads package. See Section 2.6.4.1 [FreeBSD], page 153.
- FreeBSD 3.x and 4.x with native threads. See Section 2.6.4.1 [FreeBSD], page 153.
- FreeBSD 4.x with LinuxThreads. See Section 2.6.4.1 [FreeBSD], page 153.
- HP-UX 10.20 with the DCE threads or the MIT-pthreads package. See Section 2.6.5.1 [HP-UX 10.20], page 156.
- HP-UX 11.x with the native threads. See Section 2.6.5.2 [HP-UX 11.x], page 157.
- Linux 2.0+ with LinuxThreads 0.7.1+ or `glibc` 2.0.7+. See Section 2.6.1 [Linux], page 141.
- Mac OS X. See Section 2.6.2 [Mac OS X], page 149.
- NetBSD 1.3/1.4 Intel and NetBSD 1.3 Alpha (requires GNU make). See Section 2.6.4.2 [NetBSD], page 154.
- Novell NetWare 6.0. See Section 2.2.4 [NetWare installation], page 93.
- OpenBSD `>` 2.5 with native threads. OpenBSD `<` 2.5 with the MIT-pthreads package. See Section 2.6.4.3 [OpenBSD], page 155.
- OS/2 Warp 3, FixPack 29 and OS/2 Warp 4, FixPack 4. See Section 2.6.6 [OS/2], page 165.
- SCO OpenServer with a recent port of the FSU Pthreads package. See Section 2.6.5.8 [SCO], page 164.
- SCO UnixWare 7.1.x. See Section 2.6.5.9 [SCO UnixWare], page 165.
- SGI Irix 6.x with native threads. See Section 2.6.5.7 [SGI-Irix], page 163.
- Solaris 2.5 and above with native threads on SPARC and x86. See Section 2.6.3 [Solaris], page 149.
- SunOS 4.x with the MIT-pthreads package. See Section 2.6.3 [Solaris], page 149.
- Tru64 Unix
- Windows 9x, Me, NT, 2000, and XP. See Section 2.2.1 [Windows installation], page 77.

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 will be only as stable as the thread library if that library is unstable in the code that is called by MySQL, even if everything else is perfect.
- The capability of the kernel and/or 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/or the thread library to run many threads which acquire and release a mutex over a short critical region frequently without excessive context switches. In other words, if the implementation of `pthread_mutex_lock()` is too anxious to yield CPU time, this will hurt MySQL tremendously. If this issue is not taken care of, adding extra CPUs will actually make 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 that 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 8.2, 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 will join the top club once the thread library is improved. We also hope that at some point we will be 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 will require 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 1.7.1.1 [Mailing-list], page 32.

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. And in some cases, the reason one OS is better than the other could simply be that we have put forth more effort into testing on and optimizing for that particular platform. We are just stating our observations to help you decide which platform to use MySQL in your setup.

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. Until recently it was available only in preview form from the BitKeeper source repository. An early alpha release has now been issued to allow more widespread testing.
- MySQL 4.1 is a development release series to which major new features have been added. It is still at alpha status. Sources and binaries are available for use and testing on development systems.
- MySQL 4.0 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 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 will not affect anything that's already 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 4.0. Note that 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 chances with 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 will be based.

The MySQL naming scheme uses release names that consist of three numbers and a suffix, for example, `mysql-4.1.0-alpha`. The numbers within the release name are is 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 (`0`) 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 C [News], page 1040. 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 will be 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) 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.

Note that 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 22.1.2 [MySQL test suite], page 984 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 [MySQL Benchmarks], page 399.

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 [MySQL Benchmarks], page 399.

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 both MySQL versions.

  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 [`mysqld-max`], page 219.

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

Circumstances under which you probably will be better off with a source installation include the following:

- 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 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 onwards)
  - `--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 will be committed to the source repository and you can access it there. The bugfix will 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 also 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 extended email-supported customers want and try to help them out.

No one has to download a new release. The News section will tell you if the new release has something you really want. See Appendix C [News], page 1040.

We use the following policy when updating MySQL:

- Releases are issued within each release 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, but if small bugs are found, a release with only bugfixes will be 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 usually 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 will 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 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 1.8.7 [Bugs], page 52.

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.

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 will never recur without being detected. (About 90% of all fixed bugs have a test case.)
- We also 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 1.8.7 [Bugs], page 52.
- 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 will not publish a binary for a platform for which the test or benchmark suite fails. If it's a general error in the source, we fix this 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://www.mysql.com/`, we send out an announcement message to the `mysql` and `announce` mailing lists. See Section 1.7.1.1 [Mailing-list], page 32. 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 in 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, after the release is done, we get any bug reports that there was (after all) anything critically wrong with the build on a specific platform, we will fix this 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 good 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, we at MySQL AB 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 (see Section 2.2 [Quick Standard Installation], page 77), we do offer binary distributions for a number of platforms in the form of of compressed tar files (`.tar.gz`).

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.

Binaries 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.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 --enable-assembler --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. Please note that these are provided only as a courtesy. Since MySQL AB does not have full control over these systems, we can provide only limited support for the binaries built on these systems.

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-prefix=/usr/local/mysql --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 MySQL AB provided in the past. These binaries are no longer 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 2 4 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 1.7.1.1 [Mailing-list], page 32.

RPM distributions prior to MySQL Version 3.22 are user-contributed. Beginning with Version 3.22, the RPM distributions are generated by us at 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.

For the Windows distribution, please see Section 2.2.1 [Windows installation], page 77.

2.1.3 How to Get MySQL

Check the MySQL home page (`http://www.mysql.com/`) for information about the current version and for downloading instructions.

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

For a complete up-to-date list of MySQL Web/download mirrors, see `http://www.mysql.com/downloads/mirrors.html`. There you will also find information about becoming a MySQL mirror site and how to report a bad or out-of-date mirror.

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.

Verifying the `MD5 Checksum`

After you have downloaded the package, you should make sure that the 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
```

Note that not all operating systems support the `md5sum` command—on some it is simply called `md5`, 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 command `openssl md5 package_name` instead. A DOS/Windows implementation of the `md5` command is available from `http://www.fourmilab.ch/md5/`.

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.

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 already 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 build@mysql.com. You can either cut and paste it directly from here, or obtain it from http://www.keyserver.net/.

```
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+owwRBACl4GIfUfCyEDSIePvEW3SAFUdJBtoQHH/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 this key into your personal public GPG keyring by using `gpg --import`. See the GPG documentation for more info 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.

Signature Checking Using RPM

For RPM packages, there is no separate signature. RPM packages actually 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 it into your own GPG public keyring), you need to import the key into the RPM keyring first. RPM 4.1 no longer uses your personal GPG keyring (and GPG itself), but rather maintains its own keyring (because it's 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, use `rpm --import`.

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 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.

2.1.5 Installation Layouts

This section describes the default layout of the directories created by installing binary and source distributions.

On Windows, the default installation directory is 'C:\mysql', which 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
'/usr/sbin'	mysqld server
'/var/lib/mysql'	Log files, databases
'/usr/share/doc/packages'	Documentation
'include'	Include (header) files
'lib'	Libraries
'scripts'	mysql_install_db
'/usr/share/mysql'	Error message and character set files
'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

This section covers 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 2.2.5 [Installing binary], page 95 for generic installation instructions for these packages that apply to all platforms.

See Section 2.1 [General Installation Issues], page 59 for more information on what other binary distributions are available and how to obtain them.

2.2.1 Installing MySQL on Windows

The installation process for MySQL on Windows has the following steps:

1. Install the distribution.
2. Set up an option file if necessary.

3. Select the server you want to use.
4. Start the server.
5. Assign passwords to the initial MySQL accounts.

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 2.3.6 [Windows source build], page 111.

2.2.1.1 Windows System Requirements

To run MySQL on Windows, you need the following:

- A 32-bit Windows operating system such as 9x, Me, NT, 2000, or XP. The NT family (Windows NT, 2000, and XP) permits you to run the MySQL server as a service. See Section 2.2.1.7 [NT start], page 83.
- TCP/IP protocol support.
- A copy of the MySQL binary distribution for Windows, which can be downloaded from `http://www.mysql.com/downloads/`.

  Note: The distribution files are supplied with a zipped format and we recommend the use of an adequate FTP client with resume feature to avoid corruption of files during the download process.
- A `ZIP` program to unpack the distribution file.
- Enough space on the hard drive to unpack, install, and create the databases in accordance with your requirements.
- If you plan to connect to the MySQL server via ODBC, you also need the `MyODBC` driver. See Section 20.3 [ODBC], page 957.
- 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 14.2.5 [`CREATE TABLE`], page 670.

2.2.1.2 Installing a Windows Binary Distribution

To install MySQL on Windows using a binary distribution, follow this procedure:

1. If you are working on a Windows NT, 2000, or XP machine, make sure that you have logged in as a user with administrator privileges.
2. If you are doing an upgrade of an earlier MySQL installation, it is necessary to stop the current server. On Windows NT, 2000, or XP machines, if you are running the server as a Windows service, stop it as follows from the command prompt:

```
C:\> NET STOP MySQL
```

If you plan to use a different server after the upgrade (for example, if you want to run `mysqld-max` rather than `mysqld`), remove the existing service:

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

You can reinstall the service to use the proper server after upgrading.

If you are not running the MySQL server as a service, stop it like this:

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

3. Exit the `WinMySQLAdmin` program if it is running.
4. Unzip the distribution file to a temporary directory.
5. Run the `setup.exe` program to begin the installation process. If you want to install MySQL into a location other than the default directory ('`C:\mysql`'), use the `Browse` button to specify your preferred directory. If you do not install MySQL into the default location, you will need to specify the location whenever you start the server. The easiest way to do this is to use an option file, as described in Section 2.2.1.3 [Windows prepare environment], page 79.
6. Finish the install process.

Important note: Early alpha Windows distributions for MySQL 4.1 do not contain an installer program. A 4.1 distribution is a ZIP file that you just unzip in the location where you want to install MySQL. For example, to install '`mysql-4.1.1-alpha-win.zip`' as '`C:\mysql`', unzip the distribution file on the `C:` drive, then rename the resulting '`mysql-4.1.1-alpha`' directory to '`mysql`'.

If you are upgrading to MySQL 4.1 from an earlier version, you will want to preserve your existing '`data`' directory that contains the grant tables in the `mysql` database and your own databases. Before installing 4.1, stop the server if it is running, and save your '`data`' directory to another location. Then either rename the existing '`C:\mysql`' directory or remove it. Install 4.1 as described in the preceding paragraph, and then replace its '`data`' directory with your old '`data`' directory. Start the new server and update the grant tables. This will avoid the loss of your current databases. See Section 2.5.8 [Upgrading-grant-tables], page 139.

2.2.1.3 Preparing the Windows MySQL Environment

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 will be used every time the server starts, you will find it most convenient to use an option file to specify your MySQL configuration. This is true particularly 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 version 3.23, you must manually create two new directories to hold the `InnoDB` data and log files—such as, '`C:\ibdata`' and '`C:\iblogs`'. You will also need to add some extra lines to the option file, as described in Section 16.4 [`InnoDB` configuration], page 759. (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 will be useful in this case, too.)

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.

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 one you use, it must be a plain text 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 'D:\mysql' and the data directory is located at 'D:\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=D:/mysql
# set datadir to the location of your data directory
datadir=D:/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.

Another way to manage an option file is to use the WinMySQLAdmin tool. You can find WinMySQLAdmin in the 'bin' directory of your MySQL installation, as well as a help file containing instructions for using it. WinMySQLAdmin has the capability of editing your option file, but note these points:

- WinMySQLAdmin uses only the 'my.ini' file.
- If WinMySQLAdmin finds a 'C:\my.cnf' file, it will in fact rename it to 'C:\my_cnf.bak' to disable it.

Now you are ready to test starting the server.

2.2.1.4 Selecting a Windows Server

Starting with MySQL 3.23.38, the Windows distribution includes both the normal and the MySQL-Max server binaries. Here is a list of the different MySQL servers from which you can choose:

Binary	**Description**
`mysqld`	Compiled with full debugging and automatic memory allocation checking, symbolic links, 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 symbolic links, and `InnoDB` and `BDB` tables.
`mysqld-max-nt`	Like `mysqld-max`, but compiled with support for named pipes.

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

MySQL supports TCP/IP on all Windows platforms. The `mysqld-nt` and `mysql-max-nt` servers support named pipes on NT, 2000, and XP. 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 now 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).
- These servers can be run on Windows 98 or Me, but only if TCP/IP is installed; named pipe connections cannot be used.
- On Windows 95, these servers cannot be used.

2.2.1.5 Starting the Server for the First Time

On Windows 95, 98, or Me, MySQL clients always connect to the server using TCP/IP. (This will allow 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 `http://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, or XP, clients have two options. They can use TCP/IP, or they can use a named pipe if the server supports named pipe connections.

For information about which server binary to run, see Section 2.2.1.3 [Windows prepare environment], page 79.

This section gives a general overview of starting the MySQL server. The following sections provide more specific information for particular versions of Windows.

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 will make it easier for you to identify and fix any problems.

Make sure that you are in the directory where the server is located, then enter this command:

```
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 will continue 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. The error log is the file with the '.err' extension.

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 2.4 [Post-installation], page 115.

2.2.1.6 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:

```
shell> C:\mysql\bin\mysqld
```

On non-NT versions of Windows, this will start `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, or XP, the server will run in the foreground and no command prompt will appear 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:

```
shell> 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 `root`, which is the default administrative account in the MySQL grant system. Please 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` will write a log file '`C:\mysqld.trace`' that should contain the reason why `mysqld` doesn't start. See Section D.1.2 [Making trace files], page 1200.

Use `mysqld --help` to display all the options that `mysqld` understands!

2.2.1.7 Starting MySQL as a Windows Service

On the NT family (Windows NT, 2000, or XP), the recommended way to run MySQL is to install it as a Windows service. Then Windows starts and stops the MySQL server automatically when Windows starts and stops. A 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). It is advisable to close the `Services` utility while performing server installation or removal operations from this command line. This prevents some odd errors.

To get MySQL to work with TCP/IP on Windows NT 4, you must install service pack 3 (or newer)!

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

```
shell> 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 `root`, which is the default administrative account in the MySQL grant system. Please note that users in the MySQL grant system are wholly independent from any login users under Windows.

Now install the server as a service:

```
shell> mysqld --install
```

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

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

As of MySQL 4.0.2, you can specify a specific service name after the `--install` option. As of MySQL 4.0.3, you can in addition specify a `--defaults-file` option after the service name to indicate where the server should obtain options when it starts. The rules that determine the service name and option files the server uses are as follows:

- 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: 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, avoid installing MySQL in a directory such as '`C:\Program Files`' or using a service name containing spaces.

In the usual case that you install the server with `--install` but no service name, the server is installed with a service name of `MySQL`.

As a more complex example, consider the following command (which should be entered on a single line):

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

Here, a service name 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 `[mysql]` group from the standard option files. (This would be a bad idea, because that option group is for use by the `mysql` client program.) However, because the `--defaults-file` option is present, the server reads options only from the named file, and only from the `[mysqld]` option group.

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

Once a MySQL server is installed as a service, Windows will start 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.

Please note that 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 'C:\mysql\data' directory. It is the file with a suffix of '.err'.

When mysqld is running as a service, it can be stopped by using the Services utility, the command NET STOP MySQL, or the command mysqladmin shutdown. If the service is running when Windows shuts down, Windows will stop the server automatically.

From MySQL version 3.23.44, 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:

```
shell> 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:

```
shell> 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 had to perform crash recovery at the next startup.) Starting from MySQL version 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 you don't want to start mysqld as a service, you can start it from the command line the same way as for versions of Windows that are not based on NT. For instructions, see Section 2.2.1.6 [Win95 start], page 83.

2.2.1.8 Running MySQL Client Programs on Windows

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. In MySQL 4.1, you should use the --protocol=PIPE option.

There are two versions of the MySQL command-line tool:

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.

If you want to use mysqlc, you must have a copy of the 'cygwinb19.dll' library installed somewhere that mysqlc can find it. Current distributions of MySQL include this library in the same directory as mysqlc (the 'bin' directory under the base directory of your MySQL installation). If your distribution does not have the cygwinb19.dll library in the 'bin' directory, look for it in the lib directory and copy it to your Windows system directory ('\Windows\system' or a similar place).

2.2.1.9 MySQL on Windows Compared to MySQL on Unix

MySQL for Windows has by now 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 will appear to have become 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.

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 1,024, which means that you will not be able to run as many concurrent threads on NT, 2000, and XP as on Unix.

Blocking read

MySQL uses a blocking read for each connection, which has the following implications:

- A connection will not be 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` will 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.

`DROP DATABASE`

You can't drop a database that is in use by some thread.

Killing MySQL from the Task Manager

You can't kill MySQL from the Task Manager or with the shutdown utility in Windows 95. You must take it down 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 10.2.2 [Name case sensitivity], page 497.

The '\' pathname separator character

Pathname components in Windows 95 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 don't work reliably from the Windows command-line prompt. If the pipe includes the character `^Z / CHAR(24)`, Windows will think it has encountered end-of-file and abort 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 get a problem applying the log and suspect it's because of an `^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.

`Can't open named pipe` **error**

If you use a MySQL Version 3.22 server on NT with the newest MySQL client programs, you will get the following error:

```
error 2017: can't open named pipe to host: . pipe...
```

This happens because the release version of MySQL uses named pipes on NT by default. You can avoid this error by using the `--host=localhost` option to the new MySQL clients or create an option file 'C:\my.cnf' that contains the following information:

```
[client]
host = localhost
```

Starting from 3.23.50, named pipes are enabled only if `mysqld-nt` or `mysqld-max-nt` is started with `--enable-named-pipe`.

`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 can't resolve your hostname properly.

To fix this, you should create a file '\windows\hosts' with the following information:

```
127.0.0.1       localhost
```

`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 will 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.

Here are some open issues for anyone who might want to help us improve MySQL on Windows:

- 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. 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.

2.2.2 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.

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 2.6.1.2 [Binary notes-Linux], page 141.

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 install the MySQL-Max RPM. However, you should do so only *after* installing the MySQL-server RPM. See Section 5.1.2 [mysqld-max], page 219.

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 will need this unless you only want to connect to a MySQL server running on another machine. Please 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 please see Section 2.1.4 [Verifying Package Integrity], page 73.

The server RPM places data under the '`/var/lib/mysql`' directory. The RPM also 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 Section 2.4.2.2 [Automatic start], page 122 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; ln -s rc.d/init.d .
```

However, all current major Linux distributions should already 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 now be able to start using MySQL. See Section 2.4 [Post-installation], page 115.

If something goes wrong, you can find more information in the binary installation chapter. See Section 2.2.5 [Installing binary], page 95.

2.2.3 Installing MySQL on Mac OS X

Beginning with MySQL 4.0.11, you can install MySQL on Mac OS X 10.2 ("Jaguar") 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.

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 will guide you through the installation of MySQL.

Due to a bug in the Mac OS X package installer, you may sometimes see the error message `You cannot install this software on this disk. (null)` in the destination disk selection dialog. 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 will install itself into '`/usr/local/mysql-<version>`' and will also install a symbolic link, '`/usr/local/mysql`', pointing to the new location. If a directory named '`/usr/local/mysql`' already exists, it will be renamed to '`/usr/local/mysql.bak`' first. Additionally, it will install the grant tables in the `mysql` database by executing `mysql_install_db` after the installation.

The installation layout is similar to the one of the 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], page 76.

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 already 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 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 2.5.8 [Upgrading-grant-tables], page 139.

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 will be installed into '`/Library/StartupItems/MySQL`'. It 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 '`/etc/hostconfig`'. 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 an already running MySQL server.

After the installation, you can start up MySQL by running the following commands in a terminal window. Please note that you must have administrator privileges to perform this task.

If you have installed the Startup Item:

```
shell> sudo /Library/StartupItems/MySQL/MySQL start
(Enter your password, if necessary)
(Press Control-D or enter "exit" to exit the shell)
```

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 now be able to connect to the MySQL server, for example, by running '`/usr/local/mysql/bin/mysql`'.

If you are installing MySQL for the first time, **please remember to set a password for the MySQL `root` user!**

This is done with the following two commands:

```
/usr/local/mysql/bin/mysqladmin -u root password "newpwd"
/usr/local/mysql/bin/mysqladmin -u root -h `hostname` password "newpwd"
```

Please make sure that the `hostname` command in the second line is enclosed by **backtick** characters (`) so that the shell can replace it with the output of the command (which is the hostname of your system)!

You might want to also add aliases to your shell's resource file to access `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`'.

2.2.4 Installing MySQL on NetWare

Porting MySQL to NetWare was an effort spearheaded by Novell. Novell customers will be pleased to note that NetWare 6.5 will ship 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.

In order to host MySQL, the NetWare server must meet these requirements:

- NetWare version 6.5 or later, (Support pack 1.1 is recommended; you can find this and other updates at http://support.novell.com/filefinder/18197/index.html or NetWare 6.0 with Support Pack 3 installed (you can obtain this at http://support.novell.com/filefinder/13659/index.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.

The latest binary packages for NetWare can be obtained at http://www.mysql.com/downloads/.

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 will install 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') now, as well as 'my.cnf', if you have customized it. You can then delete the old copy of MySQL.
4. You may wish 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 will also be 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
```

The behavior of mysqld_safe on NetWare is described further in Section 5.1.3 [mysqld_safe], page 221.

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.

2.2.5 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], page 67 for a detailed list.

In addition to these generic packages, we also offer binaries in platform-specific package formats for selected platforms. See Section 2.2 [Quick Standard Installation], page 77 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 will help others solve your problem. By not using mysqlbug, you lessen the likelihood of getting a solution to your problem. You will find mysqlbug in the 'bin' directory after you unpack the distribution. See Section 1.7.1.3 [Bug reports], page 34.

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.

A more detailed description follows.

To install a binary distribution, follow these steps, then proceed to Section 2.4 [Post-installation], page 115, for post-installation setup and testing:

1. Add a 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 may wish to call the user and group something else instead of `mysql`.
2. Pick the directory under which you want to unpack the distribution, and move into it. In the following example, we unpack the distribution under '`/usr/local`' (The following instructions, therefore, assume that you have permission to create files and directories in '`/usr/local`'. If that directory is protected, you will need to perform the installation as `root`.)
3. Obtain a distribution file from one of the sites listed in Section 2.1.3 [Getting MySQL], page 73.

   MySQL `tar` file binary distributions have names like '`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-gnu-i586`). For a given release, binary distributions for all platforms are built from the same MySQL source distribution.
4. Change into the intended installation directory:

   ```
   shell> cd /usr/local
   ```
5. Unpack the distribution, which will create 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
   ```
6. Change into the installation directory:

   ```
   shell> cd mysql
   ```

   You will 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 E [Environment variables], page 1211.

'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.

7. 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. If you run the command as mysql, you can omit it.

   Note that for MySQL versions older than Version 3.22.10, mysql_install_db left the server running after creating the grant tables. This is no longer true; you will need to restart the server after performing the remaining steps in this procedure.

8. Change ownership of program binaries to root and ownership of the data directory to the user that you will 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.

9. 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 2.4.2.2 [Automatic start], page 122.

10. 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 2.7 [Perl support], page 166.

11. 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 will 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.

Now proceed to Section 2.4 [Post-installation], page 115.

More information about mysqld_safe is given in Section 5.1.3 [mysqld_safe], page 221.

2.3 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 will work for you. We put a lot of effort into making sure that our binaries are built with the best possible options.

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 tar implementations that come pre-installed with the operating system (for example, Sun tar) is known to have problems with long file names). In that case, 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 Version 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 recent version of gcc, recent enough to understand the -fno-exceptions option, it is **very important** that you use it. 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 will give 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 will help others solve your problem. By not using mysqlbug, you lessen the likelihood of getting a solution to your problem. You will find mysqlbug in the 'scripts' directory after you unpack the distribution. See Section 1.7.1.3 [Bug reports], page 34.

2.3.1 Quick 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
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> rpm --rebuild --clean MySQL-VERSION.src.rpm
```

This will make a binary RPM that you can install.

A more detailed description follows.

To install a source distribution, follow these steps, then proceed to Section 2.4 [Post-installation], page 115, for post-installation initialization and testing:

1. Add a 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 may wish to call the user and group something else instead of `mysql`.
2. Pick the directory under which you want to unpack the distribution, and move into it.
3. Obtain a distribution file from one of the sites listed in Section 2.1.3 [Getting MySQL], page 73. MySQL source distributions are provided as compressed `tar` archives and have names like '`mysql-VERSION.tar.gz`', where `VERSION` is a number like `5.0.0-alpha`.
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 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 2.3.2 [`configure` options], page 101, 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 1.7.1.3 [Bug reports], page 34.

   If the compile fails, see Section 2.3.4 [Compilation problems], page 106, for help with a number of common problems.

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 [Option files], page 212 and Section 16.4 [`InnoDB` configuration], page 759.

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
   ```

   Note that for MySQL versions older than Version 3.22.10, `mysql_install_db` left the server running after creating the grant tables. This is no longer true; you will need to restart the server after performing the remaining steps in this procedure.

10. Change ownership of binaries to `root` and ownership of the data directory to the user that you will 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 2.4.2.2 [Automatic start], page 122.

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 2.7 [Perl support], page 166.

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 `bin/safe_mysqld` for `bin/mysqld_safe` in the command.

If that command fails immediately and prints `mysqld ended`, you can find some information in the file '`mysql-data-directory/'hostname'.err`'. The likely reason is that you already have another `mysqld` server running. See Section 5.9 [Multiple servers], page 351.

Now proceed to Section 2.4 [Post-installation], page 115.

2.3.2 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 E [Environment variables], page 1211. 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` will not compile (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 will 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 get an 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 [Option files], page 212.

* 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
```

  Note that the given file must be an absolute pathname. You can also later change the location of '`mysql.sock`' by using a MySQL option file. See Section A.4.5 [Problems with 'mysql.sock'], page 1018.
* 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 will 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 optimal 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], page 67. There are some configuration settings you can tweak to make an even faster binary, but this is only for advanced users. See Section 7.5.3 [Compile and link options], page 440.

If the build fails and produces errors about your compiler or linker not being able to create the shared library 'libmysqlclient.so.#' ('#' is a version number), you can work around this problem by giving the --disable-shared option to configure. In this case, configure will not build a shared 'libmysqlclient.so.#' library.

- You can configure MySQL not to use DEFAULT column values for non-NULL columns (that is, columns that are not allowed to be NULL). See Section 1.8.6.2 [constraint NOT NULL], page 51.

```
shell> CXXFLAGS=-DDONT_USE_DEFAULT_FIELDS ./configure
```

  The effect of this flag is to cause any INSERT statement to fail unless it provides explicit values for all columns that require a non-NULL value.

- By default, MySQL uses the ISO-8859-1 (Latin1) 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 5.7.1 [Character sets], page 340.

  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 command. See Section 14.5.3.1 [SET], page 702.

  Warning: If you change character sets after having created any tables, you will 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 sets separated by spaces, complex to include all characters 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 D.1 [Debugging server], page 1199.

- 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 will create a `libmysqlclient_r` library with which you should link your threaded applications. See Section 20.2.14 [Threaded clients], page 950.
- Options that pertain to particular systems can be found in the system-specific section of this manual. See Section 2.6 [Operating System Specific Notes], page 141.

2.3.3 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 will do).

To obtain our most recent development source tree, use these instructions:

1. Download `BitKeeper` from `http://www.bitmover.com/cgi-bin/download.cgi`. You will 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 alpha 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 will be 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/"
   ```

   Now, simply 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 will need GNU `make`, `autoconf` 2.53 (or newer), `automake` 1.5, `libtool` 1.5, and `m4` to run the next set of commands. Even though many operating systems already come with their own implementation of `make`, chances are high that the compilation will fail 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 already 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/make/`

   If you are trying to configure MySQL 4.1 or later, you will 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 22.1.2 [MySQL test suite], page 984.
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 1.7.1.1 [Mailing-list], page 32. 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` will produce 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 (`bk ci` or `bk citool`) will trigger 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 will 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 which can be cloned with:

```
shell> bk clone bk://mysql.bkbits.net/mysqldoc mysqldoc
```

There are also public BitKeeper trees for MySQL Control Center and Connector/ODBC. 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 Connector/ODBC, use this command:

```
shell> bk clone http://mysql.bkbits.net/myodbc3 myodbc3
```

2.3.4 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 2.3.5 [MIT-pthreads], page 110 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 already has 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 re-running `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 when compiling '`sql_yacc.cc`', such as the ones shown here, you have probably run out of memory or swap space:

  ```
  Internal compiler error: program cc1plus got fatal signal 11
  ```

  Or:

  ```
  Out of virtual memory
  ```

  Or:

  ```
  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 you to not use libg++ or libstdc++ with MySQL because this will only increase 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 14.2.5 [CREATE TABLE], page 670 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], page 67, 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 will 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 already 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 D.2 [Debugging client], page 1204.
- 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.

2.3.5 MIT-pthreads Notes

This section describes some of the issues involved in using MIT-pthreads.

Note that on Linux you should **not** use MIT-pthreads but use the installed LinuxThreads implementation instead. See Section 2.6.1 [Linux], page 141.

If your system does not provide native thread support, you will 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 [Which OS], page 60.

Note that, beginning with MySQL 4.0.2, MIT-pthreads are 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 will create a new subdirectory `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 will not know whether MIT-pthreads is being used and will use Unix socket connections by default. Because Unix socket files do not work under MIT-pthreads on some platforms, this means you will 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 (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.)

2.3.6 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 2.2.1 [Windows installation], page 77.

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 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://www.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 2.3.6.1 [Windows VC++ Build], page 112. To build from the BitKeeper tree, proceed to Section 2.3.6.2 [Windows BitKeeper Build], page 114.

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 1.7.1.1 [Mailing-list], page 32.

2.3.6.1 Building MySQL Using VC++

Note: MySQL 4.1 and above VC++ workspace files 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, '`workdir`').
2. Unpack the source distribution in the aforementioned directory using `WinZip` or other Windows tools 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 will expect 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 will 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 ('`C:\my.cnf`' or the '`my.ini`' file in your Windows directory). 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 2.2.1 [Windows installation], page 77. You'll need to 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 install 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 \mysql
   mkdir \mysql\bin
   mkdir \mysql\data
   mkdir \mysql\share
   mkdir \mysql\scripts
   ```

   If you want to compile other clients and link them to MySQL, you should also create several additional directories:

   ```
   mkdir \mysql\include
   mkdir \mysql\lib
   mkdir \mysql\lib\debug
   mkdir \mysql\lib\opt
   ```

   If you want to benchmark MySQL, create this directory:

   ```
   mkdir \mysql\sql-bench
   ```

   Benchmarking requires Perl support.
2. From the 'workdir' directory, copy into the C:\mysql directory the following directories:

   ```
   copy client_release\*.exe C:\mysql\bin
   copy client_debug\mysqld.exe C:\mysql\bin\mysqld-debug.exe
   xcopy scripts\*.* C:\mysql\scripts /E
   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:

   ```
   copy lib_debug\mysqlclient.lib C:\mysql\lib\debug
   copy lib_debug\libmysql.* C:\mysql\lib\debug
   copy lib_debug\zlib.* C:\mysql\lib\debug
   copy lib_release\mysqlclient.lib C:\mysql\lib\opt
   copy lib_release\libmysql.* C:\mysql\lib\opt
   copy lib_release\zlib.* C:\mysql\lib\opt
   copy include\*.h C:\mysql\include
   copy libmysql\libmysql.def C:\mysql\include
   ```

   If you want to benchmark MySQL, you should also do this:

```
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 2.2.1 [Windows installation], page 77.

2.3.6.2 Creating a Windows Source Package from the Latest Development 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. (The procedure is known to work well on Linux, for example.)

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 2.3.3 [Installing source tree], page 104.
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:
   ```
   --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
   --help    Show this help message
   ```
   By default, make_win_src_distribution creates a zipped 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 2.3.6.1 [Windows VC++ Build], page 112.

2.3.7 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.

2.4 Post-Installation Setup and Testing

There are some issues you should address after installing MySQL. 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.

The following sections describe post-installation procedures for Windows systems and for Unix systems.

2.4.1 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 already set up with a set of preinitialized accounts in the `mysql` database under the data directory. However, you should assign passwords to the accounts.

The default privileges on Windows give all local users full privileges to all databases without specifying a password. To make MySQL more secure, you should set a password for at least the `root` accounts. You should also remove the row in the `mysql.user` table that has `Host='localhost'` and `User=''`. This account allows anonymous access but has full privileges, so removing it improves security.

The following example shows how to remove the anonymous-user account that has all privileges, and then assigns a password to the `root` accounts:

```
C:\> C:\mysql\bin\mysql -u root mysql
mysql> DELETE FROM user WHERE Host='localhost' AND User='';
mysql> FLUSH PRIVILEGES;
mysql> QUIT
C:\> C:\mysql\bin\mysqladmin -u root password "newpwd"
C:\> C:\mysql\bin\mysqladmin -u root -h host_name password "newpwd"
```

Replace "newpwd" with the actual `root` password that you want to use. The double quotes are not always necessary, but they are helpful if the password contains spaces or other characters that are special to your command interpreter. In the second `mysqladmin` command, replace `host_name` with the name of the server host.

After you've assigned a password to the `root` accounts, you'll need to specify it whenever you run a MySQL client program to connect to the server as `root`. For example, if you want to use `mysqladmin` to shut down the server, you can do so using this command:

```
C:\> mysqladmin --user=root -p shutdown
```

If you are using a server from a *very* old version of MySQL, the `mysqladmin` command to set the password will fail with an error: `parse error near 'SET password'`. The solution to this problem is to upgrade the server to a newer version of MySQL.

With the current MySQL versions, you can easily add new users and change privileges with `GRANT` and `REVOKE` commands. See Section 14.5.1.2 [GRANT], page 689.

2.4.2 Unix Post-Installation Procedures

After you have installed 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.

The grant tables on Unix 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 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 2.4.2.2 [Automatic start], page 122.

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.

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 5.5.2 [Default privileges], page 301.

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, 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 do it for you. To initialize the grant tables, run the `mysql_install_db` script:
   - If you installed MySQL using a binary distribution, use this command:

     ```
     shell> scripts/mysql_install_db --user=mysql
     ```
   - If you installed MySQL using a source distribution, use this command:

     ```
     shell> bin/mysql_install_db --user=mysql
     ```

The mysql_install_db script creates 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 5.5.2 [Default privileges], page 301. 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.

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.

If you have problems with mysql_install_db, see Section 2.4.2.1 [mysql_install_db], page 120.

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 already 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 Version 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 [Changing MySQL user], page 1011.

If you neglected to create the grant tables before proceeding to this step, the following message will appear 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 2.4.2.3 [Starting server], page 124.

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 2.4.2.3 [Starting server], page 124.

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 2.7.1 [Perl installation], page 167.

   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 Version 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. None of the initial MySQL accounts have a password, so you should assign passwords using the instructions in Section 5.5.2 [Default privileges], page 301.

2.4.2.1 Problems Running mysql_install_db

The purpose of the mysql_install_db script is to generate new MySQL privilege tables. It will not overwrite existing MySQL privilege tables, and it will 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 1.7.1.3 [Bug reports], page 34.

There is already a mysqld process running

This indicates that the server is already running, in which case the grant tables probably have already 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 already have an existing MySQL installation, but want to put a new installation in a different location. For example, you might have a production installation already, 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 already in use by the first server. In this case, you will 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 to run on the same machine, see Section 5.9 [Multiple servers], page 351.

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 will occur 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> scripts/mysql_install_db --user=mysql
shell> bin/mysqld_safe --user=mysql &
```

See Section A.4.5 [Problems with 'mysql.sock'], page 1018. See Appendix E [Environment variables], page 1211.

2.4.2.2 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, and XP). 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 2.2.1.7 [NT start], page 83.
- 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 [mysqld_safe], page 221.
- 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 [mysql.server], page 224.
- 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 2.2.3 [Mac OS X installation], page 91 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 non-standard 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.

Note that if you use the Linux server RPM package (MySQL-server-VERSION.rpm), the mysql.server script will already have been installed in the '/etc/init.d' directory with the name 'mysql'. You need not install it manually. See Section 2.2.2 [Linux-RPM], page 89 for more information on the Linux RPM packages.

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, or else first create '`/etc/init.d`' as a symbolic link that points to '`/etc/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]` and `[mysqld-major-version]`
`mysql.server`	`[mysql.server]` and `[mysqld]`
`mysqld_safe`	`[mysqld]`, `[server]`, and `[mysqld_safe]`

`[mysqld-major-version]` means that groups with names like `[mysqld-4.0]`, `[mysqld-4.1]`, and `[mysqld-5.0]` will be 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 will 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 [Option files], page 212.

2.4.2.3 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.
- Check the error log to see why the server doesn't start.
- Verify that the network interfaces the server wants to use are available.
- Check the ownership and permissions of the data directory and its contents. They must be set such that the server can access and modify them.

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 will fail to start. From MySQL 4.0 on, `InnoDB` uses default values for its configuration options if you specify none. See Section 16.4 [`InnoDB` configuration], page 759.
- If you are using `BDB` (Berkeley DB) tables, you should familiarize yourself with the different `BDB`-specific startup options. See Section 15.4.3 [BDB start], page 753.

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 will 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 will look 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 will show 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 'your-host-name' variables
```

Whichever method you use to start the server, if it 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 error 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 already 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 5.9 [Multiple servers], page 351.)

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.

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.

If mysqld_safe starts the server 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. See Section D.1.2 [Making trace files], page 1200.

2.5 Upgrading/Downgrading 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 change log for the release series to which you are upgrading to see what new features you can use. For example, before upgrading from MySQL 4.1 to 5.0, read the 5.0 news items.
- Before you do an upgrade, you should back up your old databases.

- If you are running MySQL Server on Windows, see Section 2.5.7 [Windows upgrading], page 138.
- If you are using replication, see Section 6.6 [Replication Upgrade], page 374 for information on upgrading your replication setup.
- After upgrading, be sure that your grant tables are up to date. (These are the tables in the `mysql` database.) Occasionally new columns or tables are added to support new features. To take advantage of these features, your tables must be current. The upgrade procedure is described in Section 2.5.8 [Upgrading-grant-tables], page 139.

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 4.0. If you change the character set when running MySQL, you must run `myisamchk -r -q --set-character-set=charset` on all tables. Otherwise, your indexes may not be ordered correctly, because changing the character set may also change the sort order.

If you upgrade or 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 upgrading. After upgrading, reload the dump file using `mysql` to re-create 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, please 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 old installation. You can check this with: `program-name --print-defaults`. If this outputs 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 Python `MySQLdb` module.

2.5.1 Upgrading from Version 4.1 to 5.0

In general, you should do the following when upgrading to MySQL 5.0 from an earlier version:

- Read the 5.0 news items to see what significant new features you can use in 5.0. See Section C.1 [News-5.0.x], page 1040.
- If you are running MySQL Server on Windows, see Section 2.5.7 [Windows upgrading], page 138.

- If you are using replication, see Section 6.6 [Replication Upgrade], page 374 for information on upgrading your replication setup.
- 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 2.5.8 [Upgrading-grant-tables], page 139.

2.5.2 Upgrading from Version 4.0 to 4.1

In general, you should do the following when upgrading to MySQL 4.1 from an earlier version:

- Check the items in the change list found later in this section to see whether any of them might affect your applications.
- Read the 4.1 news items to see what significant new features you can use in 4.1. See Section C.2 [News-4.1.x], page 1043.
- If you are running MySQL Server on Windows, see Section 2.5.7 [Windows upgrading], page 138.

  Important note: Early alpha Windows distributions for MySQL 4.1 do not contain an installer program. See Section 2.2.1.2 [Windows binary installation], page 78 for instructions on how to install such a distribution.
- If you are using replication, see Section 6.6 [Replication Upgrade], page 374 for information on upgrading your replication setup.
- After upgrading, update the grant tables to have the new longer `Password` column that is needed for secure handling of passwords. The procedure uses `mysql_fix_privilege_tables` and is described in Section 2.5.8 [Upgrading-grant-tables], page 139. Implications of the password-handling change for applications are given later in this section. If you don't do this, MySQL will not us the new more secure protocol to authenticate.
- 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 convert your BDB tables to text format and delete all `log.??????` files before you start MySQL 4.0 and read back the data.
- String comparison now works according to SQL standard: Instead of stripping end space before comparison, we now extend the shorter string with space. The problem with this is that now `'a' > 'a\t'`, which it wasn't before. If you have any tables where you have a `CHAR` or `VARCHAR` column in which the last character in the column may be less than `ASCII(32)`, you should use `REPAIR TABLE` or `myisamchk` to ensure that the table is correct.
- If you have table columns that store character data represented in a character set that the 4.1 server now supports directly, you can convert the columns to the proper character set using the instructions in Section 11.10.2 [Charset-conversion], page 526.
- Some API calls such as `mysql_real_query()` now 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");
    }
```

- If you are using an old DBD-mysql module (Msql-MySQL-modules) you have to upgrade to use the newer DBD-mysql modules. Anything above DBD-mysql-2 should be fine.

  If you don't upgrade, some commands (such as DBI->do()) will not notice error conditions correctly.
- Option --defaults-file=option-file-name will now give an error if the option file doesn't exists.

Several visible behaviors have changed between MySQL 4.0 and MySQL 4.1 to fix some critical bugs and make MySQL more compatible with the ANSI SQL standard. 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 [Server options], page 228.

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 will 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 will help 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 list describes 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 now have a character set. See Chapter 11 [Charset], page 507. 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 applications that use only one character set.
- 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 [mysqldump], page 477.

- **Important note:** If you upgrade to InnoDB-4.1.1 or higher, it is difficult to downgrade back to 4.0 or 4.1.0! That is because earlier versions of `InnoDB` are not aware of multiple tablespaces.
- If you are running multiple servers on the same Windows machine, you should use a different `--shared_memory_base_name` option for each server.
- The interface to aggregated UDF functions has changed a bit. You must now declare a `xxx_clear()` function for each aggregate function `XXX()`.

SQL Changes:

- When using `DELETE` with many tables, you have to use the alias of the tables from which you want to delete, not the actual table name.

  ```
  Change:

  DELETE test FROM test AS t1, test2 WHERE ...

  to

  DELETE t1 FROM test AS t1, test2 WHERE ...
  ```
- `TIMESTAMP` is now returned as a string in `'YYYY-MM-DD HH:MM:SS'` format. (The `--new` option can be used from 4.0.12 on to make a 4.0 server behave as 4.1 in this respect.) 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.
- Binary values such as `0xFFDF` now 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 will be 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 now 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 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.

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 will 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 will change (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 20.2.4 [C API Prepared statements], page 911.
- 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.
- Upgrade to 4.1 and run the `mysql_fix_privilege_tables` script to widen the `Password` column in the `user` table. If you know that all clients also have been upgraded to 4.1, don't run the server with the `--old-passwords` option. Instead, change the passwords on all existing accounts so that they have the new 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 5.4.9 [Password hashing], page 295.

2.5.3 Upgrading from Version 3.23 to 4.0

In general, you should do the following when upgrading to MySQL 4.0 from an earlier version:

- Check the items in the change list found later in this section to see whether any of them might affect your applications.

- Read the 4.0 news items to see what significant new features you can use in 4.0. See Section C.3 [News-4.0.x], page 1059.
- If you are running MySQL Server on Windows, see Section 2.5.7 [Windows upgrading], page 138.
- If you are using replication, see Section 6.6 [Replication Upgrade], page 374 for information on upgrading your replication setup.
- 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 2.5.8 [Upgrading-grant-tables], page 139.
- 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 will work even if you don't perform the preceding actions, but you will not be able to 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 Version 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 is a more complete list that tells what you must watch out for when upgrading to version 4.0:

Server Changes:

- MySQL 4.0 has a lot of new privileges in the `mysql.user` table. See Section 14.5.1.2 [`GRANT`], page 689.

To get these new privileges to work, you must update the grant tables. The procedure is described in Section 2.5.8 [Upgrading-grant-tables], page 139. 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 5.3.3 [Privileges options], page 273.

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 will need `REPLICATION SLAVE` (instead of `FILE`) for new slaves.

- `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 now 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 now given in bytes (they were given in megabytes before 4.0.3).
- `mysqld` now 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 now turned off by default. Your can turn this on by doing `--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` will 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` now returns an empty set if binary logging is not enabled.
- `SHOW SLAVE STATUS` now 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 now reserved words.
- `DOUBLE` and `FLOAT` columns now honor the `UNSIGNED` flag on storage (before, `UNSIGNED` was ignored for these columns).
- The result of all bitwise operators (`|`, `&`, `<<`, `>>`, and `~`) is now unsigned. This may cause problems if you are using them in a context where you want a signed result. See Section 13.7 [Cast Functions], page 605.

  Note: When you use subtraction between integer values where one is of type `UNSIGNED`, the result will be 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 13.7 [Cast Functions], page 605.
- You should use integers to store values in `BIGINT` columns (instead of using strings, as you did in MySQL 3.23). Using strings will still work, but using integers is more efficient.
- In 3.23, `INSERT INTO ... SELECT` always had `IGNORE` enabled. In 4.0.1, MySQL will stop (and possibly roll 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 will get an error if you have an active `LOCK TABLES` or transaction 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 will return incorrect results. See Section 13.6.4 [Fulltext Fine-tuning], page 603.
- `LOCATE()` and `INSTR()` are case-sensitive if one of the arguments is a binary string. Otherwise they are case-insensitive.

- `STRCMP()` now 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.
- `HEX(string)` now returns the characters in `string` 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 now 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 20.2.14 [Threaded clients], page 950.

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.

2.5.4 Upgrading from Version 3.22 to 3.23

MySQL Version 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 Version 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`.

Version 3.22 and 3.21 clients will work without any problems with a Version 3.23 server.

When upgrading to MySQL 3.23 from an earlier version, note the following changes:

Client Program Changes:

- The MySQL client `mysql` is now by default started with the option `--no-named-commands (-g)`. 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 Version 3.22 and Version 3.23, you should not use the `--opt` or `--all` option to `mysqldump`.

Table Changes:

- 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.

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 3.22 because `configure` didn't detect the availability of the `readlink()` system call.)
- `OPTIMIZE TABLE` now works only for `MyISAM` tables. For other table types, you can use `ALTER TABLE` to optimize the table. During `OPTIMIZE TABLE`, the table is now locked to prevent it from being used by other threads.
- Date functions that work on parts of dates (such as `MONTH()`) will now return 0 for `0000-00-00` dates. (In MySQL 3.22, these functions returned `NULL`.)
- The default return type of `IF()` now depends on both arguments and not only the first argument.
- `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` will complain about 0 values because they may change if you dump and restore the table. `AUTO_INCREMENT` for `MyISAM` tables is now handled at a lower level and is much faster than before. In addition, for `MyISAM` tables, old numbers are no longer reused, even if you delete rows from the table.
- `CASE`, `DELAYED`, `ELSE`, `END`, `FULLTEXT`, `INNER`, `RIGHT`, `THEN`, and `WHEN` are now reserved words.
- `FLOAT(X)` is now 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 now be of one of the following formats: `[[[DAYS] [H]H:]MM:]SS[.fraction]` or `[[[[[H]H]H]H]MM]SS[.fraction]`.
- `LIKE` now 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 now 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. (MySQL Version 3.22 and later already allowed this syntax.)
- In MySQL Version 3.22, the output of `SELECT DISTINCT ...` was almost always sorted. In Version 3.23, you must use `GROUP BY` or `ORDER BY` to obtain sorted output.
- `SUM()` now returns `NULL` instead of 0 if there are no matching rows. This is required by standard SQL.

- An AND or OR with NULL values will now return NULL instead of 0. This mostly affects queries that use NOT on an AND/OR expression as NOT NULL = NULL.
- LPAD() and RPAD() now shorten the result string if it's longer than the length argument.

C API Changes:

- mysql_fetch_fields_direct() is now a function (it used to be a macro) and it returns a pointer to a MYSQL_FIELD instead of a MYSQL_FIELD.
- mysql_num_fields() can no longer be used on a MYSQL* object (it's now a function that takes a MYSQL_RES* value as an argument). With a MYSQL* object, you should now use mysql_field_count() instead.

2.5.5 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 will 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 will add the new privileges that you need to use the GRANT command. If you forget this, you will 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 2.5.8 [Upgrading-grant-tables], page 139.
- The C API interface to mysql_real_connect() has changed. If you have an old client program that calls this function, you must place 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.

2.5.6 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() will 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 will need to make the following changes:

- All client code must be recompiled. If you are using ODBC, you must get the new 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 Version 3.20.28 and above can handle the new `user` table format without affecting clients. If you have a MySQL version earlier than Version 3.20.28, passwords will 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 will be unable to connect and will 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 now be specified before any `ORDER BY` clause.
- The parameters to `LOCATE()` have been swapped.
- There are some new reserved words. The most notable are `DATE`, `TIME`, and `TIMESTAMP`.

2.5.7 Upgrading MySQL Under Windows

When upgrading MySQL under Windows, please follow these steps:

1. Download the latest Windows distribution of MySQL.
2. Choose a time of day with low usage, where a maintenance break is acceptable.
3. Alert the users who still are active about the maintenance break.
4. Stop the running MySQL Server (for example, with `NET STOP MySQL` or with the `Services` utility if you are running MySQL as a service, or with `mysqladmin shutdown` otherwise).
5. Exit the `WinMySQLAdmin` program if it is running.
6. Run the installation script of the Windows distribution by clicking the Install button in WinZip and following the installation steps of the script.

   Important note: Early alpha Windows distributions for MySQL 4.1 do not contain an installer program. See Section 2.2.1.2 [Windows binary installation], page 78 for instructions on how to install such a distribution.
7. You may either overwrite your old MySQL installation (usually located at '`C:\mysql`'), or install it into a different directory, such as `C:\mysql4`. Overwriting the old 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. Update the grant tables. The procedure is described in Section 2.5.8 [Upgrading-grant-tables], page 139.

Possible error situations:

```
A system error has occurred.
System error 1067 has occurred.
The process terminated unexpectedly.
```

This error means that your 'my.cnf' file (by default 'C:\my.cnf') contains an option that cannot be recognized by MySQL. You can verify that this is the case by trying to restart MySQL with the 'my.cnf' file renamed, for example, to 'my_cnf.old' to prevent the server from using it. Once you have verified it, you need to identify which option is the culprit. Create a new 'my.cnf' file and move parts of the old file to it (restarting the server after you move each part) until you determine which option causes server startup to fail.

2.5.8 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 may see some Duplicate column name warnings as it runs; they can be ignored.

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:\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 may see some Duplicate column name warnings as mysql processes the statements in the mysql_fix_privilege_tables.sql script; they can be ignored.

After running the script, stop the server and restart it.

2.5.9 Copying MySQL Databases to Another Machine

If you are using MySQL Version 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 15.1 [MyISAM Tables], page 738.

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 will create 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's where the grant tables (`user`, `db`, and `host`) are stored. You may 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.

2.6 Operating System-Specific Notes

2.6.1 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 likely will see none of them.

2.6.1.1 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 will give you a significant speed boost. Your system also will be more stable.

When using LinuxThreads, you will see a minimum of three `mysqld` processes running. These are in fact threads. There will be one thread for the LinuxThreads manager, one thread to handle connections, and one thread to handle alarms and signals.

2.6.1.2 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 will 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 probably will 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 will 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` with `su` 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 (an RPM or the `tar.gz` distribution) 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 will 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.

2.6.1.3 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 our binary 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://www.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 will 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 will not be 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 version 3.23.36 will use 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 will occur 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 will need to apply it in the `linuxthreads` directory with `patch -p0 </tmp/linuxthreads-2.2.2.patch`. We hope it will be 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 will be 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 are doing it 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 will resolve 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 will 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
```

2.6.1.4 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 2.4.2.2 [Automatic start], page 122.

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 will be 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 will be plenty of room for the stack of each individual thread, 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() will successfully unmap an already mapped region if you ask it to map out an address already 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 2.6.1.3 [Source notes-Linux], page 142. 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 now 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 now 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 will give 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 `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 gave 60% throughput increase with 400 clients.

We are currently also trying to collect more information on how well MySQL performs on 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.4.1 [Crashing], page 1013.

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 [mysqld_safe], page 221.

2.6.1.5 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 now 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 will 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. Links to the mail archives are available online at http://lists.mysql.com/.

2.6.1.6 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.

2.6.1.7 Linux Alpha Notes

MySQL Version 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 will 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 will dump core at startup time. In other words, **don't** use `--with-mysqld-ldflags=-all-static` with `gcc`.

2.6.1.8 Linux PowerPC Notes

MySQL should work on MkLinux with the newest `glibc` package (tested with `glibc` 2.0.7).

2.6.1.9 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).

2.6.1.10 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 [Link errors], page 1010.

2.6.2 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.

2.6.2.1 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 2.2.3 [Mac OS X installation], page 91.

2.6.2.2 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 2.2.3 [Mac OS X installation], page 91.

2.6.3 Solaris Notes

On Solaris, you may run into trouble even before you get the MySQL distribution unpacked! Solaris tar can't handle long file names, 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://www.mysql.com/downloads/os-solaris.html.

Sun native threads only work on Solaris 2.5 and higher. For Version 2.4 and earlier, MySQL will automatically use MIT-pthreads. See Section 2.3.5 [MIT-pthreads], page 110.

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. Note that 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 -mcpu flags.

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 may have to edit the configure script to 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 multi-thread option (-mt):

```
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 will be 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 will 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 will 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 will 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.

2.6.3.1 Solaris 2.7/2.8 Notes

You can normally 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 Version 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 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 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.

2.6.3.2 Solaris x86 Notes

On Solaris 8 on x86, `mysqld` will dump 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 will avoid 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 D.1.3 [Using `gdb` on `mysqld`], page 1201.

2.6.4 BSD Notes

This section provides information about using MySQL on BSD variants.

2.6.4.1 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 will suddenly cause 100% CPU load and will be unresponsive. If you encounter this problem, try to start up 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 `wait_timeout` is not working (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:

```
shell> 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
shell> gmake
shell> gmake install
shell> cd /usr/local/mysql
shell> bin/mysql_install_db
shell> bin/mysqld_safe &
```

If you notice that `configure` will use MIT-pthreads, you should read the MIT-pthreads notes. See Section 2.3.5 [MIT-pthreads], page 110.

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.2.18 [Not enough file handles], page 1009. 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 [mysqld_safe], page 221.

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 will probably help. See Appendix E [Environment variables], page 1211.

2.6.4.2 NetBSD Notes

To compile on NetBSD, you need GNU `make`. Otherwise, the build process will fail when `make` tries to run `lint` on C++ files.

2.6.4.3 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
```

2.6.4.4 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 haven't allocated enough file descriptors for MySQL.

In this case, try starting `mysqld_safe` as `root` with the following options:

```
shell> mysqld_safe --user=mysql --open-files-limit=2048 &
```

2.6.4.5 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 will probably help. See Appendix E [Environment variables], page 1211.

2.6.4.6 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:

```
shell> 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:

```
shell> 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 will run 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`.

2.6.4.7 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 asking for this!

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).

2.6.5 Other Unix Notes

2.6.5.1 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
```

2.6.5.2 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 will solve 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 will get the 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 (details below).

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`'. Change this line:

```
extern int pthread_atfork(void (*prepare)(), void (*parent)(),
                                          void (*child)());
```

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 a 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.

2.6.5.3 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 \
                --sysconfdir=/etc/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 will produce 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://publib.boulder.ibm.com/doc_link/en_US/a_doc_lib/cmds/aixcmds5/uname.htm. This will give 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 with:

```
shell> 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 will die when it issues its next command.

On some versions of AIX, linking with libbind.a makes getservbyname core dump. 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 initgoups.

```
#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 about using a lot of memory at: http://publib16.boulder.ibm.com/pseries/en_US/aixprggd/genprogc/lrg_prg_support.htm.

2.6.5.4 SunOS 4 Notes

On SunOS 4, MIT-pthreads is needed to compile MySQL. This in turn means you will 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:

```
shell> ./configure --disable-shared --with-mysqld-ldflags=-all-static
```

When compiling readline, you may get warnings about duplicate defines. These may be ignored.

When compiling mysqld, there will be some implicit declaration of function warnings. These may be ignored.

2.6.5.5 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:

```
shell> 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 [mysqld_safe], page 221.

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:

```
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
```

2.6.5.6 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` will use the following thread libraries automatically: `--with-named-thread-libs="-lpthread -lmach -lexc -lc"`.

When using `gcc`, you can also try running `configure` like this:

```
shell> 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:

```
shell> 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 will die when it issues its next command.

With `gcc` 2.95.2, you will probably run into 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.

2.6.5.7 SGI Irix Notes

If you are using Irix Version 6.5.3 or newer, mysqld will be able to create threads only if you run it as a user with CAP_SCHED_MGT privileges (like root) or give the mysqld server this privilege with the following shell command:

```
shell> 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:

```
shell> extra/replace bool curses_bool < /usr/include/curses.h \
           > include/curses.h
shell> 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
```

2.6.5.8 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."

For the moment, the recommended compiler on OpenServer is `gcc` 2.95.2. With this, you should be able to compile MySQL with just:

```
CC=gcc CXX=gcc ./configure ... (options)
```

1. For OpenServer 5.0.x, you need to use gcc-2.95.2p1 or newer from the Skunkware. `http://www.sco.com/skunkware/` and choose browser OpenServer packages or by ftp to ftp2.caldera.com in the '`pub/skunkware/osr5/devtools/gcc`' directory.
2. You need the port of GCC 2.5.x for this product and the Development system. They are required on this version of SCO Unix. You cannot just use the GCC Dev system.
3. 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 `http://www.mysql.com/Downloads/SCO/FSU-threads-3.5c.tar`
4. 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 will 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.
5. To build FSU Pthreads on your system, do the following:
   1. Run `./configure` in the '`threads/src`' directory and select the SCO OpenServer option. This command copies '`Makefile.SCO5`' to '`Makefile`'.
   2. Run `make`.
   3. To install in the default '`/usr/include`' directory, login as root, then `cd` to the '`thread/src`' directory, and run `make install`.
6. Remember to use GNU `make` when making MySQL.
7. If you don't start `mysqld_safe` as `root`, you probably will get only the default 110 open files per process. `mysqld` will write a note about this in the log file.
8. With SCO 3.2V5.0.5, you should use FSU Pthreads version 3.5c or newer. You should also use `gcc` 2.95.2 or newer!

   The following `configure` command should work:

   ```
   shell> ./configure --prefix=/usr/local/mysql --disable-shared
   ```
9. With SCO 3.2V4.2, you should use FSU Pthreads version 3.5c or newer. The following `configure` command should work:

   ```
   shell> 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 include files at `http://www.mysql.com/Downloads/SCO/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 libraries' 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 `http://www.mysql.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/opense` 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.

2.6.5.9 SCO UnixWare Version 7.1.x Notes

On UnixWare 7.1.0, 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 CXX=CC ./configure --prefix=/usr/local/mysql
```

If you want to use `gcc`, you must use `gcc` 2.95.2 or newer.

```
CC=gcc CXX=g++ ./configure --prefix=/usr/local/mysql
```

1. SCO provides operating system patches at `ftp://ftp.sco.com/pub/unixware7` for UnixWare 7.1.1 and 7.1.3 and at `ftp://ftp.sco.com/pub/openunix8` for OpenUNIX 8.0.0.
2. SCO provides information about security fixes at `ftp://ftp.sco.com/pub/security/OpenUNIX` for OpenUNIX and at `ftp://ftp.sco.com/pub/security/UnixWare` for UnixWare.

2.6.6 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 will probably run into 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 now called 'install.cmd' and is a REXX script, which will set up the default MySQL security settings and create 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 will put 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';
```

2.6.7 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.

2.7 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 **will 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.

Note that if you are using MySQL 4.1 client library, you must use 2.9003 or newer.

As of Version 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 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 [MySQL Benchmarks], page 399.

2.7.1 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 will 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 already 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 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 will fail.

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`.

2.7.2 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 already 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";
```

2.7.3 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
```

This is because the Perl dynaloader will 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 (like 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:

```
shell> LD_LIBRARY_PATH=/lib:/usr/lib:/usr/local/lib:/usr/progressive/lib
Or:
shell> LD_LIBRARY_PATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:\
/usr/progressive/lib:/usr/skunk/lib
shell> LIBPATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:\
/usr/progressive/lib:/usr/skunk/lib
shell> 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 will indicate the exact make command you will 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.

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 will 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 already-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 running 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 will 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 will 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 realising what you need to do. Enter a semicolon to complete the statement, and `mysql` will execute 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 inadvertantly 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 will happen. 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 will 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

Now that 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 will be 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 14.5.1.2 [GRANT], page 689.

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` will tell 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 will need and what columns will 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 for now: 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 will 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.

Now that 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 '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 '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 5.3.4 [LOAD DATA LOCAL], page 274 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 now as 1989, not 1979.

String comparisons normally are case-insensitive, so you can specify the name as `'bowser'`, `'BOWSER'`, etc. The query result will be 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, though `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 will be 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], page 187.

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 will match 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 already 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:

```
dos> 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 MySQL Version 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 MySQL Version 4.1 or greater), 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 10.3 [Variables], page 498.

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 16 [InnoDB], page 758. See also Section 1.8.5.5 [ANSI diff Foreign Keys], page 48.

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 will work 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');

INSERT INTO shirt VALUES
(NULL, 'polo', 'blue', LAST_INSERT_ID()),
(NULL, 'dress', 'white', LAST_INSERT_ID()),
(NULL, 't-shirt', 'blue', LAST_INSERT_ID());


INSERT INTO person VALUES (NULL, 'Lilliana Angelovska');

INSERT INTO shirt VALUES
(NULL, 'dress', 'orange', LAST_INSERT_ID()),
(NULL, 'polo', 'red', LAST_INSERT_ID()),
(NULL, 'dress', 'blue', LAST_INSERT_ID()),
(NULL, 't-shirt', 'white', LAST_INSERT_ID());


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

MySQL doesn't yet optimize when you search on two different keys combined with OR (searching on one key with different OR parts is optimized quite well):

```
SELECT field1_index, field2_index FROM test_table
WHERE field1_index = '1' OR  field2_index = '1'
```

The reason is that we haven't yet had time to come up with an efficient way to handle this in the general case. (The AND handling is, in comparison, now completely general and works very well.)

In MySQL 4.0 and up, you can solve this problem efficiently by using a UNION that combines the output of two separate SELECT statements. See Section 14.1.7.2 [UNION], page 650. 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()` will actually return 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 will be 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.)

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 sceening 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 sceening 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 into 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.

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

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

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

More information on running the server can be found in Section 14.5 [Database Administration], page 689. Client and utility programs are described in more detail in Chapter 8 [Client-Side Scripts], page 448.

Most MySQL distribution formats 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], page 59 for information on 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.

```
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 [Program Options], page 210.

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.

Note that 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, though in practice option files are used more commonly for this purpose. (Section 5.9.2 [Multiple Unix servers], page 356 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 by examining environment variables first, then option files, 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 will prompt 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 [Program variables], page 216 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 will not exit with an error if it does not recognize the option, but instead will issue 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 will 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: `mysql`, `mysqladmin`, `mysqld`, `mysqld_safe`, `mysql.server`, `mysqldump`, `mysqlimport`, `mysqlshow`, `mysqlcheck`, `mysqlhotcopy`, `myisamchk`, and `myisampack`.

On Windows, MySQL programs read startup options from the following files:

Filename	**Purpose**

`windows-dir\my.ini`	Global options
`C:\my.cnf`	Global options

`windows-dir` represents the location of your Windows directory. This is commonly '`C:\Windows`' or '`C:\WinNT`'. Check the value of the `WINDIR` environment variable to see where this directory is located on your system.

On Unix, MySQL programs read startup options from the following files:

Filename	**Purpose**
`/etc/my.cnf`	Global options
`DATADIR/my.cnf`	Server-specific options
`defaults-extra-file`	The file specified with `--defaults-extra-file=path`, if any
`~/.my.cnf`	User-specific options

`DATADIR` represents the location of the MySQL data directory. 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 listed 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 [Program variables], page 216 for more information on setting program variables.

Note that for options and values, all leading and trailing blanks are automatically deleted. 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.

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=1M

[mysqldump]
quick
```

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
```

If you have a source distribution, you will 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). 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.

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 will get 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.

Note that 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 the 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. The syntax for Bourne shell and variants (sh, bash, zsh, etc.) is:

```
MYSQL_TCP_PORT=3306
```

For csh and tcsh, use this syntax:

```
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 E [Environment variables], page 1211 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 64MB, use either of the following commands:

```
shell> mysql --max_allowed_packet=6710740
shell> mysql --max_allowed_packet=64M
```

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=6710740
```

Or:

```
[mysql]
max_allowed_packet=64M
```

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=6710740
shell> mysql --set-variable=max_allowed_packet=64M
```

In an option file, omit the leading dashes:

```
[mysql]
set-variable = max_allowed_packet=6710740
```

Or:

```
[mysql]
set-variable = max_allowed_packet=64M
```

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 [Server system variables], page 240 and Section 14.5.3.1 [`SET OPTION`], page 702.

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 MySQL], page 228.

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 [Program Options], page 210.

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 MySQL], page 228.

`mysqld-max`
: A version of the server that includes additional features. See Section 5.1.2 [`mysqld-max`], page 219.

`mysqld_safe`
: A server startup script. `mysqld_safe` attempts to start `mysqld-max` if it exists, and `mysqld` otherwise. See Section 5.1.3 [`mysqld_safe`], page 221.

`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 [`mysql.server`], page 224.

`mysqld_multi`
: A server startup script that can start or stop multiple servers installed on the system. See Section 5.1.5 [`mysqld_multi`], page 224.

`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 5.6.2 [Table maintenance], page 321.

`make_binary_distribution`
: This program makes a binary release of a compiled MySQL. This could be sent by FTP to '`/pub/mysql/Incoming`' on `support.mysql.com` for the convenience of other MySQL users.

`mysqlbug`
: The MySQL bug reporting script. It can be be used to send a bug report to the MySQL 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, the 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://www.mysql.com/downloads/mysql-4.0.html`. See Section 2.2.1 [Windows installation], page 77.
- 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 already installed the regular server RPM. See Section 2.2.2 [Linux-RPM], page 89 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://www.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 used only for MySQL 3.23 because `InnoDB` is not included by default until MySQL 4. From MySQL 4 on, `InnoDB` is included by default in binary distributions, so you do not need a MySQL-Max server to obtain `InnoDB` support.

`--with-bdb`
This option enables support for the Berkeley DB (BDB) storage engine.

`CFLAGS=-DUSE_SYMDIR`
This define enables symbolic link support for Windows.

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. The following table shows which platforms allow MySQL-Max binaries to include BDB:

System	BDB Support
AIX 4.3	N
HP-UX 11.0	N
Linux-Alpha	N
Linux-IA-64	N
Linux-Intel	Y
Mac OS X	N
NetWare	N
SCO OSR5	Y
Solaris-Intel	N
Solaris-SPARC	Y
UnixWare	Y
Windows/NT	Y

To find out which storage engines your server supports, issue the following statement:

```
mysql> SHOW ENGINES;
```

Before MySQL 4.1.2, 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_bdb         | NO       |
| have_crypt       | YES      |
| have_innodb      | YES      |
| have_isam        | NO       |
| have_raid        | NO       |
| have_symlink     | DISABLED |
| have_openssl     | NO       |
| have_query_cache | YES      |
+------------------+----------+
```

The values in the second column indicate the server's level of support for each feature:

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 hostname.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 16.3 [InnoDB in MySQL 3.23], page 759. You can specify configuration options for the BDB storage engine, too, but BDB will not disable itself if you do not provide them. See Section 15.4.3 [BDB start], page 753.

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 will still attempt 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 [Server options], page 228.

Note that 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 in the [mysqld_safe] group of an option file. See Section 4.3.2 [Option files], page 212.

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, though you should rename such sections to [mysqld_safe] when you begin using MySQL 4.0 or later.)

mysqld_safe supports the following options:

`--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 5.8.1 [Error log], page 345.

`--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 will start 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 distributoins typically install the server in slightly different locations. (See Section 2.1.5 [Installation layouts], page 76.) `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 distribution at the time it was built. They should be correct if MySQL is installed in the location specified at configuration time.

Because `mysqld_safe` will try 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 'hostname.err' file in the data directory.
6. Sends mysqld_safe screen output to the 'hostname.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.

Note that if you use the Linux server RPM package (MySQL-server-VERSION.rpm), the mysql.server script will already have been installed in the '/etc/init.d' directory with the name 'mysql'. You need not install it manually. See Section 2.2.2 [Linux-RPM], page 89 for more information on the Linux RPM packages.

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 2.4.2.2 [Automatic start], page 122.

mysql.server reads options from the [mysql.server] and [mysqld] sections of option files. (For backward compatibility, it also reads [mysql_server] sections, though 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 distinquish 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 2.4.2.2 [Automatic start], page 122.) 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 5.9 [Multiple servers], page 351.

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 start 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 an example 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 clients.

`--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`.

`--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 5.4.2 [Privileges], page 276. You will 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 will restart 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 [mysqld_safe], page 221. The test performed by these lines should be successful, or you might encounter problems.

- The Unix socket file and the TCP/IP port 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 will 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 *will not* give you extra performance in a threaded system. See Section 5.9 [Multiple servers], page 351.

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 [Option files], page 212.

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 [Program Options], page 210. 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 [Option files], page 212.

`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 5.3.3 [Privileges options], page 273.

- SSL-related options: See Section 5.5.8.5 [SSL options], page 318.
- Binary log control options: See Section 5.8.4 [Binary log], page 347.
- Replication-related options: See Section 6.8 [Replication Options], page 379.
- Options specific to particular storage engines: See Section 15.1.1 [`MyISAM` start], page 740, Section 15.4.3 [`BDB` start], page 753, Section 16.5 [`InnoDB` start], page 764.

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.

`--ansi`
: Use standard SQL syntax instead of MySQL syntax. See Section 1.8.3 [ANSI mode], page 41. 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 on file. This option prevents most "table full" errors, but also slows down queries for which in-memory tables would suffice. Since Version 3.23.2, MySQL 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` will not close the console screen if this option is used.

`--character-sets-dir=path`
: The directory where character sets are installed. See Section 5.7.1 [Character sets], page 340.

`--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`.

`--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 [`mysqld_safe`], page 221. Note that on some systems, such as Solaris, you will not get a core file if you are also using the `--user` option.

`--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 D.1.2 [Making trace files], page 1200.

`--default-character-set=charset`
: Use `charset` as the default character set. See Section 5.7.1 [Character sets], page 340.

`--default-collation=collation`
: Use `collation` as the default collation. This option is available as of MySQL 4.1.1. See Section 5.7.1 [Character sets], page 340.

`--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 15 [Table types], page 737.

`--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 [Server parameters], page 437. See Section 15.1.1 [MyISAM start], page 740.

  Note: If you set this variable to ALL, you should not use `MyISAM` tables from another program (like from another MySQL server or with myisamchk) when the table is in use. Doing so, will lead 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, and XP systems, and can be used only with the `mysqld-nt` and `mysqld-max-nt` servers that support named pipe connections.

`--exit-info, -T`
: 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), you will easily get `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 uses 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 as 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 only does a write of all changes to disk after each SQL statement and lets the operating system handle the syncing to disk. See Section A.4.1 [Crashing], page 1013.

`--init-file=file`
Read SQL statements from this file at startup. Each statement must be on a single line and should not include comments.

`--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 5.7.2 [Languages], page 341.

`--log[=file], -l [file]`
Log connections and queries to this file. See Section 5.8.2 [Query log], page 346. If you don't specify a file name, MySQL will use `hostname.log` as filename.

`--log-bin=[file]`
Log all queries that change data to this file. Used for backup and replication. See Section 5.8.4 [Binary log], page 347. If you don't specify a file name, MySQL will use `hostname-bin` as filename.

`--log-bin-index[=file]`
The index file for binary log file names. See Section 5.8.4 [Binary log], page 347. If you don't specify file name, MySQL will use `hostname-bin.index` as filename.

`--log-error[=file]`
Log errors and startup messages to this file. See Section 5.8.1 [Error log], page 345. If you don't specify file name, MySQL will use `hostname.err` as 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. If you are using `--log-slow-queries` and `--log-long-format`, then queries that are not using indexes also are logged to the slow query log. Note that `--log-long-format` is deprecated as of MySQL version 4.1, when `--log-short-format` was introduced (the 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 queries log.

`--log-queries-not-using-indexes`
: If you are using this option with `--log-slow-queries`, then also queries that are not using indexes are logged to the slow query log. This option is available as of MySQL 4.1. See Section 5.8.5 [Slow query log], page 350.

`--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 options 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 file. Note that the default for the amount of information logged has changed in MySQL 4.1. See the `--log-long-format` and `--log-long-format` options for details. See Section 5.8.5 [Slow query log], page 350.

`--log-update[=file]`
: Log updates to `file.#` where `#` is a unique number if not given. See Section 5.8.3 [Update log], page 346. The update log is deprecated and is removed in MySQL 5.0.0; you should use the binary log instead (`--log-bin`). See Section 5.8.4 [Binary log], page 347. Starting from version 5.0.0, using `--log-update` will just turn on the binary log instead (see Section C.1.2 [News-5.0.0], page 1042).

`--log-warnings, -W`
: Print out warnings like `Aborted connection...` to the '`.err`' file. Enabling this option is recommended, for example, if you use replication (you will get more information about what is happening, such as messages about network failures and reconnections). This option is enabled by default as of MySQL 4.1.2; to disable it, use `--skip-log-warnings`. See Section A.2.11 [Communication errors], page 1006.

  This option was named `--warnings` before MySQL 4.0.

`--low-priority-updates`
: Table-modifying operations (`INSERT`/`DELETE`/`UPDATE`) will have lower priority than selects. It can also be done via `{INSERT | REPLACE | UPDATE | DELETE} 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], page 423.

`--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, seprate them by commas. You can also use a value of `""` to disable this option. If this option is used, `mysqld` will on open check if the table is marked as crashed or if the table wasn't closed properly. (The last option works only if you are running with `--skip-external-locking`.) If this is the case `mysqld` will run check on the table. If the table was corrupted, `mysqld` will attempt 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 table was changed during recover, save a backup of the '`tbl_name.MYD`' data file as '`tbl_name-datetime.BAK`'.
`FORCE`	Run recovery even if we will 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 will add 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 will force a repair of a table even if some rows would be deleted, but it will keep the old data file as a backup so that you can later examine what happened.

`--new` From version 4.0.12, 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:

- `TIMESTAMP` is returned as a string with the format `'YYYY-MM-DD HH:MM:SS'`. See Chapter 12 [Column types], page 533.

This option can be used to help you see how your applications will behave in MySQL 4.1, without actually upgrading to 4.1.

`--pid-file=path`
The path to the process ID file used by `mysqld_safe`.

`--port=num, -P num`
The port number to use when listening for TCP/IP connections.

`--old-protocol, -o`
: Use the 3.20 protocol for compatibility with some very old clients. See Section 2.5.6 [Upgrading-from-3.20], page 137.

`--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 D.1 [Debugging server], page 1199.

`--open-files-limit=`
: To change the number of file descriptors available to `mysqld`. If this is not set or set to 0, then `mysqld` will use this value to reserve file descriptors to use with `setrlimit()`. If this value is 0 then `mysqld` will reserve `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'.

`--safe-mode`
: Skip some optimize 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 now a `SHOW DATABASES` privilege that can be used to control access to database names on a per-account basis. See Section 14.5.1.2 [`GRANT`], page 689.

`--safe-user-create`
: If this is enabled, a user can't create new users with the `GRANT` statement, if the user doesn't have `INSERT` privilege to the `mysql.user` table or any column in this table.

`--skip-bdb`
: Disable the `BDB` storage engine. This saves memory and might speed up some operations. Do not use this operation 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 only to be used 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 [Server parameters], page 437.

`--skip-external-locking`
: Don't use system locking. To use `isamchk` or `myisamchk` you must shut down the server. See Section 1.2.3 [Stability], page 8. Note that in MySQL Version 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-IP resolution. Instead, query the DNS server every time a client connects. See Section 7.5.5 [DNS], page 443.

`--skip-innodb`
: Disable the `InnoDB` storage engine. This saves memory and disk space and might speed up some operations. Do not use this operation 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 7.5.5 [DNS], page 443.

`--skip-networking`
: Don't listen for TCP/IP connections at all. All interaction with `mysqld` must be made via named pipes (on Windows) or Unix socket files (on Unix). This option is highly recommended for systems where only local clients are allowed. See Section 7.5.5 [DNS], page 443.

`--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 [Windows symbolic links], page 446.
- 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 14.2.5 [CREATE TABLE], page 670.

This option was added in MySQL 4.0.13.

`--skip-safemalloc`
: If MySQL is configured with `--with-debug=full`, all MySQL programs checks 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 some privilege for the database or has the `SHOW DATABASES` privilege.

`--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 D.1 [Debugging server], page 1199.

`--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 1.8.2 [SQL mode], page 40. 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, as 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 14.4.6 [SET TRANSACTION], page 688.

`--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 will fail.

`--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 will change its user ID during its startup sequence, causing it to run as that particular user rather than as `root`. See Section 5.3.1 [Security guidelines], page 269.

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 '`datadir/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`' will be 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.

You can assign a value to a server system variables 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], page 240. The section on tuning server parameters includes information on how to optimize them. See Section 7.5.2 [Server parameters], page 437.

You can change the values of most system variables for a running server with the `SET` statement. See Section 14.5.3.1 [`SET OPTION`], page 702.

If you want to restrict the maximum value 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 applications to tailor server operation to their 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. 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. Setting the `GLOBAL` variable affects the operation of all clients that connect from that time on. Setting the `SESSION` variable affects only the current client. `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 value also can be empty (`--sql-mode=""`) if you want to reset it.

The following list describes the supported modes:

`ANSI_QUOTES`
: Treat ‘`"`’ as an identifier quote character (like the MySQL Server ‘`‘`’ quote character) and not as a string quote character. You can still use ‘`‘`’ to quote identifers in ANSI mode. With `ANSI_QUOTES` enabled, you cannot use double quotes to quote a literal string, because it will be intepreted as an identifier. (New in MySQL 4.0.0.)

`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_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. 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, normally MySQL 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 statements in the dump output to enable `NO_AUTO_VALUE_ON_ZERO`.) (New in MySQL 4.1.1.)

`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 field-specific 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 index-specific 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 table-specific 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 13.7 [Cast Functions], page 605. (New in MySQL 4.0.2.)

ONLY_FULL_GROUP_BY
Don't allow queries which in the GROUP BY part refers 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.)

The following special modes are provided as shorthand for combinations of mode values from the preceding list. They are available as of MySQL 4.1.1.

ANSI Equivalent to REAL_AS_FLOAT, PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, ONLY_FULL_GROUP_BY. See Section 1.8.3 [ANSI mode], page 41.

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.

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.

MYSQL40 Equivalent to NO_FIELD_OPTIONS.

ORACLE Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS.

POSTGRESQL
Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS.

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 intialized from the global variable only for clients that connect after the change. It does not affect the session variable for any client that is already 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. 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], page 260.

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 at startup 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. Many variables have both global and session values. See Section 10.4 [System Variables], page 499 for more information.

```
mysql> SHOW VARIABLES;
+---------------------------------+------------------------------+
| Variable_name                   | Value                        |
+---------------------------------+------------------------------|
| back_log                        | 50                           |
| basedir                         | /usr/local/mysql             |
| bdb_cache_size                  | 8388572                      |
| bdb_home                        | /usr/local/mysql             |
| bdb_log_buffer_size             | 32768                        |
| bdb_logdir                      |                              |
| bdb_max_lock                    | 10000                        |
| bdb_shared_data                 | OFF                          |
| bdb_tmpdir                      | /tmp/                        |
| bdb_version                     | Sleepycat Software: ...      |
| binlog_cache_size               | 32768                        |
| bulk_insert_buffer_size         | 8388608                      |
| character_set                   | latin1                       |
| character_sets                  | latin1 big5 czech euc_kr     |
| concurrent_insert               | ON                           |
| connect_timeout                 | 5                            |
| convert_character_set           |                              |
| datadir                         | /usr/local/mysql/data/       |
| default_week_format             | 0                            |
| delay_key_write                 | ON                           |
| delayed_insert_limit            | 100                          |
| delayed_insert_timeout          | 300                          |
| delayed_queue_size              | 1000                         |
| flush                           | OFF                          |
| flush_time                      | 0                            |
| ft_boolean_syntax               | + -><()~*:""&|               |
| ft_max_word_len                 | 84                           |
| ft_min_word_len                 | 4                            |
| ft_query_expansion_limit        | 20                           |
| ft_stopword_file                | (built-in)                   |
| have_bdb                        | YES                          |
| have_innodb                     | YES                          |
| have_isam                       | YES                          |
| have_openssl                    | YES                          |
| have_query_cache                | YES                          |
| have_raid                       | NO                           |
```

```
| have_symlink                    | DISABLED                     |
| init_file                       |                              |
| innodb_additional_mem_pool_size | 1048576                      |
| 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_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            | 5242880                      |
| innodb_log_files_in_group       | 2                            |
| innodb_log_group_home_dir       | ./                           |
| innodb_mirrored_log_groups      | 1                            |
| innodb_thread_concurrency       | 8                            |
| interactive_timeout             | 28800                        |
| join_buffer_size                | 131072                       |
| key_buffer_size                 | 16773120                     |
| key_cache_age_threshold         | 300                          |
| key_cache_block_size            | 1024                         |
| key_cache_division_limit        | 100                          |
| language                        | /usr/local/mysql/share/...   |
| large_files_support             | ON                           |
| local_infile                    | ON                           |
| locked_in_memory                | OFF                          |
| log                             | OFF                          |
| log_bin                         | OFF                          |
| log_slave_updates               | OFF                          |
| log_slow_queries                | OFF                          |
| log_update                      | OFF                          |
| log_warnings                    | OFF                          |
| long_query_time                 | 10                           |
| low_priority_updates            | OFF                          |
| lower_case_table_names          | 0                            |
| max_allowed_packet              | 1047552                      |
| 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_join_size                   | 4294967295                  |
| max_relay_log_size              | 0                           |
| max_sort_length                 | 1024                        |
| max_tmp_tables                  | 32                          |
| max_user_connections            | 0                           |
| max_write_lock_count            | 4294967295                  |
| myisam_max_extra_sort_file_size | 268435456                   |
| myisam_max_sort_file_size       | 2147483647                  |
| myisam_recover_options          | force                       |
| myisam_repair_threads           | 1                           |
| myisam_sort_buffer_size         | 8388608                     |
| net_buffer_length               | 16384                       |
| net_read_timeout                | 30                          |
| net_retry_count                 | 10                          |
| net_write_timeout               | 60                          |
| open_files_limit                | 1024                        |
| pid_file                        | /usr/local/mysql/name.pid   |
| port                            | 3306                        |
| protocol_version                | 10                          |
| query_cache_limit               | 1048576                     |
| query_cache_size                | 0                           |
| query_cache_type                | ON                          |
| read_buffer_size                | 131072                      |
| read_rnd_buffer_size            | 262144                      |
| rpl_recovery_rank               | 0                           |
| 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                | 2097116                     |
| sql_mode                        |                             |
| table_cache                     | 64                          |
| table_type                      | MYISAM                      |
| thread_cache_size               | 3                           |
| thread_stack                    | 131072                      |
| timezone                        | EEST                        |
| tmp_table_size                  | 33554432                    |
| tmpdir                          | /tmp/:/mnt/hd2/tmp/         |
| tx_isolation                    | READ-COMMITTED              |
| version                         | 4.0.4-beta                  |
| wait_timeout                    | 28800                       |
+---------------------------------+-----------------------------+
```

Most system variables are described here. InnoDB system variables are listed at Section 16.5 [InnoDB start], page 764.

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 [Server parameters], page 437.

The system variables have the following meanings. Variables with no version indicated have been present since at least MySQL 3.22.

`ansi_mode`
: Is ON if mysqld was started with --ansi. See Section 1.8.3 [ANSI mode], page 41. This variable was added in MySQL 3.23.6 and removed in 3.23.41.

`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(2) 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 will be 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
: 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
: The BDB storage engine version. This variable was added in MySQL 3.23.31.

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 5.8.4 [Binary log], page 347.

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, for some functions, 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 vari-

able 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
: 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.

collation_connection
: 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
: 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.

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 given 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 will speed 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 [Server options], page 228. See Section 15.1.1 [MyISAM start], page 740.

Note that --external-lock doesn't offer any protection against index corruption for tables that uses 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
How many rows to queue when handling INSERT DELAYED statements. If the queue becomes full, any client that issues an INSERT DELAYED statement will wait until there is room in the queue again.

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 will be 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 13.6.1 [Fulltext Boolean], page 600.

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 fourteen 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.

`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.

`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 an empty string ('') disables stopword filtering. This variable was added in MySQL 4.0.10.

  Note: `FULLTEXT` indexes must be rebuilt after changing this variable.

`group_concat_max_len`
: The maximum allowed result length for the `GROUP_CONCAT()` function. This variable was added in MySQL 4.1.0.

`have_bdb`
: `YES` if `mysqld` supports `BDB` tables. `DISABLED` if `--skip-bdb` is used. This variable was added in MySQL 3.23.30.

`have_innodb`
: `YES` if `mysqld` supports `InnoDB` tables. `DISABLED` if `--skip-innodb` is used. This variable was added in MySQL 3.23.37.

`have_innodb`
: `YES` if `mysqld` supports `ISAM` tables. `DISABLED` if `--skip-isam` is used. This variable was added in MySQL 3.23.30.

`have_raid`
: `YES` if `mysqld` supports the `RAID` option. This variable was added in MySQL 3.23.30.

`have_openssl`
: `YES` if `mysqld` supports SSL (encryption) of the client/server protocol. This variable was added in MySQL 3.23.43.

`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. This variable was added in MySQL 4.1.2.

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'
```

`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 16.5 [`InnoDB` start], page 764.

`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.

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. Remember that 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 14.4.5 [`LOCK TABLES`], page 686.

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 14.5.3 [`SHOW`], page 701.

The `Key_reads/Key_read_requests` ratio normally should 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_used` status variable and the buffer blocksize. 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:

```
(Key_blocks_used * key_cache_block_size) / key_buffer_size
```

Before MySQL 4.1.1, key cache blocks are 1024 bytes, so the fraction of the key buffer in use is:

```
(Key_blocks_used * 1024) / key_buffer_size
```

See Section 7.4.6 [`MyISAM` key cache], page 430.

`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 [`MyISAM` key cache], page 430.

`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 [`MyISAM` key cache], page 430.

`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 [`MyISAM` key cache], page 430.

`language`
: The language used for error messages.

`large_file_support`
: Whether `mysqld` was compiled with options for big file support. This variable was added in MySQL 3.23.28.

`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 5.8.2 [Query log], page 346.

`log_bin`
: Whether the binary log is enabled. This variable was added in MySQL 3.23.14. See Section 5.8.4 [Binary log], page 347.

`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 Options], page 379.

`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 5.8.5 [Slow query log], page 350.

`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 5.8.3 [Update log], page 346.

`long_query_time`
: If a query takes longer than this many seconds, the `Slow_queries` counter 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 5.8.5 [Slow query log], page 350.

`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_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 10.2.2 [Name case sensitivity], page 497.

Note that 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 0 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 will 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 [Table cache], page 435 for comments on file descriptor limits. See Section A.2.7 [Too many connections], page 1004.

`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 will be 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 will need to examine more than `max_join_size` row combinations or 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 already is in the query cache, no result size check is performed, because the result has already been computed and it will not burden the server to send it to the client.

  This variable previously was named `sql_max_join_size`.

`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], page 364.

`max_seeks_for_key`
: Limit the assumed maximum number of seeks when looking up rows based on a key. The MySQL optimizer will assume that no more than this number of key seeks will be required when searching for matching rows in a table by scanning a key, regardless of the actual cardinality of the key. 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`
: (This option doesn't yet do anything.) The maximum number of temporary tables a client can keep open at the same time.

`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.

`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.

`myisam_data_pointer_size`
: 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.2.12 [Full table], page 1007.

`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 will be 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.

open_files_limit
: The number of files 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.

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.

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 5.10.3 [Query Cache Configuration], page 360. 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 will be 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 will 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 variable 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.

`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.

  Note that server startup will fail 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.

`skip_external_locking`
: Is `OFF` if `mysqld` uses external locking. This variable was added in MySQL 4.0.3. Previously, it was named `skip_locking`.

`skip_networking`
: Is `ON` if we allow only local (non-TCP/IP) connections. On Unix, local connections use a Unix socket file. On Windows, local connections use a named pipe. 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 some privilege for the database or has the `SHOW DATABASES` privilege.

`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 [Temporary files], page 1017.

`sql_mode` The current server SQL mode. This variable was added in MySQL 3.23.41. See Section 1.8.2 [SQL mode], page 40.

`storage_engine`
This variable is a synonym for `table_type`. It was added in MySQL 4.1.2.

`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], page 262. 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 reopenend), then you should increase the value of the `table_cache` variable.

For more information about the table cache, see Section 7.4.8 [Table cache], page 435.

`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 [Server options], page 228.

`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 aren't already `thread_cache_size` threads there. Requests for threads are satsified 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 examing the difference between the `Connections` and `Threads_created` status variables (see Section 5.2.4 [Server status variables], page 262 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 [MySQL Benchmarks], page 399.

`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. See Section A.4.6 [Timezone problems], page 1019.

`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 will fail.

  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.

`version` The version number for the server.

`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 10.4 [System Variables], page 499.

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`
`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
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	boolean	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
max_user_connections	numeric	GLOBAL
max_write_lock_count	numeric	GLOBAL
myisam_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
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
server_id	numeric	GLOBAL
slave_compressed_protocol	boolean	GLOBAL
slave_net_timeout	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_quote_show_create	boolean	SESSION
sql_safe_updates	boolean	SESSION
sql_select_limit	numeric	SESSION
sql_slave_skip_counter	numeric	GLOBAL
sql_warnings	boolean	SESSION
storage_engine	enumeration	GLOBAL | SESSION
table_cache	numeric	GLOBAL
table_type	enumeration	GLOBAL | SESSION
thread_cache_size	numeric	GLOBAL
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 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 the 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 variable 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.2.11 [Communication errors], page 1006.

`Aborted_connects`
: The number of tries to connect to the MySQL server that failed. See Section A.2.11 [Communication errors], page 1006.

`Binlog_cache_use`
: The number of transactions that used the temporary binary log cache. This variable was added in MySQL 4.1.2.

`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.

`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_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 will be 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 will be 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 will be 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.

`Key_blocks_used`
: The number of used 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], page 240.

`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 with `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.

`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 number of key blocks in the key cache that have changed but haven't yet been flushed to disk.

`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`
: Number of free memory blocks in query cache.

`Qcache_free_memory`
: Amount of free memory for query cache.

`Qcache_hits`
: Number of cache hits.

`Qcache_inserts`
: Number of queries added to the cache.

`Qcache_lowmem_prunes`
: Number of queries that were deleted from cache because of low memory.

`Qcache_not_cached`
: Number of non-cached queries (not cachable, or due to `query_cache_type`).

`Qcache_queries_in_cache`
: Number of queries registered in the cache.

`Qcache_total_blocks`
: Total number of blocks in query cache.

`Questions`
: The number of queries that have been sent to the server.

`Rpl_status`
: 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 indexes of your tables. This variable was added in MySQL 3.23.25.

`Select_full_range_join`
: The number of joins where we used a range search on reference table. This variable was added in MySQL 3.23.25.

`Select_range`
: The number of joins where we 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 where we 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 where we 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`
: Is `ON` if this 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` 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 5.8.5 [Slow query log], page 350.

`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 a table lock was acquired immediately. This variable was added as of MySQL 3.23.33.

`Table_locks_waited`
: The number of times 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 with `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 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 5.4 [Privilege system], page 275.

5.3.1 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 which 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 item about setting a `root` password.
  - 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 where 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 the 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:
  - Users of 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.
  - Users of MySQL C API: Use the `mysql_real_escape_string()` API call.
  - Users of MySQL++: Use the `escape` and `quote` modifiers for query streams.
  - Users of Perl DBI: Use the `quote()` method or use placeholders.
  - Users of 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.

5.3.2 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 that can sniff the traffic between the client and the server can crack the password. (See Section 5.4.9 [Password hashing], page 295 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 Version 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 5.5.8 [Secure connections], page 311.

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 identify 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
  mysql> UPDATE 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 will be 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 [Changing MySQL user], page 1011.

Note that running mysql 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-symlink option.) This is especially important if you run mysqld as root, because anyone that has write access to the server's data directory could then delete any file in the system! See Section 7.6.1.2 [Symbolic links to tables], page 445.
- 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 will 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.

5.3.3 Startup Options for mysqld Concerning Security

The following mysqld options affect security:

`--local-infile[={0|1}]`
: If you start the server with `--local-infile=0`, clients cannot use `LOCAL` in `LOAD DATA` statements. See Section 5.3.4 [LOAD DATA LOCAL], page 274.

`--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 now a `SHOW DATABASES` privilege that can be used to control access to database names on a per-account basis. See Section 14.5.1.2 [GRANT], page 689.

`--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'@'hostname';
```

This will ensure that the user can't change any privilege columns directly, but has to use the `GRANT` statement to give privileges to other users.

`--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 some privilege for the database or has the `SHOW DATABASES` privilege.

5.3.4 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 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 20.2.3.40 [`mysql_options()`], page 892.
- 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 enable local data file loading. In any case, successful use of a local loading operation requires that the server is enabled to allow it.
- 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
```

5.4 The MySQL Access Privilege System

MySQL has an advanced but non-standard security/privilege system. This section describes how it works.

5.4.1 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 an anonymous user and to grant privileges for MySQL-specific functions such as `LOAD DATA INFILE` and administrative operations.

5.4.2 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**. The system grants privileges 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 `joe` one set of privileges for connections from `office.com`, and a different set of privileges for connections from `elsewhere.com`.

MySQL access control involves two stages:

- Stage 1: The server checks whether you are even allowed 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.

Note that if your privileges are changed (either by yourself or someone else) while you are connected, those changes will not necessarily take effect immediately for the next statement you issue. See Section 5.4.7 [Privilege changes], page 290 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 rereads them under the circumstances indicated in Section 5.4.7 [Privilege changes], page 290. 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 14.5.1.2 [GRANT], page 689. 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
	References_priv	References_priv	References_priv
	Reload_priv		
	Shutdown_priv		
	Process_priv		
	File_priv		
	Show_db_priv		
	Super_priv		
	Create_tmp_table_priv	Create_tmp_table_priv	Create_tmp_table_priv
	Lock_tables_priv	Lock_tables_priv	Lock_tables_priv
	Execute_priv		
	Repl_slave_priv		
	Repl_client_priv		
	ssl_type		
	ssl_cipher		
	x509_issuer		
	x509_subject		
	max_questions		
	max_updates		
	max_connections		

During the second stage of access control (request verification), the server may, if the request involves tables, additionally consult the `tables_priv` and `columns_priv` tables that provide finer control at the table and column levels. The columns in these tables are shown here:

Table Name	**tables_priv**	**columns_priv**
Scope columns	Host Db User Table_name	Host Db User Table_name Column_name
Privilege columns	Table_priv Column_priv	Column_priv
Other columns	Timestamp Grantor	Timestamp

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 entry applies. For example, a `user` table entry 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 entry 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 entry applies.
* Privilege columns indicate the privileges granted by a table entry, 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 5.4.6 [Request access], page 287.

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(60)`
`Column_name`	`CHAR(60)`

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 Version 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 disabled or enabled, with the default being disabled.

In the `tables_priv` and `columns_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'`

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 entry 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 entry, then populate the host table with an entry for each of those hosts. This mechanism is described more detail in Section 5.4.6 [Request access], page 287.

  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.

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 reread 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 5.4.7 [Privilege changes], page 290.

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. One way to check the privileges for a given account is to 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 or column privileges specified in the tables_priv or columns_priv tables.

For additional help in diagnosing privilege-related problems, see Section 5.4.8 [Access denied], page 290. For general advice on security issues, see Section 5.3 [Security], page 269.

5.4.3 Privileges Provided by MySQL

Information about account privileges is stored in the user, db, host, tables_priv, and columns_priv tables in the mysql database. The MySQL server reads the contents of these tables into memory when it starts and rereads them under the circumstances indicated in Section 5.4.7 [Privilege changes], page 290. Access-control decisions are based on the in-memory copies of the grant tables.

The names used in this manual to refer to the privileges provided by MySQL are shown in the following table, along with the table 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 14.5.1.2 [GRANT], page 689.

Privilege	**Column**	**Context**
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	Create_priv	databases, tables, or indexes
DROP	Drop_priv	databases or tables
GRANT	Grant_priv	databases or tables
REFERENCES	References_priv	databases or tables
CREATE TEMPORARY TABLES	Create_tmp_table_priv	server administration
EXECUTE	Execute_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.

The EXECUTE and REFERENCES privileges currently are unused.

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 CREATE and DROP privileges allow you to create new databases and tables, or to drop (remove) existing databases and tables. Note that 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 INDEX privilege allows you to create or drop (remove) indexes. Note that 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 GRANT privilege allows you to give to other users those privileges you yourself possess.

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. (Note that 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 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, and reload
SHUTDOWN	shutdown
PROCESS	processlist
SUPER	kill

The reload command tells the server to reread 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-* 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 and the SUPER privilege to kill them. See Section 14.5.4.3 [KILL], page 724. 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 when they 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 privileges only to those accounts that need them, but you should exercise particular caution in granting administrative privileges:

- 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 `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 `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.

5.4.4 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 will prompt 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 5.5.7 [Password security], page 310.

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 [Option files], page 212.
- 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 5.5.7 [Password security], page 310. For a list of variables, see Appendix E [Environment variables], page 1211.

5.4.5 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 Version 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 entry 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 5.4.6 [Request access], page 287.

  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 will never match 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 entry 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.) Note that 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 5.4.9 [Password hashing], page 295.

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, etc. (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 entry 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 entry 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'` entry 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 entry, 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 will be 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 entry containing 'jeffrey' as the User column value, but by the entry with no username! (As a result, jeffrey will be 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 entry 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 13.8.3 [Information functions], page 611. 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.

5.4.6 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 5.4.2 [Privileges], page 276, 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 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 will 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 entry because that is the only table that specifies administrative privileges. Access is granted if the entry allows the requested operation and denied otherwise. For example, if you want to execute mysqladmin shutdown but your user table entry 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, etc.), the server first checks the user's global (superuser) privileges by looking in the user table entry. If the entry 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 the user wants to access. If there is no entry for the Host and User, access is denied.
2. If there is a matching db table entry and its Host column is not blank, that entry defines the user's database-specific privileges.

3. If the matching db table entry'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 entry 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 entry 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 entry 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 entry grants one privilege and the db table entry 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 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 mysqlaccess) to make sure that your access privileges are actually set up the way you think they are.

5.4.7 When Privilege Changes Take Effect

When mysqld starts, all grant table contents are read into memory and become effective at that point for access control.

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 will have *no effect* until you restart the server. This may leave you wondering why your changes don't seem to make any difference!

5.4.8 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, 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 than 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.

- The grant tables must be properly set up so that the server can use them for access control. For installations on Windows using a binary distribution or on Linux using a server RPM distribution, the installation process initializes the mysql database containing the grant tables. For other MySQL installation types, you should initialize the grant tables manually by running the mysql_install_db script. See Section 5.5.2 [Default privileges], page 301.

  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 5.5.2 [Default privileges], page 301.
- 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 you should always make sure after an upgrade that your tables have the current structure. For instructions, see Section 2.5.8 [Upgrading-grant-tables], page 139.
- 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 instructions on how to deal with this, see Section 5.4.9 [Password hashing], page 295.
- If you try to connect as root and get the following error, it means that you don't have an entry 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 will 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 with the --no-defaults option as follows:

  ```
  shell> mysqladmin --no-defaults -u root ver
  ```

  The option files that clients use are listed in Section 4.3.2 [Option files], page 212. Environment variables are listed in Appendix E [Environment variables], page 1211.
- 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 5.5.6 [Passwords], page 308.

If you have lost the `root` password, you can restart `mysqld` with `--skip-grant-tables` to change the password. See Section A.4.2 [Resetting permissions], page 1015.

- 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 will not work. For example, the following statement sets a password, but fails to encrypt it, so the user will not be 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 5.5.6 [Passwords], page 308.

- `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 entry 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 entry with the `Host` and `User` columns matching your computer's hostname and your MySQL username.

- The `Access denied` error message will tell 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 entry 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 invoking the command-line tool `mysql` on the server host and 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 entry 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 will indicate how you really did connect. (Then replace the `'%'` in the `user` table entry with 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 7.5.5 [DNS], page 443.

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 entry 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 will 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 doesn't work, 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 will allow 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 issue a FLUSH PRIVILEGES statement or execute a mysqladmin flush-privileges command to cause the server to reread the privilege tables. Otherwise, your changes have no effect until the next time the server is restarted. Remember that after you set 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 5.4.7 [Privilege changes], page 290.
- 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. (Note that 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 will prompt 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 will print host and user information about attempted connections, as well as information about each command issued. See Section D.1.2 [Making trace files], page 1200.
- 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 1.7.1.3 [Bug reports], page 34. In some cases, you may need to restart `mysqld` with `--skip-grant-tables` to run `mysqldump`.

5.4.9 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.

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 5.4.9.2 [Password hashing 4.1.0], page 299.

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 will be 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, though 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 now 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 will 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 will 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 will only generate 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.

5.4.9.1 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 will break. 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()`.

5.4.9.2 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.

5.5 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 secure the initial MySQL accounts that are created when you install MySQL
- 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

5.5.1 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 will be 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 13.8.2 [Encryption functions],

page 608. 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. (Their names and access privileges are discussed in Section 5.5.2 [Default privileges], page 301.) Thereafter, you normally set up, modify, and remove MySQL accounts using the `GRANT` and `REVOKE` statements. See Section 14.5.1.2 [`GRANT`], page 689.

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
```

Note that there must be no space between the `-p` option and the following password value. See Section 5.4.4 [Connecting], page 282.

If you want the client program to ask for a password, you should use a `--password` or `-p` option without any following password value:

```
shell> mysql --user=monty --password db_name
shell> mysql -u monty -p db_name
```

In these examples, `db_name` is **not** the password. Instead, the program will print a prompt and wait for you to enter the password.

On some systems, the library call that MySQL uses to prompt for a password automatically limits the password to eight characters. Internally, MySQL doesn't have any limit for the length of the password.

5.5.2 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 2.4.2 [Unix post-installation], page 116.

The grant tables define the initial accounts and their access privileges. These accounts are set up as follows:

- Two accounts are created with a username of `root` and no password. These are superuser accounts that can do anything. Connections for `root` must be made from the local host, either by specifying a hostname of `localhost` or the actual hostname or IP number.

  Note: 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.

- Two anonymous-user accounts are created with an empty username and no password. These accounts can do anything with databases that have a name of 'test' or that start with 'test_'. Connections must be made from the local host, either by specifying a hostname of localhost or the actual hostname or IP number. Because these accounts have no passwords, any local user can connect without a password and be treated as the anonymous user.
- Other privileges are denied. For example, normal users can't perform administrative operations by executing commands such as mysqladmin shutdown or mysqladmin processlist.

Note: The default privileges are different for Windows. See Section 2.4.1 [Windows post-installation], page 115.

As noted, none of the initial accounts have passwords. Because your installation is initially wide open, one of the first things you should do is to secure these accounts:

- You should assign passwords to the MySQL root accounts.
- If you want to prevent clients from connecting as anonymous users without a password, you should also assign passwords to the anonymous accounts or else remove them.

You can assign passwords to the root accounts several ways. The following discussion demonstrates three methods:

- By using the SET PASSWORD statement
- By using the mysqladmin command-line client program
- By using the UPDATE statement

To assign passwords using SET PASSWORD, connect to the server as root and issue two SET PASSWORD statements, as follows:

```
shell> mysql -u root mysql
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR 'root'@'host_name' = PASSWORD('newpwd');
```

Replace "newpwd" with the actual root password that you want to use. Be sure to encrypt the password using the PASSWORD() function. In the second command, 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 the SET PASSWORD statements:

```
mysql> SELECT Host, User FROM User;
```

Look for the record that has root in the User column and something other than localhost in the Host column.

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"
```

Replace "newpwd" with the actual root password that you want to use. The double quotes are not always necessary, but you should use them if the password contains spaces or other characters that are special to your command interpreter. In the second command, replace host_name with the name of the server host.

Another way to assign passwords 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 following UPDATE statement assigns a password to both root accounts at once:

```
shell> mysql -u root mysql
mysql> UPDATE user SET Password = PASSWORD('newpwd')
    ->     WHERE User = 'root';
mysql> FLUSH PRIVILEGES;
```

Replace "newpwd" with the actual root password that you want to use. Be sure to encrypt the password using the PASSWORD() function.

Note that after you update the password in the user table directly using UPDATE, you should tell the server to reread the grant tables with FLUSH PRIVILEGES. Otherwise, the change will go unnoticed otherwise until you restart the server.

After the root password has been set, you must supply that password whenever you connect to the server as root.

You might want to defer setting a password for the root accounts until later, so that you don't need to specify it while you perform additional setup or testing. However, be sure to set it before using your installation for any real production work.

If you want 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, do this:

```
shell> mysql -u root mysql
mysql> SET PASSWORD FOR ''@'localhost' = PASSWORD('newpwd');
mysql> SET PASSWORD FOR ''@'host_name' = PASSWORD('newpwd');
```

The value of host_name is the same as when you assigned the root account passwords.

To use UPDATE, do this:

```
shell> mysql -u root mysql
mysql> UPDATE user SET Password = PASSWORD('newpwd')
    ->     WHERE User = '';
mysql> FLUSH PRIVILEGES;
```

If you prefer to remove the anonymous accounts instead, do so as follows:

```
shell> mysql -u root mysql
mysql> DELETE FROM user WHERE User = '';
mysql> FLUSH PRIVILEGES;
```

5.5.3 Adding New User Accounts to MySQL

You can create MySQL accounts 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 14.5.1.2 [GRANT], page 689.

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 5.5.2 [Default privileges], page 301. This means that to make changes, you must be on the same machine where mysqld is running, you must connect to it 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 should specify that password for this mysql command and also for those later in this section.

After connecting to the server, 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 5.4.5 [Connection access], page 284.)
- 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-* 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 reread the grant tables. Otherwise, the changes will 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 Version 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;
```

5.5.4 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 14.5.1.1 [Drop user], page 689.

5.5.5 Limiting Account Resources

Before MySQL 4.0.2, the only available method for limiting use of MySQL server resources has been 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.

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 resource-related columns. Resource limits are stored in the max_questions, max_updates, and max_connections columns. If your user table doesn't have these columns, it needs to be upgraded, see Section 2.5.8 [Upgrading-grant-tables], page 139.

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;
```

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 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).

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 resource-use counts can be reset globally for all accounts, or individually for a given count:

- 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 already has.

5.5.6 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 5.5.1 [User names], page 300.

5.5.7 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 `-pyour_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
  Enter password: ********
  ```

  The '`*`' characters represent 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 [Option files], page 212 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 will be 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 E [Environment variables], page 1211.

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.

5.5.8 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 uncrypted connection or a secure encrypted SSL connection according the requirements of individual applications.

5.5.8.1 Basic SSL Concepts

To understand how MySQL uses SSL, it's necessary to explain some basic SSL and X509 concepts. People who are already 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 will 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.

5.5.8.2 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 2.5.8 [Upgrading-grant-tables], page 139.
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.

5.5.8.3 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.

5.5.8.4 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 14.5.1.2 [GRANT], page 689.

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 unneccessary.

  ```
  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 will be 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.

5.5.8.5 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 are compiled without SSL support, a normal unencrypted connection will be used.

  The secure way to ensure that a SSL connection will be 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.

5.5.8.6 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 now have an ODBC connection to MySQL, encrypted using SSH.

5.6 Disaster Prevention and Recovery

This section discusses how to make database backups and how to perform table maintenance. The syntax of the SQL statements described here is given in Section 14.5 [Database Administration], page 689.

5.6.1 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 14.4.5 [`LOCK TABLES`], page 686. See Section 14.5.4.2 [`FLUSH`], page 722. You only need 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 already 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 14.1.7 [SELECT], page 643. See Section 14.5.2.2 [BACKUP TABLE], page 697.

Another way to back up a database is to use the mysqldump program or the mysqlhotcopy script. See Section 8.8 [mysqldump], page 477. See Section 8.9 [mysqlhotcopy], page 483.

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 will 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 5.8.4 [Binary log], page 347. 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.

If your MySQL server is a slave replication server, then regardless of the backup method you choose, you should also backup the 'master.info' and 'relay-log.info' files when you backup 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 something, try to recover your tables 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 will work only if you have enabled binary logging by starting MySQL with the --log-bin option; see Section 5.8.4 [Binary log], page 347.

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 [mysqlbinlog], page 470 for more information on the mysqlbinlog utility and how to use it.

   If you are using the update logs instead, 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 [Replication Intro], page 363.

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.

5.6.2 Using myisamchk for 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 15 [Table types], page 737.

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.

Note that 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)

Note that myisamchk operations that affect indexes can cause FULLTEXT indexes to be rebuilt will full-text parameters that are incompatible with the values used by the MySQL server. To avoid this, read the instructions in Section 5.6.2.2 [myisamchk general options], page 323.

An alternative to using myisamchk for checking or repairing MyISAM tables is to use the CHECK TABLE or REPAIR TABLE statements, which are available as of MySQL 3.23.13. See Section 14.5.2.3 [CHECK TABLE], page 697 and Section 14.5.2.6 [REPAIR TABLE], page 700.

In most cases, you can also use the OPTIMIZE TABLES statement to optimize and repair tables, but this is not as fast or reliable (in case of really serious problems) as myisamchk. On the other hand, OPTIMIZE TABLE is easier to use and you don't have to worry about flushing tables. See Section 14.5.2.5 [OPTIMIZE TABLE], page 699.

The mysqlcheck client program provides a command-line interface to the CHECK TABLE, REPAIR TABLE, and OPTIMIZE TABLE statements.

5.6.2.1 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 5.6.2.6 [myisamchk memory], page 328.

You must ensure that no other program is using the tables while you are running myisamchk.

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 (like 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. In MySQL Version 3.23, the easiest way to avoid this problem is to use CHECK TABLE instead of myisamchk to check tables.

5.6.2.2 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 only be locked 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:

- If you use --safe-recover.
- If 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.

The ft_min_word_len, ft_max_word_len, and myisam_block_size variables are available as of MySQL 4.0.0. ft_stopword_file is available as of MySQL 4.0.19.

myisam_block_size is the size used for index blocks.

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.

5.6.2.3 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 will help 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.

5.6.2.4 Repair Options for `myisamchk`

`myisamchk` supports the following options for table repair operations:

`--backup, -B`
: Make a backup of the '.MYD' file as 'filename-time.BAK'

`--character-sets-dir=path`
: The directory where character sets are installed. See Section 5.7.1 [Character sets], page 340.

`--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 to recover every possible row from the data file. Normally this will also find 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 a symlink points to. This option doesn't exist as of MySQL 4.0, because versions from 4.0 on will 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`. (Note that 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.

`--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 will be used successively in 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`.

5.6.2.5 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`
Prints 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 already 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 #`
Sorts 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.

5.6.2.6 `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` will also need a lot of disk space:

- Double the size of the record 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 record 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 will 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=#`).

If you have a problem with disk space during repair, you can try to use `--safe-recover` instead of `--recover`.

5.6.2.7 Using myisamchk for Crash Recovery

If you run mysqld with --skip-external-locking (which is the default on some systems, like 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 will 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.4.1 [Crashing], page 1013.

The MyISAM table section contains reason for why a table could be corrupted. See Section 15.1.4 [MyISAM table problems], page 744.

When performing crash recovery, it is important to understand that each 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.

5.6.2.8 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.

5.6.2.9 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.

Starting with MySQL Version 3.23.14, you can repair `MyISAM` tables with the `REPAIR TABLE` command. See Section 14.5.2.6 [`REPAIR TABLE`], page 700.

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), is not an error 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`.

You can also use this technique for error 136 (no more room in index file).

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 `cd` to the database directory and check the permissions of the table files. Make sure that they are readable by the Unix 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.

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 14.5.2.3 [CHECK TABLE], page 697 and Section 14.5.2.6 [REPAIR TABLE], page 700.

The options that you can use for table maintenance with `myisamchk` and `isamchk` are described in several of the earlier subsections of Section 5.6.2 [Table maintenance], page 321.

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 will still be 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 will attempt 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 will remove incorrect records and deleted records from the data file and reconstruct 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 only reach this stage 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 SQL version 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 now. (This shouldn't be an endless loop.)

As of MySQL 4.0.2, you can also use `REPAIR TABLE ... 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.

5.6.2.10 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 14.5.2.5 [OPTIMIZE TABLE], page 699.

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 5.6.2.1 [myisamchk syntax], page 322.

5.6.3 Setting Up a Table Maintenance Regimen

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 Version 3.23.13. See Section 14.5.2.3 [CHECK TABLE], page 697 and Section 14.5.2.6 [REPAIR TABLE], page 700.

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. (This is an "expected crashed table.") 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 we can examine and repair them when needed.

As we haven't had any unexpectedly crashed tables (tables that become corrupted for reasons other than hardware trouble) for a couple of years now (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 you don't need to maintain MySQL tables that much. If you are changing tables with dynamic-sized rows (tables with `VARCHAR`, `BLOB` or `TEXT` columns) or have tables with many deleted rows you may want to from time to time (once a month?) defragment/reclaim space from the tables.

You can do this by using `OPTIMIZE TABLE` on the tables in question. 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> myisamchk -r -s --sort-index  -O sort_buffer_size=16M */*.MYI
```

5.6.4 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.

Example 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
```

An 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
```

An 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
Datafilepointer (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
```

An 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
```

An 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 5.6.2.10 [Optimisation], page 333.

- `Data file 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 5.6.2.10 [Optimisation], page 333.

- `Data file 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's data file can become, in bytes.
- `Max keyfile length`

  How long the table's 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 with `myisamchk`. See Section 5.6.2.10 [Optimisation], page 333.

- `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`], page 449, for an example of this information and a description of what it means.

5.7 MySQL Localization and International Usage

5.7.1 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 USA 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 will be 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-charset | complex | all | none` options to `configure`, and the character set configuration files listed in '`SHAREDIR/charsets/Index`'. See Section 2.3.2 [`configure` options], page 101.

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 2.3.2 [`configure` options], page 101.

If you change the character set when running MySQL, that may also change the sort order. Consequently, 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 will switch 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 will need 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=character-set-name
```

This is normally unnecessary, however.

5.7.1.1 Using the German Character Set

To get German sorting order, you should start mysqld with a --default-character-set=latin1_de option. This will give you the following characteristics:

- 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.

5.7.2 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 a the --languag 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
```

Note that 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.

5.7.3 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 5.7.4 [Character arrays], page 343.
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 5.7.4 [Character arrays], page 343.
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 5.7.5 [String collating], page 344.
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 1.7.1.1 [Mailing-list], page 32.

5.7.4 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 will be case-insensitive. MySQL will sort characters based on the value of sort_order[character]. For more complicated sorting rules, see the discussion of string collating in Section 5.7.5 [String collating], page 344.

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
```

5.7.5 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.

Right now the best documentation for this is the character sets that are already implemented. 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).

5.7.6 Multi-byte Character Support

If your want to add support for a new character set that includes multi-byte characters, you need to use the multi-byte character functions.

Right now the best documentation on this is the character sets that are already implemented. 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.

5.7.7 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 will display 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 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	Problems encountering 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	Established client connections and executed statements.
The update log	Statements that change data. This log is deprecated.
The binary log	Stores all statements that change data. Also used for replication.
The slow log	Stores 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 14.5.4.2 [FLUSH], page 722.

If you are using MySQL replication capabilities, slave replication servers maintain additional log files called relay logs. These are discussed in Chapter 6 [Replication], page 363.

5.8.1 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 will write 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 will contain a stack trace if mysqld dies. The trace can be used to determine where mysqld died. See Section D.1.4 [Using stack trace], page 1202.

Beginning with MySQL 4.0.10, you can specify where mysqld stores the error log file with the option --log-error[=file_name]. If no file_name value is given, mysqld uses the name ''hostname'.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 will be 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 'hostname'.err. You could change this filename by specifying a --err-log=filename 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.

5.8.2 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 'hostname.log' This will log 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 safe_mysqld a --log option 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 5.8.4 [Binary log], page 347.

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.

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.

5.8.3 The Update Log

Note: The update log has been deprecated and replaced by the binary log. See Section 5.8.4 [Binary log], page 347. 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 the value it already has.

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.

5.8.4 The Binary Log

The binary log has replaced the old update log, which is 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, like the old update log, only logs 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 the value it already has.

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 5.8.2 [Query log], page 346.

The primary purpose of the binary log is to be able to update the database during a restore operation as fully as possible, as the binary log will contain 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], page 363.

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.

If you supply an extension in the log name (for example, --log-bin=filename.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 this 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 14.5.4.5 [RESET], page 725 and Section 14.6.1 [Replication Master SQL], page 726.

You can use the following options to mysqld to affect what is logged to the binary log. Note 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 others databases that are not explicitly mentioned are ignored. Note that 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-do-db=sales, and you do USE prices; UPDATE sales.january SET amount=amount+1000;, this statement will not be 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. Note that 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 will be written into the binary log.

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 in the following order:

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. So 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.

So for example, a slave running with only `binlog-do-db=sales` will 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 will ever need 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 3 days old. You can remove them manually, or preferably using `PURGE MASTER LOGS` (see Section 14.6.1 [Replication Master SQL], page 726) which will also safely update 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 14.5.3.1 [`SET`], page 702.

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 [`mysqlbinlog`], page 470 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 will be 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 `max_binlog_cache_size` (default 4G) can be used to restrict the total size used to cache a multiple-statement transaction. If a transaction is larger than this, it will fail and roll back.

If you are using the update log or binary log, concurrent inserts will be 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.

5.8.5 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 thus 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.

If you also use the `--log-long-format` when logging slow queries, then queries that are not using indexes are logged as well. See Section 5.2.1 [Server options], page 228.

5.8.6 Log File Maintenance

The MySQL Server can create a number of different log files, which make it easy to see what is going on. See Section 5.8 [Log Files], page 345. 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 will want to backup and remove old log files from time to time and tell MySQL to start logging to new files. See Section 5.6.1 [Backup], page 319.

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. Note that 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 Version 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 flush the logs and then move away the old update log files to a backup. If you are using the normal logging, you can do something like:

```
shell> cd mysql-data-directory
shell> mv mysql.log mysql.old
shell> mysqladmin flush-logs
```

Then make a backup and remove '`mysql.old`'.

5.9 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 [Program Options], page 210.

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 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 5.8.6 [Log file maintenance], page 350.

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 for the reason that it is 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 file names that are unique to each server. Otherwise, the servers will try to log to the same files.

This 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 will become the speed bottleneck. It is not meant for such use.
- Another risk with NFS is that you will have 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 will perform 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 will 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` will determine 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 will be more convenient to use option files to specify for each server those option values that must be unique to it.

5.9.1 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 2.2.1 [Windows installation], page 77. 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 5.9 [Multiple servers], page 351.)

5.9.1.1 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 fo 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:\> mysqld --defaults-file=C:\my-opts1.cnf
C:\> mysqld-max --defaults-file=C:\my-opts2.cnf
```

On NT, each server will start 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:\> mysqladmin --port=3307 shutdown
C:\> mysqladmin --port=3308 shutdown
```

Servers configured as just described will 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:\> mysqld-nt --defaults-file=C:\my-opts1.cnf
```

'C:\my-opts2.cnf' would be modified similarly for use by the second server.

5.9.1.2 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 2.2.1.7 [NT start], page 83.

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` will 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..

5.9.2 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` will default 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> /path/to/mysqld_safe --socket=file_name --port=port_number
```

To start a second server, provide different `--socket` and `--port` option values, and pass a 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> scripts/mysql_install_db
shell> bin/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 will apply to any client programs that you invoke from the same shell. Thus, connections for those clients automatically will be directed to the second server!

Appendix E [Environment variables], page 1211 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-to-option-file
```

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 [mysqld_multi], page 224.

5.9.3 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 E [Environment variables], page 1211.
- 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 [Option files], page 212.
- 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 20.2.3 [C API functions], page 866.

- 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 20.6 [Perl], page 967.

  Other programming interfaces may provide similar capabilities for reading option files.

5.10 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.

Note: The query cache does not return stale data. When data is modified, any relevant entries in the query cache are flushed.

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: It doesn't 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 with 2GB RAM and a 64MB query cache.

- If all of 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 is normally 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.

5.10.1 How the Query Cache Operates

This section describes how the query cache works when it is operational. Section 5.10.3 [Query Cache Configuration], page 360 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 otther reasons. Queries that uses different databases, uses 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 5.10.4 [Query Cache Status and Maintenance], page 361.

If a table changes then all cached queries that used 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.

A query cannot be cached if it contains any of the following functions:

Function	**Function**	**Function**
`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 will not be 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 20.3.6 [ODBC and last_insert_id], page 965.
- It uses `TEMPORARY` tables.

- It does not use any tables.
- The user has a column 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 will not be used.

The query cache does work 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.

5.10.2 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.

5.10.3 Query Cache Configuration

Several mysqld 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], page 240, 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.

Assuming that the query cache is enabled, the query_cache_type variable influences how it works. This variable 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 only of those statements that begin with SELECT SQL_CACHE.

Setting the GLOBAL 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 is usually 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 (time wise), 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 cause the query cache to have to 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 you 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.

5.10.4 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], page 262. Some uses for them are described here.

<total number of SELECT queries> = `Com_select + Qcache_hits + `<queries with errors found by parser>

`Com_select` = `Qcache_inserts + Qcache_not_cached + `<queries with errors found during fields/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.

Note: 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 5.10.3 [Query Cache Configuration], page 360.

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 14.6 [Replication SQL], page 726.

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 5.6.1 [Backup], page 319.

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 5.8.4 [Binary log], page 347.

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 will 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 will take 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 will simply connect to the master and wait for updates to process. If the master goes away or the slave loses connectivity with your master, it will keep 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], page 388.

6.3.1 Replication Master Thread States

The following list shows the most common states you will 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 will 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 will occur. At that point, the thread will consider 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 will become `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 will 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.nnn`', where `host_name` is the name of the slave server host and `nnn` is a sequence number. Successive relay log files are created using successive sequence numbers, beginning with `001`. 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 Options], page 379.

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 will influence 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 14.6.2 [Replication Slave SQL], page 728 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_RELAY_LOG and MASTER_RELAY_POS options to tell the slave to reread 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 will 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 already have a snapshot of the master's data, and the master already 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], page 388.

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 14.6.1 [Replication Master SQL], page 726 and Section 14.6.2 [Replication Slave SQL], page 728. You should also familiarize yourself with replication startup options described in Section 6.8 [Replication Options], page 379.

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], page 374.

   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 CLIENT 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 will 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 will be 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 will 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 will be empty. In that case, the values that you will 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
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 will be set to 1 if you have not defined `master-host`, else it will be set to 2. Note that in the case of `server-id` omission, a master will refuse connections from all slaves, and a slave will refuse 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, to get more messages about problems (for example, network or connection problems).
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 will not be able to connect to it.

If you have forgotten to set the `server-id` value for the slave, you will get the following error in its error log:

```
Warning: one should set server_id to a non-0 value if
master_host is set.  The server will not act as a slave.
```

You will 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 will 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 Options], page 379 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 [Replication Upgrade], page 374.

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 3.23.33 and up	**Master 4.0.3 and up or any 4.1.x**	**Master 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.

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], page 374.

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 already 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 will unfortunately confuse the 4.0 or 4.1 slave.

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 will block on the slave and return 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], page 374.

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 C.1.2 [News-5.0.0], page 1042). 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 or 4.0 or 4.1 to 5.0.0, you should first ensure that all the slaves of this master are already 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 will be 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 will be 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 will be able to read its old binary logs that were written before the upgrade and send them to the 5.0.0 slaves. The slaves will recognize the old format and handle it properly. Binary logs created by master after the upgrade will be in 5.0.0 format. These too will be 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

The following list explains what is supported and what is not. Additional InnoDB-specific information about replication is given in Section 16.7.5 [InnoDB and MySQL Replication], page 775.

- Replication will be done correctly with AUTO_INCREMENT, LAST_INSERT_ID(), and TIMESTAMP values.
- The USER(), UUID(), and LOAD_FILE() functions are replicated without changes and will thus 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 FOREIGN_KEY_CHECKS variable is replicated as of version 4.0.14. The sql_mode, UNIQUE_CHECKS, and SQL_AUTO_IS_NULL variables are replicated as of 5.0.0. The SQL_SELECT_LIMIT and table_type variables are not yet replicated.
- You must use the same character set (--default-character-set) 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 character set may not be unique in the slave character set. Character sets will be replicated in 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, you will have problems if the slave is stopped in the middle of a BEGIN/COMMIT block, because the slave will restart at the beginning of the BEGIN block. This issue is on our TODO and will be fixed in the near future.
- Update statements that make use of 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 version 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.
- 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 will simply ignore 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.
- Although we have never heard of it actually occurring, it is theoretically 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 1.8.7.3 [Open bugs], page 53.

- 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 as 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 will 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 14.5.4.2 [FLUSH], page 722.
- 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` for the table to its binary log. See Section 15.3 [`HEAP`], page 749 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 temporary 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 will 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 will know when an event that it reads was originally created by itself and will 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 will be no infinite loop. But this circular setup will work 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 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 will also be able to deal with network connectivity outages. However, the slave will 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], page 240.
- 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 will be greatly increased if you have a good uninterruptible power supply.
- 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 will exit and wait for the DBA 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 will be 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` will cause 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 already 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()` for 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 5.8.4 [Binary log], page 347.

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 will occur 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 will have 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 14.6.2.1 [CHANGE MASTER TO], page 728.

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 about what it is doing. For example, it will warn you that it succeeded in reconnecting after a network/connection failure, and warn you about how each slave thread started. This option is enabled by default as of MySQL 4.1.2; to disable it, use `--skip-log-warnings`.

  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 will not be started. 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 5.5.8.5 [SSL options], page 318. 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], page 240.

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.nnn`, where `host_name` is the name of the slave server host and `nnn` 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 on 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 now 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 will exceed 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 will wait for free space because `--relay-log-space-limit` is exceeded, but the SQL thread will have no relay log to purge and be 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 will 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 will not be 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

will work 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-table 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 will be 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 will work 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 will work for cross-database updates. Please read the notes that follow this option list.

Example: --replicate-wild-do-table=foo%.bar% will replicate 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 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 will work for cross-database updates. Please read the notes that follow this option list.

  Example: `--replicate-wild-ignore-table=foo%.bar%` will 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-ignore-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 will 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 will 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 binlog 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 will appear 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=#`
: If this option is set to 1, use compression on the slave/client 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 will be huge, too. Therefore, it might be advisable to use this option to 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 will be 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=#`
: 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 will result in slaves becoming hopelessly out of sync with the master, and you will 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`. A full list of error messages can be found in the source distribution in the '`Docs/mysqld_error.txt`' file. The server error codes also are listed at Section 21.1 [Error-returns], page 969.

You can (but should not) also use the very non-recommended value of `all` which will ignore all error messages and keep 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 will be 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 5.8.4 [Binary log], page 347). 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` will be 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 already 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 already, 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 will 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 will take 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 PROCESSLIST` 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], page 364.

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 will block until the slave reaches the specified log file and offset. At that point, the slave will be in sync with the master and the statement will return.
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 will make 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 will produce 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 will be a little less lock contention, because the updates originating on another server will be 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 will use 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 will give 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 will take 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 will pay off in the long run. All applications that use the approach just described will be able to take advantage of a master/slave configuration, even one involving multiple slaves. The code will be a lot easier to maintain, and adding troubleshooting options will be trivial. You will 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 already, 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 already 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 already.

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 just shown that will accurately predict 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 will monitor 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 will be 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 will contain exactly 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. Now 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 will 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` will be 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], page 364. 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 now 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], page 376. If it is a bug, see Section 6.11 [Replication Bugs], page 394 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 will go out of sync, and you can have unique key violations on updates. In this case, the slave thread will stop and wait 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 will 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.

7 MySQL Optimization

Optimization is a complex task because ultimately it requires understanding of the whole system. Although it may be possible to perform some local optimizations with small knowledge of your system or application, the more optimal you want your system to become, the more you will 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 some (increasingly harder) additional ways to make the system even faster.

7.1 Optimization Overview

The most important factor in making a system fast is the basic design. You also need to know what kinds of things your system will be doing, and what your bottlenecks are.

The most common 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 spread the data on more than one disk.
- Disk reading/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 (or if it was already there), we need to process it to get to our result. Having small tables compared to the 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, the 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 mix of a steady stream of updates and slow selects on the same table. If this is a problem with some tables, you can use another table type for these. See Chapter 15 [Table types], page 737.

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:

- All columns have default values.
- If you insert a "wrong" value in a column, such as a too-large numerical value into a numerical column, MySQL will set 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 in the column.

- All calculated expressions returns a value that can be used instead of signaling an error condition. For example, 1/0 returns `NULL`

The implication of these rules is that you should not use MySQL to check column content. Instead, you should check values in the application before storing them in the database.

For more information about this, see Section 1.8.6 [Constraints], page 50 and Section 14.1.4 [INSERT], page 630.

7.1.2 Designing Applications for Portability

Because all SQL servers implement different parts of 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 choose a number of SQL servers that it should work with.

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` tests far from everything possible, but it is still comprehensive with about 450 things tested.

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.

For information, visit `http://www.mysql.com/information/crash-me.php`.

Both the MySQL benchmarks and `crash-me` programs are very database independent. By taking a look at how we have written them, 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 in the MySQL source distribution. 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.

See `http://www.mysql.com/information/benchmarks.html` for the results from the benchmarks.

As you can see in the results, all database systems have some weak points. That is, they have different design compromises that lead to different behavior.

If you strive for database independence, you need to get a good feeling for each SQL server's bottlenecks. MySQL is very fast in retrieving and updating records, but will have 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 will allow 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 `/**/` will be 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 5.10 [Query Cache], page 358.

7.1.3 What We Have Used MySQL For

During MySQL initial development, the features of MySQL were made to fit our largest customer. They handle data warehousing for a couple of the biggest retailers in Sweden.

From all stores, we get weekly summaries of all bonus card transactions, and we are expected to provide useful information for the store owners to help them find how their advertisement campaigns are affecting their own customers.

The volume of data is quite huge (about seven million summary transactions per month), and we have data for 4-10 years that we need to present to the users. We got weekly requests from our customers, who want to get "instant" access to new reports from this data.

We solved this by storing all information per month in compressed 'transaction' tables. We have a set of simple macros (script) that generates summary tables grouped by different criteria (product group, customer id, store, ...) from the transactional tables. The reports are Web pages that are dynamically generated by a small Perl script that parses a Web page, executes the SQL statements in it, and inserts the results. We would have used PHP or mod_perl instead, but they were not available at that time.

For graphical data, we wrote a simple tool in `C` that can process SQL query results and produce GIF images based on those results. This is also dynamically executed from the Perl script that parses the Web pages.

In most cases, a new report can be done simply by copying an existing script and modifying the SQL query in it. In some cases, we will need to add more fields to an existing summary table or generate a new one, but this is also quite simple, because we keep all transactions tables on disk. (Currently we have at least 50GB of transactions tables and 200GB of other customer data.)

We also let our customers access the summary tables directly with ODBC so that the advanced users can experiment with the data themselves.

We haven't had any problems handling this with quite modest Sun Ultra SPARCstation hardware (2x200MHz). We recently upgraded one of our servers to a 2 CPU 400MHz UltraSPARC, and we are now planning to start handling transactions on the product level, which would mean a tenfold increase of data. We think we can keep up with this by just adding more disk to our systems.

We are also experimenting with Intel-Linux to be able to get more CPU power cheaper. Now that we have the binary portable database format (new in Version 3.23), we will start to use this for some parts of the application.

Our initial feelings are that Linux will perform much better on low-to-medium load and Solaris will perform better when you start to get a high load because of extreme disk I/O, but we don't yet have anything conclusive about this. After some discussion with a Linux kernel developer, this might be a side effect of Linux allocating so many resources to the batch job that the interactive performance gets very low. This makes the machine feel very slow and unresponsive while big batches are running. Hopefully this will be better handled in future Linux Kernels.

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 is not written yet. 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.

The following tables show some comparative benchmark results for several database servers when accessed through ODBC on a Windows NT 4.0 machine.

Reading 2000000 rows by index	**Seconds**	**Seconds**
mysql	367	249
mysql_odbc	464	
db2_odbc	1206	
informix_odbc	121126	
ms-sql_odbc	1634	
oracle_odbc	20800	
solid_odbc	877	
sybase_odbc	17614	

Inserting 350768 rows	**Seconds**	**Seconds**
mysql	381	206
mysql_odbc	619	
db2_odbc	3460	
informix_odbc	2692	
ms-sql_odbc	4012	
oracle_odbc	11291	

solid_odbc	1801
sybase_odbc	4802

For the preceding tests, MySQL was run with an index cache size of 8MB.

We have gathered some more benchmark results at `http://www.mysql.com/information/benchmarks.html`.

Note that Oracle is not included because it asked to be removed. All Oracle benchmarks have to be passed by Oracle! We believe that makes Oracle benchmarks **very** biased because the above benchmarks are supposed to show what a standard installation can do for a single client.

To use the benchmark suite, the following requirements must be satisfied:

- The benchmark suite is provided with MySQL source distributions, so you must have a source distribution. You can either download a released distribution from `http://www.mysql.com/downloads/`, or use the current development source tree (see Section 2.3.3 [Installing source tree], page 104).
- The benchmark scripts are written in Perl and use the Perl DBI module to access database servers, so DBI must be installed. You will 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 20.6 [Perl], page 967.

The benchmark suite is located in the '`sql-bench`' directory of MySQL source distributions. To run the benchmark tests, change location into that 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 supported servers. To get a list of all options and supported servers, invoke `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://www.mysql.com/information/crash-me.php`.

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 [MySQL Benchmarks], page 399. 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 is really 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 already 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 your 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 that affects all statements: The more complex your permission setup is, the more overhead you will 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:

```
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(loop_count,expression)` 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 will execute 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 the records.

You should frequently run ANALYZE TABLE to update table statistics such as cardinality of keys, which can affect the choices the optimizer makes. See Section 14.5.2.1 [ANALYZE TABLE], page 696.

You can also see whether the optimizer joins the tables in an optimal order. To force the optimizer to use a specific join order for a SELECT statement, add a STRAIGHT_JOIN clause.

For single-table joins, 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 will 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
: SELECT identifier, the sequential number of this SELECT within the query.

select_type
: The type of SELECT clause, which can be any of the following:

  SIMPLE
  : Simple SELECT (not using UNION or subqueries)

  PRIMARY
  : Outermost SELECT

  UNION
  : Second and further SELECT statements in a UNION

  DEPENDENT UNION
  : Second and further SELECT statements in a UNION, dependent on outer subquery

`SUBQUERY` First `SELECT` in subquery

`DEPENDENT SUBQUERY`
First `SELECT` in subquery, dependent on outer subquery

`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 will be 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 constants:

```
SELECT * FROM const_table WHERE primary_key=1;

SELECT * FROM const_table
WHERE primary_key_part1=1 AND primary_key_part2=2;
```

`eq_ref` One row will be 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 compared item may be a constant or an expression that uses columns from tables that are read before this table.

In the following examples, `ref_table` will be able to use `eq_ref`:

```
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 will be 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, ref_table will be able to use ref:

```
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 will do an extra search for rows that contain NULL values.

In the following example, ref_table will be able to use ref_or_null:

```
SELECT * FROM ref_table
WHERE key_column=expr OR key_column IS NULL;
```

This join type optimization is new for MySQL 4.1.1 and is mostly used when resolving subqueries.

See Section 7.2.6 [IS NULL optimisation], page 412.

index_merge
: This join type indicates that the Index Merge optimization is used. For more information, see Section 7.2.5 [OR optimizations], page 412. In this case, the key column contains a list of used indexes, and key_len contains a list of the longest key parts for the used indexes.

unique_subquery
: This type replaces ref for some IN subqueries of the following form:

```
value IN (SELECT primary_key
              FROM single_table WHERE some_exp)
```

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_field
              FROM single_table WHERE some_exp)
```

range
: Only rows that are in a given range will be 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 will be NULL for this type.

range can be used for when an key column is compared to a constant using any of the =, <>, >, >=, <, <=, IS NULL, <=>, BETWEEN, or IN operators:

```
SELECT * FROM range_table
WHERE key_column = 10;

SELECT * FROM range_table
WHERE key_column BETWEEN 10 and 20;

SELECT * FROM range_table
WHERE key_column IN (10,20,30);

SELECT * FROM range_table
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 is usually faster than `ALL`, because the index file is usually 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 will be 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` may 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 14.2.2 [ALTER TABLE], page 663.

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 14.1.7 [SELECT], page 643.

Running `ANALYZE TABLE` or `myisamchk --analyze` on the table will help the optimizer choose better indexes. See Section 14.5.2.1 [ANALYZE TABLE], page 696 and Section 5.6.2.1 [myismchk syntax], page 322.

`key_len` The `key_len` column indicates the length of the key that MySQL decided to use. The length is `NULL` if the `key` is `NULL`. Note that this tells us how many parts of a multi-part key MySQL will actually use.

`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 will resolve the query. Here is an explanation of the different text strings that can be found in this column:

`Distinct` MySQL will stop 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 will 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 with `NOT NULL`. In this case, MySQL will scan `t1` and look 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 will 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. Instead, for each row combination in the preceding tables, it will do a check to determine which index to use (if any), and use it to retrieve the rows from the table. This is not very fast, but is faster than performing a join with no index at all.

`Using filesort`

MySQL will need 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.

`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 will need 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 will be used to restrict which rows will be matched against the next table or sent 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 you don't have this information 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 `Using filesort` and `Using temporary`.

You can get a good indication of how good a join is by multiplying all values in the `rows` column of the `EXPLAIN` output. This should tell you roughly how many rows MySQL must examine to execute the query. This number is also used when you restrict queries with the `max_join_size` variable. See Section 7.5.2 [Server parameters], page 437.

The following example shows how a `JOIN` can be optimized progressively using 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 indexes shown here:

Table	**Index**
`tt`	`ActualPC`

tt	AssignedPC
tt	ClientID
et	EMPLOYID (primary key)
do	CUSTNMBR (primary key)

- The tt.ActualPC values aren't 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 will take 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't (yet) use indexes on columns efficiently if they are declared differently. 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);
```

Now 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 now 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);
```

Now 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 isn't the case for the tt table. Fortunately, it is easy to tell MySQL about this:

```
mysql> ANALYZE TABLE tt;
```

Now 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 will 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. A 500,000-row table with an index length of 3 bytes (medium integer) gives you: `log(500,000)/log(1024/3*2/(3+4)) + 1` = 4 seeks.

This index would require storage of about 500,000 * 7 * 3/2 = 5.2M (assuming a typical index buffer fill ration of 2/3), so you will probably have much of the index in memory and you will probably need only one or two calls to read data to find the row.

For writes, however, you will 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 will slowly degenerate by log N! As long as everything is cached by the OS or SQL server, things will go only marginally slower while the table gets bigger. After the data gets too big to be cached, things will start to go much slower until your applications is only bound by disk-seeks (which increase by log N). To avoid this, increase the index cache as the data grows. See Section 7.5.2 [Server parameters], page 437.

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 [MySQL indexes], page 427 and Section 7.2.1 [EXPLAIN], page 402.

Some general tips for speeding up queries:

- 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 will use 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. For `MyISAM` tables, `myisamchk --description --verbose` also 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

The `WHERE` optimizations are put in the `SELECT` part here because they are mostly used with `SELECT`, but the same optimizations apply for `WHERE` in `DELETE` and `UPDATE` statements.

Note that this section is incomplete. MySQL does many optimizations, and we have not had time to document them all.

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()...).
- 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 the following tables are used as constant tables:

  ```
  mysql> SELECT * FROM t WHERE primary_key=1;
  mysql> 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 in GROUP BY come from the same table, then this 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 will use an in-memory temporary table.
- Each table index is queried, and the best index that spans fewer than 30% of the rows is used. If no such index can be found, a quick table scan is used.
- 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:

```
mysql> SELECT COUNT(*) FROM tbl_name;
mysql> SELECT MIN(key_part1),MAX(key_part1) FROM tbl_name;
mysql> SELECT MAX(key_part2) FROM tbl_name
    ->     WHERE key_part_1=constant;
mysql> SELECT ... FROM tbl_name
    ->     ORDER BY key_part1,key_part2,... LIMIT 10;
mysql> 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:

```
mysql> SELECT key_part1,key_part2 FROM tbl_name WHERE key_part1=val;
mysql> SELECT COUNT(*) FROM tbl_name
    ->     WHERE key_part1=val1 AND key_part2=val2;
mysql> 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:

```
mysql> SELECT ... FROM tbl_name
    ->     ORDER BY key_part1,key_part2,... ;
mysql> SELECT ... FROM tbl_name
    ->     ORDER BY key_part1 DESC,key_part2 DESC,... ;
```

7.2.5 How MySQL Optimizes `OR` Clauses

The `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.

In `EXPLAIN` output, this method appears as `index_merge` in the `type` column. In this case, the `key` column contains a list of used indexes, and `key_len` contains a list of the longest key parts for the used indexes.

Examples:

```
SELECT * FROM table WHERE key_col1 = 10 OR key_col2 = 20;

SELECT * FROM table
    WHERE (key_col1 = 10 OR key_col2 = 20) AND nonkey_col=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)
```

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.

7.2.6 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 will be 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 can additionally optimize the combination col_name = expr AND col_name IS NULL, a form that is common in resolved subqueries. EXPLAIN will show 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 t2 (a,b):

```
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 after that a separate search for rows with a NULL key value.

Note that the optimization can handle only one IS NULL level.

```
SELECT * FROM t1,t2
     WHERE (t1.a=t2.a AND t2.a IS NULL)
     OR (t1.b=t2.b AND t2.b IS NULL);
```

In the preceding case, MySQL will use key lookups only on the part (t1.a=t2.a AND t2.a IS NULL) and not be able to use the key part on b.

7.2.7 How MySQL Optimizes DISTINCT

DISTINCT combined with ORDER BY will need a temporary table in many cases.

Note that because DISTINCT may use GROUP BY, you should be aware of how MySQL works with in fields in ORDER BY or HAVING that are not part of the selected fields. See Section 13.9.3 [GROUP-BY-hidden-fields], page 624.

When combining LIMIT row_count with DISTINCT, MySQL will stop as soon as it finds row_count unique rows.

If you don't use columns from all tables named in a query, MySQL will stop scanning the not-used tables as soon as it finds the first match.

```
SELECT DISTINCT t1.a FROM t1,t2 where t1.a=t2.a;
```

In this case, assuming that `t1` is used before `t2` (check with `EXPLAIN`), then MySQL will stop reading from `t2` (for that particular row in `t1`) when the first row in `t2` is found.

7.2.8 How MySQL Optimizes LEFT JOIN and RIGHT JOIN

`A LEFT JOIN B join_condition` in MySQL is implemented as follows:

- The table `B` is set to be dependent on table `A` and all tables on which `A` is dependent.
- The table `A` is set to be dependent on all tables (except `B`) that are used in the `LEFT JOIN` condition.
- The `LEFT JOIN` condition is used to decide how we should 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 it is dependent on. 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`, then MySQL will stop 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`.

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 will do a full scan on `b` because the `LEFT JOIN` will force it to be read before `d`:

```
SELECT * FROM a,b LEFT JOIN c ON (c.key=a.key) LEFT JOIN d (d.key=a.key)
         WHERE b.key=d.key
```

The fix in this case is to change the query to:

```
SELECT * FROM b,a LEFT JOIN c ON (c.key=a.key) LEFT JOIN d (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.column would be `NULL`:

```
SELECT * FROM t1 LEFT JOIN t2 ON (column) 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.column=t2.column;
```

This can be made faster as MySQL can now use table `t2` before table `t1` if this would result in a better query plan. To force a specific table order, use `STRAIGHT JOIN`.

7.2.9 How MySQL Optimizes ORDER BY

In some cases, MySQL can use an index to satisfy an `ORDER BY` or `GROUP BY` request 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 will use the index to resolve the `ORDER BY / GROUP 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 WHERE key_part1=constant GROUP 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` (note that MySQL will still use indexes to find the rows that match the `WHERE` clause):

- 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 not-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.

In those cases where MySQL must sort the result, it uses the following algorithm:

1. Read all rows according to key or by table scanning. Rows that don't match the `WHERE` clause are skipped.
2. Store the sort key values in a buffer. The size of the buffer is the value of `sort_buffer_size`.
3. When the buffer gets full, run a qsort on it and store the result in a temporary file. Save a pointer to the sorted block. (If all rows 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 (`read_rnd_buffer_size`. The code for this step is in the '`sql/records.cc`' source file.

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`], page 402.

If you want to increase `ORDER BY` speed, first see whether you can get MySQL to use indexes instead of using 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 disk 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 on different **physical** disks, not different partitions of the same disk.

By default, MySQL sorts all `GROUP BY x,y,...` queries as if you specified `ORDER BY x,y,...` in the query as well. If you include the `ORDER BY` clause explicitly that contains the same column list, MySQL optimizes it away without any speed penalty, though 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`:

```
INSERT INTO foo
SELECT a,COUNT(*) FROM bar GROUP BY a ORDER BY NULL;
```

7.2.10 How MySQL Optimizes `LIMIT`

In some cases, MySQL will handle the 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 it normally 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` will 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.

7.2.11 Speed of INSERT Queries

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 up the `bulk_insert_buffer_size` variable to make it even faster. See Section 5.2.3 [Server system variables], page 240.
- If you are inserting a lot of rows from different clients, you can get higher speed by using the `INSERT DELAYED` statement. See Section 14.1.4 [INSERT], page 630.
- Note that with `MyISAM` tables you can insert rows at the same time `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 14.1.5 [LOAD DATA], page 635.
- It is possible, with some extra work, 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 will remove all use of all indexes for the table.
  4. Insert data into the table with `LOAD DATA INFILE`. This will not update any indexes and will therefore be very fast.
  5. If you are going to only read the table in the future, use `myisampack` to make it smaller. See Section 15.1.3.3 [Compressed format], page 744.

6. Re-create the indexes with `myisamchk -r -q /path/to/db/tbl_name`. This will create 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 table; the main difference is that you can let `myisamchk` allocate much more temporary memory for the index creation that 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 `FLUSH TABLES` steps.

- You can speed up multiple-statement `INSERT` operations that are done with multiple statements by locking your tables:

```
mysql> LOCK TABLES a WRITE;
mysql> INSERT INTO a VALUES (1,23),(2,34),(4,33);
mysql> INSERT INTO a VALUES (8,26),(6,29);
mysql> 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. Locking is 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, but the maximum wait time for individual threads might go up because they wait for locks. For example:

```
Thread 1 does 1000 inserts
Threads 2, 3, and 4 do 1 insert
Thread 5 does 1000 inserts
```

If you don't use locking, 2, 3, and 4 will finish before 1 and 5. If you use locking, 2, 3, and 4 probably will 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 will 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 both `LOAD DATA INFILE` and `INSERT`, enlarge the key buffer. See Section 7.5.2 [Server parameters], page 437.

7.2.12 Speed of UPDATE Queries

Update queries are optimized as a SELECT query with the additional overhead of a write. The speed of the write is dependent on the size of the data that is being updated and the number of indexes that are updated. Indexes that are not changed will not be 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 14.5.2.5 [OPTIMIZE TABLE], page 699.

7.2.13 Speed of DELETE Queries

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 [Server parameters], page 437.

If you want to delete all rows in the table, use TRUNCATE TABLE tbl_name rather than DELETE FROM tbl_name. See Section 14.1.9 [TRUNCATE], page 661.

7.2.14 Other Optimization Tips

This section lists a number of miscellaneous tips for improving query processing speed:

- Use persistent connections to the database to avoid the 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 [Server parameters], page 437.
- 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], page 402.
- Try to avoid complex SELECT queries on MyISAM tables that are updated frequently. This is 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 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=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 will use dynamic record format if it includes a single variable-length column. See Chapter 15 [Table types], page 737.
- It's not normally 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 and don't need most of the columns. See Chapter 15 [Table types], page 737.
- 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 table SET count=count+1 WHERE index_column=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 will also give better performance with most databases, because the row locking manager in this case will have less to do.
- If you need to collect statistics from large log tables, use summary tables instead of scanning the whole 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 only created from summary tables that are generated from the actual 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 need 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 you need these 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 bigger 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 when using a dynamic table format with MyISAM tables. See Section 14.5.2.5 [OPTIMIZE TABLE], page 699.
- Use HEAP tables when possible to get more speed. See Chapter 15 [Table types], page 737.
- 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 Version 3.23, you got slow joins otherwise.

  Try to keep the 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 will involve fewer disk accesses. You will 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], page 363.

- 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 tables are open, you should ensure that they are okay 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 How MySQL Locks Tables

You can find a discussion about different locking methods in the appendix. See Section D.4 [Locking methods], page 1206.

Except for InnoDB and BDB storage engines, all locking in MySQL is deadlock-free for storage engines that use table-level locking. This include the MyISAM, MEMORY (HEAP), and ISAM engines. 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.

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.

The 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 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 will wait until there are no more updates.

To work around this for the case where you want to do many INSERT and SELECT operations on a table, 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;
```

You can use the LOW_PRIORITY options with INSERT, UPDATE, or DELETE, or HIGH_PRIORITY with SELECT if you want to prioritize retrieval in some specific cases. You can also start mysqld with --low-priority-updates to get the same behavior.

Using SQL_BUFFER_RESULT can also help to make the duration of table locks shorter. See Section 14.1.7 [SELECT], page 643.

You could also 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.

7.3.2 Table Locking Issues

The table locking code in MySQL is deadlock free.

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 large tables, table locking is much better than row locking for most applications, but there are some pitfalls.

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.

As of MySQL Version 3.23.7 (3.23.25 for Windows), you can insert rows into a MyISAM table at the same time other threads are reading from it. Note that currently this works at the time the insert is made only if there are no holes resulting from rows having been deleted from the middle of the table. When all holes have been filled with new data, concurrent inserts are re-enabled automatically.

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. You can change this behavior by using LOW_PRIORITY with update statements or HIGH_PRIORITY with SELECT statements.)

Starting from MySQL Version 3.23.7, you can use the max_write_lock_count 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.

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 access the problem table will also be 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 a used table. This client will wait until the SELECT is finished.

- Another client issues another `SELECT` statement on the same table. Because `UPDATE` has higher priority than `SELECT`, this `SELECT` will wait for the `UPDATE` to finish. It will also wait 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 will give all statements that update (modify) a table lower priority than `SELECT` statements. In this case, the second `SELECT` statement in the previous scenario would execute before the `INSERT` statement, and would not need to wait for the first `SELECT` to finish.
- You can give a specific `INSERT`, `UPDATE`, or `DELETE` statement lower priority with the `LOW_PRIORITY` attribute.
- Start `mysqld` with a low value for `max_write_lock_count` to allow `READ` locks after a certain number of `WRITE` locks.
- You can specify that all updates issued by a specific thread should be done with low priority by using the SQL statement: `SET LOW_PRIORITY_UPDATES=1`. See Section 14.5.3.1 [SET], page 702.
- You can specify that a specific `SELECT` is very important with the `HIGH_PRIORITY` attribute. See Section 14.1.7 [SELECT], page 643.
- If you have problems with `INSERT` combined with `SELECT`, switch to using `MyISAM` tables, which support concurrent `SELECT` and `INSERT` statements.
- If you mainly mix `INSERT` and `SELECT` statements, the `DELAYED` attribute to `INSERT` will probably solve your problems. See Section 14.1.4 [INSERT], page 630.
- If you have problems with mixed `SELECT` and `DELETE` statements, the `LIMIT` option to `DELETE` may help. See Section 14.1.1 [DELETE], page 626.

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 will cause 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 will 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 lose a lot of space, because you must duplicate indexes from the nodes (because you can't store the row in the nodes).
- Deletes will 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 smaller 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 15 [Table types], page 737.

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 15.1.3 [`MyISAM` table formats], page 742.
- The primary index of a table should be as short as possible. This makes identification of one 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 will give you more hits in the index cache and thus fewer disk seeks. See Section 7.5.2 [Server parameters], page 437.
- 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 15 [Table types], page 737. 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.

For CHAR and VARCHAR columns, you can index a prefix of a column. This is much faster and requires less disk space than indexing the whole column. The syntax to use in the CREATE TABLE statement to index a column prefix looks like this:

```
INDEX index_name (col_name(length))
```

The example here creates an index for the first 10 characters of the name column:

```
mysql> CREATE TABLE test (
    ->        name CHAR(200) NOT NULL,
    ->        INDEX index_name (name(10)));
```

BLOB and TEXT columns can be indexed as well, but for these types an index prefix is mandatory, not optional. The prefix may be up to 255 bytes long.

As of MySQL Version 3.23.23, you can also create FULLTEXT indexes. They are used for full-text search. 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 13.6 [Fulltext Search], page 597 for details.

As of MySQL Version 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 supports hash indexes. As of MySQL Version 4.1.0, the engine also supports B-tree indexes.

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 [Indexes], page 426).

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:

```
mysql> 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 will be used in the following queries:

```
mysql> SELECT * FROM test WHERE last_name='Widenius';

mysql> SELECT * FROM test
    ->     WHERE last_name='Widenius' AND first_name='Michael';

mysql> SELECT * FROM test
    ->     WHERE last_name='Widenius'
    ->     AND (first_name='Michael' OR first_name='Monty');

mysql> SELECT * FROM test
    ->     WHERE last_name='Widenius'
    ->     AND first_name >='M' AND first_name < 'N';
```

However, the `name` index will **not** be used in the following queries:

```
mysql> SELECT * FROM test WHERE first_name='Michael';

mysql> 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 bigger 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 14.2.4 [`CREATE INDEX`], page 669.

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.

- 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 will do 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 will return at once:

```
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_part_1,key_part_2). If all key parts are followed by DESC, the key is read in reverse order. See Section 7.2.9 [ORDER BY optimisation], page 415.
- 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 will find 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:

```
mysql> SELECT * FROM tbl_name WHERE col1=val1;
mysql> SELECT * FROM tbl_name WHERE col2=val2;
mysql> 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).

An index is used for columns that you compare with the =, >, >=, <, <=, or BETWEEN operators.

MySQL also uses indexes 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:

```
mysql> SELECT * FROM tbl_name WHERE key_col LIKE 'Patrick%';
mysql> 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 will not use indexes:

```
mysql> SELECT * FROM tbl_name WHERE key_col LIKE '%Patrick%';
mysql> 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 will use 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 will use 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_part_3=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_part_1 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 will 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 more than 30% of the rows in the table. (In this case, a table scan is probably much faster, because it will require many fewer seeks.) However, if such a query uses LIMIT to only retrieve part of the rows, MySQL will use 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 = or <=> comparisons (but are VERY fast).
- The optimizer cannot use a hash index to speed up ORDER BY operations. (The 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 MEMORY table.
- Only whole keys can be used to search for a row. (With a B-tree index, any 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 operation:

- 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 may 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 10.4.1 [Structured System Variables], page 501 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 medium-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 will remain in the cache.
- An update operation most frequently executed for temporary tables is performed much faster when the updated node already 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 already will be in the 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 will be 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 will be 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 still will be 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 will be 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 `key_cache_block_size` variable will be 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 will 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 file, 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 [Table cache], page 435.

7.4.8 How MySQL Opens and Closes Tables

The `table_cache`, `max_connections`, and `max_tmp_tables` system variables affect the maximum 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, though 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.2.18 [Not enough file handles], page 1009.

The cache of open tables will be 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). 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 14.1.3 [HANDLER], page 628.

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 done a lot of `FLUSH TABLES`, you should increase your table cache size. See Section 14.5.3.16 [SHOW STATUS], page 716 and Section 5.2.4 [Server status variables], page 262.

7.4.9 Drawbacks to Creating Large Numbers of Tables in the Same Database

If you have many `MyISAM` or `ISAM` tables in a database directory, open, close, and create operations will be slow. If you execute `SELECT` statements on many different tables, there will be 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/Compile Time 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 really 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 2.2 kernel has really good SMP support). Also, 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 LFS (large filesystem) patch for the ext2 filesystem. Other filesystems such as ReiserFS and XFS do not have this 2GB limitation.

Because we have not run MySQL in production on that many platforms, we advise you to test your intended platform before choosing it, if possible.

Other tips:

- If you have enough RAM, you could remove all swap devices. Some operating systems will 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 default for Linux because Linux file locking is not yet safe.

  Note that the `--skip-external-locking` option will 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`/`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], page 240 and Section 5.2.4 [Server status variables], page 262.

MySQL uses algorithms that are very scalable, so you can usually run with very little memory. However, normally you will 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 \
           -O 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 will contain 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 will not be reflected in the output.

7.5.3 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 will 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 (it is not needed). Note that with some versions of `pgcc`, the resulting code will run 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 SQL 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 on.

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` 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. (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 will be 8-11% slower than connecting to the local server on the same host, even for connections over 100Mb 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 will be 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 will be 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 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 option `-mcpu=v8 -Wa,-xarch=v8plusa` 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.

7.5.4 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 [Server parameters], page 437.
- Each connection uses some thread-specific space: a stack (default 64KB, variable `thread_stack`), a connection buffer (variable `net_buffer_length`), and 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 is 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 in MySQL before Version 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 14.5.3.1 [SET Syntax], page 702.

In MySQL Version 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 will affect 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 [Temporary files], page 1017.
- 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. Normally 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 [Table cache], page 435.
- A FLUSH TABLES statement or mysqladmin flush-tables command closes all tables that are not in use and marks all in-use tables to be closed when the currently executing thread finishes. This effectively frees most in-use memory.

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 commercial memory-leakage detectors, so there should be no memory leaks.

7.5.5 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 and 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 will 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 [Symbolic links], page 444.

  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 will 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 [Custom Benchmarks], page 400.

Note that 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. Note that 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 will 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 will work 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
```

Now, for any table `tbl_a` in `db1`, there also appears to be a table `tbl_a` in `db2`. If one thread updates `db1.tbl_a` and another thread updates `db2.tbl_a`, there will be problems.

If you really need to do this, you must change the following code in '`mysys/mf_format.c`':

```
if (flag & 32 || (!lstat(to,&stat_buff) && S_ISLNK(stat_buff.st_mode)))
```

to

```
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 [Windows symbolic links], page 446.

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 will be 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 doing `SHOW VARIABLES LIKE 'have_symlink'`.

In MySQL 4.0, symlinks are fully supported only for `MyISAM` tables. For other table types, you will probably 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 will 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 14.2.5 [`CREATE TABLE`], page 670.
- `myisamchk` will 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 and the old symlinks and the files to which they pointed are deleted. (In other words, the new table will not be symlinked.)
- If you are not using symlinks, you should use the `--skip-symlink` 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 the 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 said previously, only the data and index files can be symbolic links). Attempting to do this (for example, to make synonyms) will produce 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
  ```

  Now there will be problems if one thread reads `db1.tbl1` and another thread updates `db1.tbl2`:
  - The query cache will be fooled (it will believe `tbl1` has not been updated so will return out-of-date results).
  - The `ALTER` commands on `tbl2` will also fail.

7.6.1.3 Using Symbolic Links for Databases on Windows

Beginning with MySQL Version 3.23.16, the `mysqld-max` and `mysql-max-nt` servers in the MySQL distribution 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, though the procedure for setting up the link is different.

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.

For example, if the MySQL data directory is 'C:\mysql\data' and you want to have database foo located at 'D:\data\foo', you should create the file 'C:\mysql\data\foo.sym' that contains the pathname D:\data\foo\. After that, all tables created in the database foo will be created in 'D:\data\foo'. (The 'D:\data\foo' directory must already exist for this to work.) Also, note that the symbolic link will not be used if a directory with the database name exists in the MySQL data directory. This means that if you already have a database directory named 'foo' in the data directory, you must move it to 'D:\data' before the symbolic link will be effective. (To avoid problems, the server should not be running when you move the database directory.)

In 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
```

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], page 209 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`], page 449.

`mysql`
: The command-line tool for interactively entering SQL statements or executing them from a file in batch mode. See Section 8.3 [`mysql`], page 456.

`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`], page 466.

`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 [`mysqlbinlog`], page 470.

`mysqlcc`
: A client that provides a graphical interface for interacting with the server. See Section 8.6 [`mysqlcc`], page 472.

`mysqlcheck`
: A table-maintenance client that checks, repairs, analyzes, and optimizes tables. See Section 8.7 [Using `mysqlcheck`], page 474.

`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 [`mysqldump`], page 477.

`mysqlhotcopy`
: A utility that quickly makes backups of `MyISAM` or `ISAM` tables while the server is running. See Section 8.9 [`mysqlhotcopy`], page 483.

`mysqlimport`
: A client that imports text files into their respective tables using `LOAD DATA INFILE`. See Section 8.10 [`mysqlimport`], page 484.

`mysqlshow`
: A client that displays information about databases, tables, columns, and indexes. See Section 8.11 [`mysqlshow`], page 486.

`perror`
: A utility that displays the meaning of system or MySQL error codes. See Section 8.12 [`perror`], page 488.

`replace`
: A utility program that changes strings in place in files or on the standard input. See Section 8.13 [`replace` utility], page 488.

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 5.5.7 [Password security], page 310.

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 [Program Options], page 210.

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 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` will print 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 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 will occupy one bit.

`empty-zero`
The number of columns containing values that are only binary zeros; these will 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 will contain 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 will contain 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 5.7.1 [Character sets], page 340.

`--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 5.7.1 [Character sets], page 340.

`--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 [Server SQL mode], page 237.

`--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], page 460.

`--no-tee`
: Do not copy output to a file. Tee files are discussed further in Section 8.3.1 [`mysql` Commands], page 460.

`--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], page 460.

`--password[=password], -p[password]`
: The password to use when connecting to the server. Note that if you use the short option form (`-p`), you *cannot* have a space between the option and the password. If no password is given on the command line, you will be 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], page 460.

`--protocol={TCP | SOCKET | PIPE | MEMORY}`
: The connection protocol to use. New in MySQL 4.1.

`--quick, -q`
: Don't cache each query result, print it row-by-row. 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], page 464 for more information about this option.

`--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], page 460.

`--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_histfile`' 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.
```

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 will be 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 will be 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 now 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 will 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 will be 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

'\' 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` option 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 [Option files], page 212.) The overlap may cause you problems if you use single backslashes. For example, `\s` will be 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 already running mysql, you can execute an SQL script file using the source or \. command:

```
mysql> source filename;
mysql> \. filename;
```

For more information about batch mode, see Section 3.5 [Batch mode], page 196.

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. 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 will delete 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 14.5.3.1 [SET], page 702.

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 will 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 will immediately and automatically try 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 databasename`
: Create a new database.

`drop databasename`
: Delete a database 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.

`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.

`ping`
: Check whether the server is alive.

`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 5.8.5 [Slow query log], page 350.

`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 5.7.1 [Character sets], page 340.

`--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'`.

`--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. Note that if you use the short option form (`-p`), you *cannot* have a space between the option and the

password. If no password is given on the command line, you will be 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 number of seconds before connection timeout. (Default value is 0.)

`shutdown_timeout`
: The number of seconds to wait for shutdown. (Default value is 0.)

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. However, this is deprecated because we instead want to make it easy to to apply binary logs to a local MySQL server.

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 5.8.4 [Binary log], page 347.

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
: Force reading of unknown binary log events.

--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. Note that if you use the short option form (-p), you *cannot* have a space between the option and the password. If no password is given on the command line, you will be 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`
: Start reading the binary log at position `N`.

`--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.

`--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 5.6.1 [Backup], page 319):

```
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.

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 will cause 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 will drop the temporary table. When the second mysql process attempts to use the table, the server will report "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 5.3.4 [LOAD DATA LOCAL], page 274.

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 will be 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 intertwined statements originating from different clients that used temporary tables of the same name. This is fixed in MySQL 4.1.

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 not included with MySQL distributions, but can be downloaded separately at http://www.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. Note that if you use the short option form (`-p`), you *cannot* have a space between the option and the password. If no password is given on the command line, you will be 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 will be 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 will repair tables by default.

The following names can be used to change `mysqlcheck` default behavior:

`mysqlrepair`	The default option will be `--repair`
`mysqlanalyze`	The default option will be `--analyze`
`mysqloptimize`	The default option will be `--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 5.7.1 [Character sets], page 340.

`--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 5.7.1 [Character sets], page 340.

`--extended, -e`
: If you are using this option to check tables, it ensures that they are 100% consistent but will take 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. Note that if you use the short option form (`-p`), you *cannot* have a space between the option and the password. If no password is given on the command line, you will be 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 will contain SQL statements to create the table and/or populate the table.

If you are doing a backup on the server, you should consider using the `mysqlhotcopy` instead. See Section 8.9 [mysqlhotcopy], page 483.

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 will be 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` will load the whole result set into memory before dumping the result. This will probably be 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 and you are going to generate a dump that will be reloaded into a very old MySQL server, you should not use the `--opt` or `-e` options.

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.

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 7.2.11 [Insert speed], page 417.

--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.

--compatible=name
: Produce output that is compatible with other database systems or with older MySQL servers. The value of name can be mysql323, mysql40, postgresql, oracle, mssql, db2, sapdb, 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 [Server SQL mode], page 237.

: 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`
To dump several databases. Note the difference in usage. In this case, no tables are given. All name arguments on the command line are regarded as database names. A `USE db_name` statement is 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 5.7.1 [Character sets], page 340. If not specified, `mysqldump` 10.3 (MySQL-4.1.2) or later uses `utf8`; earlier versions use `latin1`.

`--delayed`
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`. 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 14.1.5 [`LOAD DATA`], page 635.

`--first-slave, -x`
Locks all tables across all databases.

`--flush-logs, -F`
Flush the MySQL server log files before starting the dump. Note that if you use this option in combination with the `--all-databases` (or `-A`) option, the logs are flushed *for each database dumped*.

`--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`.

`--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.

Please note that when dumping multiple databases, `--lock-tables` locks tables for each database separately. So, using this option will not guarantee that the tables in the dump file will be logically consistent between databases. Tables in different databases may be dumped in completely different states.

`--master-data`
This option is like `--first-slave`, but also produces `CHANGE MASTER TO` statements that will make your 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 the slave.

`--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. Note that if you use the short option form (`-p`), you *cannot* have a space between the option and the password. If no password is given on the command line, you will be 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.

`--single-transaction`
This option issues a `BEGIN` SQL statement before dumping data from the server. It is mostly useful with `InnoDB` tables and `READ COMMITTED` transaction isolation level, because in this mode it will dump the consistent state of the database at the time then `BEGIN` was issued without blocking any applications.

When using this option, you should keep in mind that only transactional tables will be 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 version 4.0.2. This option is mutually exclusive with the `--lock-tables` option, because `LOCK TABLES` causes any pending transactions to be committed implicitly.

`--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`
Produces 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 use a MySQL account that has the `FILE` privilege, and the server must have permission to write files in the directory you specify.

`--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 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. When creating multiple-row-insert statements (as with option `--extended-insert` or `--opt`), `mysqldump` will create rows up to `max_allowed_packet` length. If you increase this variable, you should also ensure that the `max_allowed_packet` variable in the MySQL server is at least this large.

`net_buffer_length`
The initial size of the buffer for client/server communication.

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 entire databases.

```
shell> mysqldump --opt db_name > backup-file.sql
```

You can read the dump file back into the server with:

```
shell> mysql db_name < backup-file.sql
```

Or:

```
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
```

For more information on making backups, see Section 5.6.1 [Backup], page 319.

8.9 The mysqlhotcopy Database Backup Program

`mysqlhotcopy` is a Perl script that 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 on NetWare as of MySQL 4.0.18.

```
shell> mysqlhotcopy db_name [/path/to/new_directory]
shell> mysqlhotcopy db_name_1 ... db_name_n /path/to/new_directory
shell> mysqlhotcopy db_name./regex/
```

`mysqlhotcopy` supports the following options:

`--help, -?`
: Display a help message and exit.

`--allowold`
: Don't abort if target already 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 (now 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`.

`--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 clients.

`--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 14.1.5 [LOAD DATA], page 635.

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 14.1.5 [`LOAD DATA`], page 635.

`--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 from the client host. By default, text files are assumed to be on the server host if you connect to `localhost` (which is the default 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. Note that if you use the short option form (`-p`), you *cannot* have a space between the option and the password. If no password is given on the command line, you will be 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 14.5.3 [SHOW], page 701.

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 will 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 5.7.1 [Character sets], page 340.

`--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 5.7.1 [Character sets], page 340.

`--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. Note that if you use the short option form (`-p`), you *cannot* have a space between the option and the password. If no password is given on the command line, you will be 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 20.1.1 [msql2mysql], page 857.

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.

9 MySQL Language Reference

MySQL has a very complex, but intuitive and easy to learn SQL interface. The next several chapters of the manual comprise a language reference. They describe the various commands, types, and functions you will need to know in order to use MySQL efficiently and effectively. These chapters also serve as a reference to all functionality included in MySQL. The material they contain is grouped by topic:

-
  See Chapter 10 [Language Structure], page 491 for information about how to write data values, identifiers, and comments.
- See Chapter 12 [Column types], page 533 for information about the various column types that MySQL supports for storing data in tables.
- See Chapter 13 [Functions], page 555 for a list of the functions and operators that you can use when writing expressions.
- See Chapter 14 [SQL Syntax], page 626 for descriptions of the SQL statements that MySQL supports.

In order to use this material most effectively, you may find it useful to refer to the various indexes.

10 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

10.1 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.

10.1.1 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 will be 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 11.3.7 [Charset-literal], page 514.

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 will cause 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 backslash, 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 13.3.1 [String comparison functions], page 573. 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 C API function `mysql_real_escape_string()` to escape characters. See Section 20.2.3.44 [`mysql_real_escape_string()`], page 898. The Perl DBI interface provides a `quote` method to convert special characters to the proper escape sequences. See Section 20.6 [Perl], page 967.
- 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.

10.1.2 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.

10.1.3 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 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'
```

10.1.4 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
```

10.1.5 `NULL` Values

The `NULL` value means "no data." `NULL` can be written in any lettercase.

Be aware that the `NULL` value is different than values such as `0` for numeric types or the empty string for string types. See Section A.5.3 [Problems with `NULL`], page 1021.

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 14.1.5 [`LOAD DATA`], page 635.

10.2 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. 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 10.6 [Reserved words], page 503. Special characters are those outside the set of alphanumeric characters from the current character set, '_', and '$'.

The 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 1.8.2 [SQL mode], page 40.

As of MySQL 4.1, identifier quote characters can be included within an identifier by quoting 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 5.7.1 [Character sets], page 340.
- 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 *only* of 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 10.2.1 [Identifier qualifiers], page 496).

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.

10.2.1 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.

10.2.2 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 1.8.4 [Extensions to ANSI], page 41.

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 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 will 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.

10.3 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 5.7.1 [Character sets], page 340. User variable names are not case insensitive 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`.

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 will *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` will 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 will report to the client that column one is a string and convert all accesses of `@a` to strings, even though @a is set to a number for the second row. After the `SELECT` statement executes, `@a` will be 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.

10.4 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 intialized from the global variable only for clients that connect after the change. It does not affect the session variable for any client that is already 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 14.5.3.1 [SET OPTION], page 702.

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 [Server options], page 228 and Section 5.2.3 [Server system variables], page 240. A list of the variables that can be set at runtime is given in Section 5.2.3.1 [Dynamic System Variables], page 260.

10.4.1 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 [MyISAM key cache], page 430.

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 will have 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 will 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=2096
```

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 2096 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.

10.5 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 1.8.5.7 [ANSI diff comments], page 49.
- 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 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.
- 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.

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.

10.6 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 10.2 [Legal names], page 495. The server SQL mode is controlled as described in Section 1.8.2 [SQL mode], page 40.

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 by quoting it.

Word	**Word**	**Word**
ADD	ALL	ALTER
ANALYZE	AND	AS
ASC	ASENSITIVE	AUTO_INCREMENT
BDB	BEFORE	BERKELEYDB
BETWEEN	BIGINT	BINARY
BLOB	BOTH	BY
CALL	CASCADE	CASE
CHANGE	CHAR	CHARACTER
CHECK	COLLATE	COLUMN
COLUMNS	CONDITION	CONNECTION
CONSTRAINT	CONTINUE	CREATE
CROSS	CURRENT_DATE	CURRENT_TIME
CURRENT_TIMESTAMP	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	ELSE	ELSEIF
ENCLOSED	ESCAPED	EXISTS
EXIT	EXPLAIN	FALSE
FETCH	FIELDS	FLOAT
FOR	FORCE	FOREIGN
FOUND	FRAC_SECOND	FROM

FULLTEXT	GRANT	GROUP
HAVING	HIGH_PRIORITY	HOUR_MICROSECOND
HOUR_MINUTE	HOUR_SECOND	IF
IGNORE	IN	INDEX
INFILE	INNER	INNODB
INOUT	INSENSITIVE	INSERT
INT	INTEGER	INTERVAL
INTO	IO_THREAD	IS
ITERATE	JOIN	KEY
KEYS	KILL	LEADING
LEAVE	LEFT	LIKE
LIMIT	LINES	LOAD
LOCALTIME	LOCALTIMESTAMP	LOCK
LONG	LONGBLOB	LONGTEXT
LOOP	LOW_PRIORITY	MASTER_SERVER_ID
MATCH	MEDIUMBLOB	MEDIUMINT
MEDIUMTEXT	MIDDLEINT	MINUTE_MICROSECOND
MINUTE_SECOND	MOD	NATURAL
NOT	NO_WRITE_TO_BINLOG	NULL
NUMERIC	ON	OPTIMIZE
OPTION	OPTIONALLY	OR
ORDER	OUT	OUTER
OUTFILE	PRECISION	PRIMARY
PRIVILEGES	PROCEDURE	PURGE
READ	REAL	REFERENCES
REGEXP	RENAME	REPEAT
REPLACE	REQUIRE	RESTRICT
RETURN	REVOKE	RIGHT
RLIKE	SECOND_MICROSECOND	SELECT
SENSITIVE	SEPARATOR	SET
SHOW	SMALLINT	SOME
SONAME	SPATIAL	SPECIFIC
SQL	SQLEXCEPTION	SQLSTATE
SQLWARNING	SQL_BIG_RESULT	SQL_CALC_FOUND_ROWS
SQL_SMALL_RESULT	SQL_TSI_DAY	SQL_TSI_FRAC_SECOND
SQL_TSI_HOUR	SQL_TSI_MINUTE	SQL_TSI_MONTH
SQL_TSI_QUARTER	SQL_TSI_SECOND	SQL_TSI_WEEK
SQL_TSI_YEAR	SSL	STARTING
STRAIGHT_JOIN	STRIPED	TABLE
TABLES	TERMINATED	THEN
TIMESTAMPADD	TIMESTAMPDIFF	TINYBLOB
TINYINT	TINYTEXT	TO
TRAILING	TRUE	UNDO
UNION	UNIQUE	UNLOCK
UNSIGNED	UPDATE	USAGE
USE	USER_RESOURCES	USING
UTC_DATE	UTC_TIME	UTC_TIMESTAMP

VALUES	VARBINARY	VARCHAR
VARCHARACTER	VARYING	WHEN
WHERE	WHILE	WITH
WRITE	XOR	YEAR_MONTH
ZEROFILL		

The following keywords are allowed by MySQL as column/table names. This is because they are very natural names and a lot of people have already used them.

- ACTION
- BIT
- DATE
- ENUM
- NO
- TEXT
- TIME
- TIMESTAMP

11 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 `MySISAM`, `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.

11.1 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, '`c`' = 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**.

Now, 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`'. Now, 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 will need to learn what character sets and collations are available, how to change their defaults, and what the various string operators do with them.

11.2 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   |
+----------+-----------------------------+---------------------+
| 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   |
...
```

The output actually includes another column that is not shown so that the example fits better on the page.

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), `_bin` (binary), or `_uca` (Unicode Collation Algorithm (`http://www.unicode.org/reports/tr10/`)).

11.3 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.

11.3.1 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.

11.3.2 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`.

11.3.3 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.

11.3.4 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.

11.3.5 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. Now, 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` will have 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`.

11.3.6 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 already 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 statement that the client sends. Thus, `SET NAMES 'cp1251'` tells the server "future incoming messages from this client will be 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 will have 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;
```

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 will send 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 will convert 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;
```

11.3.7 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.

11.3.8 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;
```

11.3.9 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)
```

11.3.10 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;
```

11.3.11 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 will be "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 literal's collation has a coercibility of 3.

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('A');
        -> 3
```

See Section 13.8.3 [Information functions], page 611.

11.3.12 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 as 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                          |
+-------------------------------+
```

After an upgrade to MySQL 4.1, 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'
```

The reason this occurs is that usernames are stored using UTF8 (see Section 11.6 [Charset-metadata], page 522). As a result, the `USER()` function and the literal string `'@'` 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 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.

11.3.13 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.

11.4 Operations Affected by Character Set Support

This section describes operations that take character set information into account as of MySQL 4.1.

11.4.1 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()`.

11.4.2 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.

11.4.3 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;
```

11.4.4 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 14.5.3.2 [SHOW CHARACTER SET], page 706.

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 14.5.3.3 [SHOW COLLATION], page 707.

SHOW CREATE DATABASE displays the CREATE DATABASE statement that will create 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 14.5.3.5 [SHOW CREATE DATABASE], page 708.

SHOW CREATE TABLE is similar, but displays the CREATE TABLE statement to create a given table. The column definitions now indicate any character set specifications, and the table options include character set information.

See Section 14.5.3.6 [SHOW CREATE TABLE], page 708.

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 14.5.3.4 [SHOW COLUMNS], page 707.

11.5 Unicode Support

There are two new (as of MySQL version 4.1) 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'` will not work.

The UTF8 character set (transform Unicode representation) is an alternative way to store Unicode data. It is implemented according to RFC2279. The idea of the UTF8 character set is that various Unicode characters fit into 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.

Currently, MySQL UTF8 support does not include four-byte sequences.

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.

11.6 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 will 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()`, and `VERSION()` functions will have the UTF8 character set by default. So will any synonyms, such the `SESSION_USER()` and `SYSTEM_USER()` synonyms for `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 will be in the character_set_system character set by default. When you say SELECT column1 FROM t, the name column1 itself will be 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 11.3.6 [Charset-connection], page 513), 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 will not be 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 will do some automatic conversion for you.

```
SELECT * FROM Table1 WHERE USER() = latin1_column;
```

This will work because the contents of latin1_column are automatically converted to UTF8 before the comparison.

```
INSERT INTO Table1 (latin1_column) SELECT USER();
```

This will work 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.

11.7 Compatibility with Other DBMSs

For SAP DB compatibility these two statements are the same:

```
CREATE TABLE t1 (f1 CHAR(n) UNICODE);
CREATE TABLE t1 (f1 CHAR(n) CHARACTER SET ucs2);
```

11.8 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.

11.9 National Character Set

Before MySQL 4.1, NCHAR and CHAR were synonymous. ANSI defines NCHAR or NATIONAL CHAR as a way to define 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';
```

11.10 Upgrading Character Sets from MySQL 4.0

Now, 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 will have to unlearn this. Henceforth, we will 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 now 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 11.10.1 [Charset-map], page 525 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 11.10.2 [Charset-conversion], page 526.

11.10.1 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

11.10.2 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 now 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;
```

11.11 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    |
| cp1251   | Windows Cyrillic            | cp1251_bulgarian_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   |
| cp1256   | Windows Arabic              | cp1256_general_ci   |
| cp1257   | Windows Baltic              | cp1257_general_ci   |
| binary   | Binary pseudo charset       | binary              |
| geostd8  | GEOSTD8 Georgian            | geostd8_general_ci  |
+----------+-----------------------------+---------------------+
```

11.11.1 Unicode Character Sets

MySQL has two Unicode character sets. You can store texts in about 650 languages using these character sets. We have not added a large number of collations for these two new sets yet, but that will be happening soon. Currently, they have default case-insensitive accent-insensitive collations, plus the binary collation.

Currently, the `ucs2_general_uca` collation has only partial support for the Unicode Collation Algorithm. Some characters are not supported yet.

- `ucs2` (UCS-2 Unicode) collations:
  - `ucs2_bin`
  - `ucs2_general_ci` (default)
  - `ucs2_general_uca`
- `utf8` (UTF-8 Unicode) collations:
  - `utf8_bin`
  - `utf8_general_ci` (default)

11.11.2 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 will 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)

11.11.3 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_ci
  - 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_ci
  - latin2_general_ci (default)
  - latin2_hungarian_ci
- macce (Mac Central European) collations:
  - macce_bin
  - macce_general_ci (default)

11.11.4 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)

11.11.5 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

11.11.6 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)

11.11.7 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)
- `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)

12 Column Types

MySQL supports a number of column types that may be grouped into 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. 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.

In addition, MySQL 4.1 and up supports extensions for handing spatial data. Information about spatial types is provided in Chapter 18 [Spatial extensions in MySQL], page 816.

The column types supported by MySQL follow. Several of the type descriptions use these conventions:

`M` Indicates the maximum display size. The maximum legal display size 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.

Note that if you specify `ZEROFILL` for a column, MySQL automatically adds the `UNSIGNED` attribute to the column.

Warning: You should be aware that when you use subtraction between integer values where one is of type `UNSIGNED`, the result will be unsigned! See Section 13.7 [Cast Functions], page 605.

12.1 Column Type Overview

12.1.1 Overview of Numeric Types

`TINYINT[(M)] [UNSIGNED] [ZEROFILL]`
A very small integer. The signed range is `-128` to `127`. The unsigned range is `0` to `255`.

`BIT`
`BOOL`
`BOOLEAN` These are synonyms for `TINYINT(1)`. The `BOOLEAN` synonym was added in version 4.1.0

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)` and `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 representation.
- '`-`', '`+`', and '`*`' will use `BIGINT` arithmetic when both arguments 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(precision) [UNSIGNED] [ZEROFILL]`
A floating-point number. `precision` 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(precision)` has the same range as the corresponding `FLOAT` and `DOUBLE` types, but the display size and number of decimals are undefined.

As of MySQL Version 3.23, this is a true floating-point value. In earlier MySQL versions, `FLOAT(precision)` always has 2 decimals.

Note that using `FLOAT` might give you some unexpected problems because all calculations in MySQL are done with double precision. See Section A.5.6 [No matching rows], page 1023.

This syntax is provided for ODBC compatibility.

`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(precision) (where precision 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(precision) (where precision 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.

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. 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 Version 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 alias was added in version 4.1.0 for compatibility with other servers.

12.1.2 Overview of Date and Time Types

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. See Section 12.3.1 [DATETIME], page 542.

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. See Section 12.3.1 [DATETIME], page 542.

TIMESTAMP[(M)]
: A timestamp. The range is '1970-01-01 00:00:00' to sometime in 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. See Section 12.3 [Date and time types], page 541.

  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 4 bytes each. From version 4.0.12, the --new option can be used to make the server behave as in version 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! See Section 12.3.1 [DATETIME], page 542.

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. See Section 12.3.2 [TIME], page 546.

YEAR[(2|4)]
: A year in two-digit or four-digit format. The default is four-digit. The allowable values are 1901 to 2155, 0000 in the four-digit year format, and 70 to 69 representing years from 1970 to 2069 in two-digit format. 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 Version 3.22. See Section 12.3.3 [YEAR], page 547.

12.1.3 Overview of String Types

[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 Version 3.23). Trailing spaces are removed when the value is retrieved. CHAR values are sorted and compared in

case-insensitive fashion according to the default character set unless the `BINARY` keyword is given. See Section 14.2.5.1 [Silent column changes], page 681.

From version 4.1.0, 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. Note that this conversion causes the column to become a variable-length column.

`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.

From version 4.1.0 on, the `ASCII` attribute can be specified. It assigns the `latin1` character set to a `CHAR` column.

From version 4.1.1, the `UNICODE` attribute can be specified. It assigns the `ucs2` character set to a `CHAR` column.

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). See Section 12.4.1 [`CHAR`], page 548.

`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 0 to 255 characters (1 to 255 prior to MySQL Version 4.0.2). **Note:** Trailing spaces are removed when the value is stored, which differs from the standard SQL specification. `VARCHAR` values are sorted and compared in case-insensitive fashion unless the `BINARY` keyword is given. See Section 14.2.5.1 [Silent column changes], page 681.

From version 4.1.0 on, 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.

`VARCHAR` is shorthand for `CHARACTER VARYING`. See Section 12.4.1 [`CHAR`], page 548.

`TINYBLOB`
`TINYTEXT`

A `BLOB` or `TEXT` column with a maximum length of 255 (2^8 − 1) characters. See Section 14.2.5.1 [Silent column changes], page 681. See Section 12.4.2 [`BLOB`], page 548.

`BLOB`
`TEXT`

A `BLOB` or `TEXT` column with a maximum length of 65535 (2^16 −1) characters. See Section 14.2.5.1 [Silent column changes], page 681. See Section 12.4.2 [`BLOB`], page 548.

MEDIUMBLOB
MEDIUMTEXT

A BLOB or TEXT column with a maximum length of 16777215 (2^24 − 1) characters. See Section 14.2.5.1 [Silent column changes], page 681. See Section 12.4.2 [BLOB], page 548.

LONGBLOB
LONGTEXT

A BLOB or TEXT column with a maximum length of 4294967295 or 4GB (2^32 − 1) characters. See Section 14.2.5.1 [Silent column changes], page 681. Up to MySQL version 3.23, the client/server protocol and MyISAM tables had a limit of 16MB per communication packet / table row. From version 4.0, the maximum allowed length of LONGBLOB or LONGTEXT columns depends on the configured maximum packet size in the client/server protocol and available memory. See Section 12.4.2 [BLOB], page 548.

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 can have a maximum of 65,535 distinct values. ENUM values are represented internally as integers. See Section 12.4.3 [ENUM], page 550.

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 can have a maximum of 64 members. SET values are represented internally as integers. See Section 12.4.4 [SET], page 551.

12.2 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 an extension to the SQL standard, MySQL also supports the integer types TINYINT, MEDIUMINT, and BIGINT as listed in the following table.

Type	Bytes	Minimum Value	Maximum Value
TINYINT	1	-128	127
SMALLINT	2	-32768	32767
MEDIUMINT	3	-8388608	8388607
INT	4	-2147483648	2147483647
BIGINT	8	-9223372036854775808	9223372036854775807

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. However, the display width does not constrain the range of values that can be stored in the column, or the number of digits that will be 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, as 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 little 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.

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 will be stored for values, and the scale represents the number of digits that will 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 5 digits and 2 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, MySQL actually can store numbers up to `9999.99` in this column. For positive numbers, it 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(p)` is equivalent to `DECIMAL(p,0)`. Similarly, the syntax `DECIMAL` is equivalent to `DECIMAL(p,0)`, where the implementation is allowed to decide the value of `p`. As of MySQL 3.23.6, both of these variant forms of the `DECIMAL`/`NUMERIC` data types are supported. Before 3.23.6, the precision and scale 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.) When a `DECIMAL` or `NUMERIC` 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.

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 8-byte double-precision floating-point values (when the server SQL mode does not include 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.

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`.

Conversions that occur due to clipping are reported as "warnings" for `ALTER TABLE`, `LOAD DATA INFILE`, `UPDATE`, and multiple-row `INSERT` statements.

12.3 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.

MySQL allows you to store certain "not strictly legal" date values, such as `'1999-11-31'`. The reason for this is that we consider date checking to be the responsibility of the application, not the SQL server. To make date checking faster, 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 like `'1999-00-00'` or `'1999-01-00'`. Note that if you store dates like these, you should not expect to get correct results for such as `DATE_SUB()` or `DATE_ADD` that require complete dates.

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 will 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:

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 Connector/ODBC are converted automatically to `NULL` in Connector/ODBC 2.50.12 and above, because ODBC can't handle such values.

12.3.1 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 that they will.)

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 will represent March, 1999, you will find that 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 will be 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 will be interpreted as the year '2010-11-12'. The value '10:45:15' will be 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. 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 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.

12.3.1.1 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.

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 already has 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 NOW().

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 sometime in the year 2037, with a resolution of one second. Values are displayed as numbers.

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 display 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 will not be a legal date and `0` will be stored.
- If you use `ALTER TABLE` to widen a narrow `TIMESTAMP` column, information will be 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.
- 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 functions 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.

12.3.1.2 `TIMESTAMP` Properties as of MySQL 4.1

From MySQL 4.1.0 on, `TIMESTAMP` properties differ from those of earlier MySQL releases:

- `TIMESTAMP` columns are displayed in the same format as `DATETIME` columns.
- Display widths are not supported in the ways described in the preceding section. In other words, you cannot use `TIMESTAMP(2)`, `TIMESTAMP(4)`, and so on.

In addition, if the MySQL server is running in `MAXDB` mode, `TIMESTAMP` is identical with `DATETIME`. That is, if the server is running in `MAXDB` mode at the time that a table is created, any `TIMESTAMP` columns are created as `DATETIME` columns. As a result, such columns use `DATETIME` display format, have the same range of values, and no automatic updating occurs.

MySQL can be run in `MAXDB` mode as of version 4.1.1. To enable this 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;
```

12.3.2 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 33. 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'` will 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.

12.3.3 The YEAR Type

The YEAR type is a 1-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 will be 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.

12.3.4 Y2K Issues and Date Types

MySQL itself is year 2000 (Y2K) safe (see Section 1.2.5 [Year 2000 compliance], page 10), 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 mean. If the heuristics used by MySQL don't produce the correct values, you should provide unambiguous input containing four-digit year values.

ORDER BY sorts two-digit YEAR/DATE/DATETIME types properly.

Note also that some functions like MIN() and MAX() will convert a TIMESTAMP/DATE to a number. This means that a timestamp with a two-digit year will not work properly with these functions. The fix in this case is to convert the TIMESTAMP/DATE to four-digit year format or use something like MIN(DATE_ADD(timestamp,INTERVAL 0 DAYS)).

12.4 String Types

The string types are CHAR, VARCHAR, BLOB, TEXT, ENUM, and SET. This section describes how these types work, their storage requirements, and how to use them in your queries.

12.4.1 The CHAR and VARCHAR Types

The CHAR and VARCHAR types are similar, but differ in the way they are stored and retrieved.

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. You can declare a VARCHAR column to be any length from 0 to 255, just as for CHAR columns. (Before MySQL 4.0.2, the length of VARCHAR may be from 1 to 255.) However, in contrast to CHAR, VARCHAR values are stored using only as many characters as are needed, plus one byte to record the length. Values are not padded; instead, trailing spaces are removed when values are stored. (This space removal differs from the standard SQL specification.) No case 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 you need a column for which trailing spaces are not removed, consider using a BLOB or TEXT type.

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 will be the same in each case, because trailing spaces are removed from CHAR columns upon retrieval.

Values in CHAR and VARCHAR columns are sorted and compared in case-insensitive fashion, unless the BINARY attribute was specified when the table was created. The BINARY attribute means that column values are sorted and compared in case-sensitive fashion using the underlying character code values rather then a lexical ordering. BINARY doesn't affect how the column is stored or retrieved.

From version 4.1.0, column type CHAR BYTE is an alias for CHAR BINARY. This is a compatibility feature.

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.

MySQL may silently change the type of a CHAR or VARCHAR column at table creation time. See Section 14.2.5.1 [Silent column changes], page 681.

12.4.2 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 12.5 [Storage requirements], page 552.

The four TEXT types TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT correspond to the four BLOB types and have the same maximum lengths and storage requirements. The only difference between BLOB and TEXT types is that sorting and comparison is performed in case-sensitive fashion for BLOB values and case-insensitive fashion for TEXT values. In other words, a TEXT is a case-insensitive BLOB. No case 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.

In most respects, you can regard a TEXT column as a VARCHAR column that can be as big as you like. Similarly, you can regard a BLOB column as a VARCHAR BINARY column. The differences are:

- You can have indexes on BLOB and TEXT columns only as of MySQL 3.23.2. 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.
- There is no trailing-space removal for BLOB and TEXT columns when values are stored or retrieved. This differs from CHAR columns (trailing spaces are removed when values are retrieved) and from VARCHAR columns (trailing spaces are removed when values are stored).
- BLOB and TEXT columns cannot have DEFAULT values.

From version 4.1.0, LONG and LONG VARCHAR map to the MEDIUMTEXT data type. This is a compatibility feature.

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:

- If you want to use GROUP BY or ORDER BY on a BLOB or TEXT column, you must convert the column value into a fixed-length object. The standard way to do this is with the SUBSTRING function. For example:

  ```
  mysql> SELECT comment FROM tbl_name,SUBSTRING(comment,20) AS substr
      ->                 ORDER BY substr;
  ```

  If you don't do this, 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], page 240.

  You can group on an expression involving BLOB or TEXT values by specifying the column position or by using an alias:

  ```
  mysql> SELECT id,SUBSTRING(blob_col,1,100)
      -> FROM tbl_name GROUP BY 2;
  mysql> SELECT id,SUBSTRING(blob_col,1,100) AS b
      -> FROM tbl_name GROUP BY b;
  ```

- The maximum size of a BLOB or TEXT object is determined by its type, but the largest value you can actually 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, mysql and mysqldump each allow you to change the client-side max_allowed_packet value. See Section 7.5.2 [Server parameters], page 437, Section 8.3 [mysql], page 456, and Section 8.8 [mysqldump], page 477.

Note that 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.

12.4.3 The ENUM Type

An ENUM is a string object with a value normally 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 values when the table is created.

Lettercase is irrelevant when you assign values to an ENUM column. However, values retrieved from the column later are displayed using the lettercase that was used in the column definition.

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 will 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 get all possible values for an ENUM column, you should use SHOW COLUMNS FROM tbl_name LIKE enum_col_name and parse the ENUM definition in the second column of the output.

12.4.4 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 3.23.51 trailing spaces are automatically deleted from SET values when the table is created.

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 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 will appear 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 will appear as 'a,d' when retrieved.

If you set a SET column to an unsupported value, the value will be ignored.

SET values are sorted numerically. NULL values sort before non-NULL SET values.

Normally, you perform a SELECT on a SET column using the LIKE operator or the FIND_IN_SET() function:

```
mysql> SELECT * FROM tbl_name WHERE set_col LIKE '%value%';
mysql> SELECT * FROM tbl_name WHERE FIND_IN_SET('value',set_col)>0;
```

But the following will also work:

```
mysql> SELECT * FROM tbl_name WHERE set_col = 'val1,val2';
mysql> SELECT * FROM tbl_name WHERE set_col & 1;
```

The first of these statements looks for an exact match. The second looks for values containing the first set member.

If you want to get all possible values for a SET column, you should use SHOW COLUMNS FROM tbl_name LIKE set_col_name and parse the SET definition in the second column of the output.

12.5 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 65534 bytes. Each BLOB and TEXT column accounts for only 5-9 bytes toward this size.

If a table includes any variable-length column types, the record format will also be variable-length. Note that 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 14.2.5.1 [Silent column changes], page 681.

Storage Requirements for Numeric Types

Column Type	**Storage Required**
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes

INT	4 bytes
INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(X)	4 if 0 <= X <= 24, 8 if 25 <= X <= 53
FLOAT	4 bytes
DOUBLE	8 bytes
DOUBLE PRECISION	8 bytes
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)

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
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 which 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 characters. 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.

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 12.4.2 [BLOB], page 548.

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 65535 values. See Section 12.4.3 [ENUM], page 550.

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 12.4.4 [SET], page 551.

12.6 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 will be 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. 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.

12.7 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(NUM)`	`CHAR(NUM) BINARY`
`CHAR VARYING(NUM)`	`VARCHAR(NUM)`
`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(NUM)`	`VARCHAR(NUM) BINARY`

Column type mapping occurs at table creation time. 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.

13 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 will become reserved words. See Section 5.2.2 [Server SQL mode], page 237.

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
```

13.1 Operators

13.1.1 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
```

13.1.2 Comparison 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.

MySQL performs comparisons 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 the is not done for arguments in `IN()`! To be safe, always use complete datetime/date/time string 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).

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 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 14.5.3.1 [SET OPTION], page 702.

- 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 14.1.8.3 [ANY IN SOME subqueries], page 654.

`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
```

Note that a comparison of `NULL` values using `=` will always be 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.

13.1.3 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. Evaluates to `1` if any operand is non-zero, to `NULL` if any operand is `NULL`, otherwise `0` is returned.

```
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.

13.1.4 Case-sensitivity 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 case sensitive even if the column isn't defined as `BINARY` or `BLOB`.

```
mysql> SELECT 'a' = 'A';
        -> 1
mysql> SELECT BINARY 'a' = 'A';
        -> 0
```

BINARY was added in MySQL 3.23.0. As of MySQL 4.0.2, BINARY str is a shorthand for CAST(str AS BINARY). See Section 13.7 [Cast Functions], page 605.

Note that in some contexts, if you cast an indexed column to BINARY, MySQL will not be able to use the index efficiently.

If you want to compare a BLOB value in case-insensitive fashion, you can do so as follows:

- Before MySQL 4.1.1, use the UPPER() function to convert the BLOB value to uppercase before performing the comparison:

  ```
  SELECT 'A' LIKE UPPER(blob_col) FROM tbl_name;
  ```

  If the comparison value is lowercase, convert the BLOB value using LOWER() instead.
- For MySQL 4.1.1 and up, BLOB columns have a character set of binary, which has no concept of lettercase. To perform a case-insensitive comparison, use the CONVERT() function to convert the BLOB 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.

13.2 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.

13.3 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 [Server parameters], page 437.

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`.

```
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. A numeric argument is converted to its equivalent 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 `bits`, you get an `on` string and for every reset bit you get an `off` string. Each string is separated by `separator` (default '`,`'), and only `number_of_bits` (default 64) of `bits` is used.

```
mysql> SELECT EXPORT_SET(5,'Y','N',',',4)
        -> Y,N,Y,N
```

`FIELD(str,str1,str2,str3,...)`

Returns the index of `str` in the `str1`, `str2`, `str3`, `...` list. Returns `0` if `str` is not found. `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 will 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`.

```
mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
        -> 'QuWhattic'
```

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 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 * 256)
+ (2nd byte code * 256^2)
+ (3rd byte code * 256^3) ...
```

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)` (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)`
: 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 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(string). 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.

13.3.1 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`.

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).

Note: Currently `LIKE` is not multi-byte character safe. Comparison is done character by character.

`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 F [Regexp], page 1212. 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. 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.

13.4 Numeric Functions

13.4.1 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 argument is an unsigned integer, and the other argument is also an integer, the result will be an unsigned integer. See Section 13.7 [Cast Functions], page 605.

`+` 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 will be calculated with `BIGINT` arithmetic only if performed in a context where its result is 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.

13.4.2 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 will be 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 (like the `%` operator in C). 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.

`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 will be 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. If `D` is negative, the integer part of the number is zeroed out.

```
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 will have no decimal point or fractional part. If `D` is negative, the integer part of the number is zeroed out.

```
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`.

13.5 Date and Time Functions

This section describes the functions that can be used to manipulate temporal values. See Section 12.3 [Date and time types], page 541 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 will select records with dates that lie in the future.

Functions that expect date values usually will accept datetime values and ignore the time part. Functions that expect time values usually will 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 will always produce the same result. This principle also applies to `CURDATE()`, `CURTIME()`, `UTC_DATE()`, `UTC_TIME()`, `UTC_TIMESTAMP()`, and to any of their synonyms.

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 date 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.

`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|TIMESTAMP, '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(TIMESTAMP,'USA')	'%Y-%m-%d-%H.%i.%s'
GET_FORMAT(TIMESTAMP,'JIS')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(TIMESTAMP,'ISO')	'%Y-%m-%d %H:%i:%s'
GET_FORMAT(TIMESTAMP,'EUR')	'%Y-%m-%d-%H.%i.%s'
GET_FORMAT(TIMESTAMP,'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.

```
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 14.5.3.1 [SET OPTION], page 702.

HOUR(time)

Returns the hour for time. The range of the return value will be 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 will be `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 `YYYYMMDDHHMMSS` 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`, and returns a `DATETIME` value. 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
```

`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 date 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 12.3 [Date and time types], page 541. 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 13.7 [Cast Functions], page 605.

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 52. 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], page 240.

The following table describes how the mode argument works:

Value	Meaning
0	Week starts on Sunday; return value range is 0 to 53; week 1 is the first week that starts in this year

`1`	Week starts on Monday; return value range is `0` to `53`; week 1 is the first week that has more than three days in this year
`2`	Week starts on Sunday; return value range is `1` to `53`; week 1 is the first week that starts in this year
`3`	Week starts on Monday; return value range is `1` to `53`; week 1 is the first week that has more than three days in this year
`4`	Week starts on Sunday; return value range is `0` to `53`; week 1 is the first week that has more than three days in this year
`5`	Week starts on Monday; return value range is `0` to `53`; week 1 is the first week that starts in this year
`6`	Week starts on Sunday; return value range is `1` to `53`; week 1 is the first week that has more than three days in this year
`7`	Week starts on Monday; return value range is `1` to `53`; week 1 is the first week that starts in this year

The `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 `2`, `3`, `6`, 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`.

```
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.

13.6 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 will be 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 already has a `FULLTEXT` index could be significantly slower.

Constraints on full-text searching are listed in Section 13.6.3 [Fulltext Restrictions], page 603.

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()`. The search is performed in case-insensitive fashion. 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.

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 13.6.1 [Fulltext Boolean], page 600 and Section 13.6.2 [Fulltext Query Expansion], page 602.

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 characters consisting of letters, digits, '’', or '_'. Some words are ignored in full-text searches:

- Any word that is too short is ignored. The default minimum length of words that will be 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 13.6.4 [Fulltext Fine-tuning], page 603.

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 will most likely find 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.

13.6.1 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 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 will be 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 will be rated lower than others, but will not be 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, not prepended.

`"`
: A phrase that is enclosed within double quote ('`"`') characters matches only rows that contain the phrase *literally, as it was typed.*

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.

13.6.2 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 will find 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” will find only "Maigret and the Reluctant Witnesses" without query expansion. A search with query expansion will find 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.

13.6.3 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.

13.6.4 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 modfications. See Section 2.3 [Installing source], page 98.

Note that full-text search was carefully tuned for the best searching effectiveness. Modifying the default behavior will, in most cases, make 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], page 240. 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], page 240. 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 running, but you must have the SUPER privilege to do so. No index rebuilding is necessary. Section 5.2.3 [Server system variables], page 240 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 will return 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.

13.6.5 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 (dependent on the language of the dataset).
- Generic user-suppliable UDF preparser.
- Make the model more flexible (by adding some adjustable parameters to `FULLTEXT` in `CREATE TABLE` and `ALTER TABLE` statements).

13.7 Cast Functions

`CAST(expr AS type)`
`CONVERT(expr,type)`
`CONVERT(expr USING transcoding_name)`

The `CAST()` and `CONVERT()` functions may 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]`

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 13.5 [Date and time functions], page 582.

To cast a string to a numeric value, 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 numerical 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 will not be able to make efficient use of the `BIGINT UNSIGNED` column type.

13.8 Other Functions

13.8.1 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
```

13.8.2 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 5.5.8 [Secure connections], page 311.

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 will return 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 5.5.8 [Secure connections], page 311.

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 will be 127.

The string length for the result will be new_len = orig_len + (8-(orig_len % 8))+1.

The DES key file has the following format:

```
key_num des_key_str
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 will be 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 command. 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.)

```
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 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."

`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 5.4.9 [Password hashing], page 295.

`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()`.

13.8.3 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('abc');
        -> 3
mysql> SELECT COERCIBILITY(USER());
        -> 2
```

The return values have the following meanings:

Coercibility	**Meaning**
`0`	Explicit collation
`1`	No collation
`2`	Implicit collation
`3`	Coercible

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.

`DATABASE()`

Returns the default (current) database name.

```
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` will return 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 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 C API function `mysql_insert_id()` can also be used to get the value. See Section 20.2.3.32 [`mysql_insert_id()`], page 886.

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`.

`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.

```
mysql> SELECT VERSION();
        -> '4.1.2-alpha-log'
```

Note that if your version string ends with `-log` this means that logging is enabled.

13.8.4 Miscellaneous Functions

`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 will have 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 already 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'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 thread is not running, this function will block and wait until the slave is started and goes to or past the specified position. If the slave is already 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 will not exist if it was never obtained by a call to GET_LOCK() or if it already has been released.

The DO statement is convenient to use with RELEASE_LOCK(). See Section 14.1.2 [DO], page 628.

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.

13.9 Functions and Modifiers for Use with GROUP BY Clauses

13.9.1 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(expr)

Returns the average value 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 values from a group. The full syntax is as follows:

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
             [ORDER BY {unsigned_integer | col_name | expr}
                 [ASC | DESC] [,col ...]]
             [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.

Note: There are still some small limitations with `GROUP_CONCAT()` when it comes to using `DISTINCT` together with `ORDER BY` and using `BLOB` values. See Section 1.8.7.3 [Open bugs], page 53.

`GROUP_CONCAT()` was added in MySQL 4.1.

`MIN(expr)`
`MAX(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 [MySQL indexes], page 427.

```
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 standard deviation of `expr` (the square root of `VARIANCE()`). This is an extension to standard SQL. The `STDDEV()` form of this function is provided for Oracle compatibility.

`SUM(expr)`

Returns the sum of `expr`. Note that if the return set has no rows, it returns `NULL`!

`VARIANCE(expr)`

Returns the standard variance of `expr` (considering rows as the whole population, not as a sample; so it has the number of rows as denominator). This is an extension to standard SQL, available only in MySQL 4.1 or later.

13.9.2 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.

13.9.3 `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 will 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 [example-Maximum-column-group-row], page 199.

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();
```

14 SQL Statement Syntax

This chapter describes the syntax for the SQL statements supported in MySQL.

14.1 Data Manipulation Statements

14.1.1 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 using `TRUNCATE TABLE`. See Section 14.1.9 [TRUNCATE], page 661.

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 will be 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 16.7.3 [InnoDB auto-increment column], page 771.

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 [example-AUTO_INCREMENT], page 203.

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 7.2.13 [Delete speed], page 419.

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 14.5.2.5 [OPTIMIZE TABLE], page 699 and Section 5.6.2.10 [Optimisation], page 333.

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 14.1.7.1 [JOIN], page 648.

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 do the above change in 4.0 is because we didn't want to break any old applications in 4.0 using the old syntax.

14.1.2 `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.

14.1.3 `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 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 14.1.7 [SELECT], page 643.

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.

14.1.4 INSERT Syntax

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    VALUES ({expr | DEFAULT},...),(...),...
    [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

Or:

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] tbl_name
    SET col_name={expr | DEFAULT}, ...
    [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

Or:

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    SELECT ...
```

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 14.1.4.1 [INSERT SELECT], page 633.

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:

- Any column not explicitly given a value is set to its default 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 14.2.5 [CREATE TABLE], page 670.

  MySQL always has a default value for all columns. This is something that is imposed on MySQL to be able to work with both transactional and non-transactional tables.

Our view is that column content checking should be done in the application and not in the database server.

Note: If you want INSERT statements to generate an error unless you explicitly specify values for all columns that require a non-NULL value, you can configure MySQL using the DONT_USE_DEFAULT_FIELDS compiler option. This behavior is available only if you compile MySQL from source. See Section 2.3.2 [configure options], page 101.

- 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.
- 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();
  ```
- 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 already 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);
  ```

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 14.1.4.2 [INSERT DELAYED], page 633.
- 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 14.1.4.2 [INSERT DELAYED], page 633. Note that LOW_PRIORITY should normally not be used with MyISAM tables because doing so disables concurrent inserts. See Section 15.1 [MyISAM], page 738.
- If you specify the IGNORE keyword in an INSERT with many rows, any rows that duplicate an existing UNIQUE index or PRIMARY KEY value in the table are ignored and are not inserted. If you do not specify IGNORE, the insert is aborted if there is any row that duplicates an existing key value. 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 already 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;
```

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 13.8.3 [Information functions], page 611 and Section 20.2.3.32 [`mysql_insert_id()`], page 886.

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 20.2.3.30 [`mysql_info()`], page 885.

14.1.4.1 `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 will not allow concurrent inserts during `INSERT ... SELECT`.

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.

14.1.4.2 `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 15.1 [MyISAM], page 738.
- 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 forcefully (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 already exists.
- The thread checks whether the handler has acquired a DELAYED lock already; 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 will wait 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 already 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 will be killed if you execute a FLUSH TABLES statement or kill it with KILL thread_id. However, before exiting, it will first store all queued rows into the table. During this time it will not accept any new INSERT statements from another thread. If you execute an INSERT DELAYED statement after this, a new handler thread will be created.

  Note that this means that INSERT DELAYED statements have higher priority than normal INSERT statements if there is an INSERT DELAYED handler already running! Other update statements will 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!

14.1.5 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 7.2.11 [Insert speed], page 417.

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 [mysqlimport], page 484.

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.
- 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.

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 5.4.3 [Privileges provided], page 279.

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 will not work. See Section 5.3.4 [LOAD DATA LOCAL], page 274.

If you need LOAD DATA to read from a pipe, you can use the following technique:

```
mkfifo /mysql/db/x/x
chmod 666 /mysql/db/x/x
cat < /dev/tcp/10.1.1.12/4711 > /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.

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;
```

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 14.1.6 [REPLACE], page 642.

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 7.2.11 [Insert speed], page 417.

`LOAD DATA INFILE` is the complement of `SELECT ... INTO OUTFILE`. See Section 14.1.7 [SELECT], page 643. 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. If a line doesn't include the prefix, the entire line is skipped.

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` will 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 will have 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/linefeed 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 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 will be 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 10.1 [Literals], page 491.

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 will 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 will cause 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 few fields, the table columns for which no input field is present are set to their default values. Default value assignment is described in Section 14.2.5 [CREATE TABLE], page 670.

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 12.3 [Date and time types], page 541.

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!

If an input row has too many fields, the extra fields are ignored and the number of warnings is incremented.

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()` C API function. See Section 20.2.3.30 [mysql_info()], page 885.

Warnings occur under the same circumstances as when values are inserted via the `INSERT` statement (see Section 14.1.4 [INSERT], page 630), 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 14.5.3.20 [SHOW WARNINGS], page 719.

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.

14.1.6 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 14.1.4 [INSERT], page 630.

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 old row and use them in the new row. It appeared that you could do this in some old MySQL versions, but that was a bug that has been corrected.

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.

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:
   1. Delete from the table the conflicting row that has the duplicate key value
   2. Try again to insert the new row into the table

14.1.7 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 14.1.7.2 [UNION], page 650 and Section 14.1.8 [Subqueries], page 652.

- 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 14.1.7.1 [JOIN], page 648.
- where_definition indicates any conditions that selected rows must satisfy.

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 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 alias], page 1022.
- 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 14.1.7.1 [JOIN], page 648. 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 14.1.7.1 [JOIN], page 648.

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 10.2 [Legal names], page 495 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:

```
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 13 [Functions], page 555.
- 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 13.9 [Group by functions and modifiers], page 618.
- As of MySQL 4.1.1, GROUP BY allows a WITH ROLLUP modifier. See Section 13.9.2 [GROUP BY Modifiers], page 621.
- The HAVING clause can refer to any column or alias named in a select_expr. It is applied nearly last, just before items are sent to the client, with no optimization. (LIMIT is applied after HAVING.)
- 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 form of SELECT. The file cannot already 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 14.1.5 [LOAD DATA], page 635.

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 22.3.1 [procedure analyse], page 997.
- 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.

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 will give 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 will run even if there is already 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], page 402. STRAIGHT_JOIN also can be used in the table_references list. See Section 14.1.7.1 [JOIN], page 648.
- SQL_BIG_RESULT can be used with GROUP BY or DISTINCT to tell the optimizer that the result set will have many rows. In this case, MySQL will directly use disk-based temporary tables if needed. MySQL will also, in this case, prefer 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 will be 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 13.8.3 [Information functions], page 611.

  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 7.2.10 [LIMIT optimisation], page 416.
- 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 5.10 [Query Cache], page 358.
- SQL_NO_CACHE tells MySQL not to store the query result in the query cache. See Section 5.10 [Query Cache], page 358. For a query that uses UNION or subqueries, this option takes effect to be used in any SELECT of the query.

14.1.7.1 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 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;
  ```

- 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 7.2.8 [LEFT JOIN optimisation], page 414.
- 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 will produce a Cartesian product between the specified tables (that is, each and every row in the first table will be joined onto 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 will only be 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 will be 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 7.2.8 [LEFT JOIN optimisation], page 414.

14.1.7.2 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 will be unique, as if you had done a DISTINCT for the total result set. If you specify ALL, you will 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.

Note: You cannot mix UNION ALL and UNION DISTINCT in the same query yet. If you use ALL for one UNION then it is used for all of them.

If you want to use an ORDER BY to sort the entire UNION result, you should use parentheses:

```
(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)
ORDER BY a;
```

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    |
+---------------+
```

14.1.8 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 earlier MySQL versions, it was necessary to work around or avoid the use of subqueries, but people starting to write code now will find that subqueries are a very useful part of the MySQL toolkit.

For MySQL versions prior to 4.1, most subqueries can be successfully rewritten using joins and other methods. See Section 14.1.8.11 [Rewriting subqueries], page 660.

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)));
```

14.1.8.1 The Subquery as Scalar Operand

In its simplest form (the *scalar* subquery as opposed to the *row* or *table* subqueries that are discussed later), a subquery is a simple operand. Thus, you can use it wherever a 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);
SELECT (SELECT s2 FROM t1);
```

The subquery in this `SELECT` 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`, because if the table is empty, as in the example, the value of the subquery will be `NULL`. There are few restrictions.

- 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, hints, `UNION` constructs, comments, functions, and so on.

So, when you see examples in the following sections that contain the rather spartan construct `(SELECT column1 FROM t1)`, imagine that your own code will contain 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 will be `2` because there is a row in `t2` containing a column `s1` that has a value of `2`.

The subquery can be part of an expression. If it is an operand for a function, don't forget the parentheses. For example:

```
SELECT UPPER((SELECT s1 FROM t1)) FROM t2;
```

14.1.8.2 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:

```
=  >  <  >=  <=  <>
```

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:

```
SELECT * FROM t1
       WHERE 2 = (SELECT COUNT(column1) FROM t1);
```

14.1.8.3 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 rows 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);
```

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. Using `<> SOME` instead helps ensure that everyone understands the true meaning of the query.

14.1.8.4 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 rows 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.

14.1.8.5 Correlated Subqueries

A *correlated subquery* is a subquery that contains a reference to a column 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 7 <> 5), 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.

14.1.8.6 `EXISTS` and `NOT EXISTS`

If a subquery returns any values 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 will contain 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 which is not in `Stores`?" But it's easier to say that a nested `NOT EXISTS` answers the question "is x TRUE for all y?"

14.1.8.7 Row Subqueries

The discussion to this point has been of *column (or scalar) subqueries*: subqueries that return a single column value. A *row subquery* is a subquery variant that returns a single row value — 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);
```

14.1.8.8 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 will provide 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.

At the moment, subqueries in the FROM clause cannot be correlated subqueries.

Subquery in the FROM clause will be executed (that is, derived temporary tables will be built) even for the EXPLAIN statement, because upper level queries need information about all tables during optimization phase.

14.1.8.9 Subquery Errors

There are some new error returns that apply only to subqueries. This section groups them together because reviewing them will help 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 will not work, but 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 1240 (ER_CARDINALITY_COL)
  SQLSTATE = 21000
  Message = "Operand should contain 1 column(s)"
  ```

  This error will occur 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 14.1.8.7 [Row subqueries], page 656. But in other contexts, the subquery must be a scalar operand.

- Incorrect number of rows from subquery:

  ```
  ERROR 1241 (ER_SUBSELECT_NO_1_ROW)
  SQLSTATE = 21000
  Message = "Subquery returns more than 1 row"
  ```

  This error will occur 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 will occur 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.

Usually, failure of a subquery causes the entire statement to fail.

14.1.8.10 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;
```

- 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`.

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. Don't worry too much about transforming to joins except for compatibility with older versions of MySQL before 4.1 that do not support subqueries.

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 will describe 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.

14.1.8.11 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 will 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.

14.1.9 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`, `TRUNCATE TABLE` is mapped to `DELETE`, so there is no difference. 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 will 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`, 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 14.1.1 [DELETE], page 626.

`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.

14.1.10 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 will be 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 will not abort even if duplicate-key errors occur during the update. Rows for which conflicts occur are not updated.

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 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 C API function `mysql_info()` 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 needed 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 will fail and roll 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.

14.2 Data Definition Statements

14.2.1 ALTER DATABASE Syntax

```
ALTER DATABASE 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 11 [Charset], page 507.

ALTER DATABASE was added in MySQL 4.1.1.

14.2.2 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 14.2.5 [CREATE TABLE], page 670.

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 14.2.5.1 [Silent column changes], page 681. For example, if you try to change a `VARCHAR` column to `CHAR`, MySQL will still use `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.
- 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 14.2.9 [RENAME TABLE], page 683.)
- 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 14.2.5 [CREATE TABLE], page 670.
- 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 14.2.5 [`CREATE TABLE`], page 670.
- `DROP INDEX` removes an index. This is a MySQL extension to standard SQL. See Section 14.2.7 [`DROP INDEX`], page 682.
- 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 will 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.
- The `FOREIGN KEY` and `REFERENCES` clauses are supported by the `InnoDB` storage engine, which implements `ADD [CONSTRAINT [symbol]] FOREIGN KEY (...) REFERENCES ... (...)`. See Section 16.7.4 [`InnoDB` foreign key constraints], page 772. For other storage engines, the clauses are parsed but ignored. The `CHECK` clause is parsed but ignored for all storage engines. See Section 14.2.5 [`CREATE TABLE`], page 670. 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 1.8.5 [Differences from ANSI], page 44.
- 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 16.7.4 [InnoDB foreign key constraints], page 772.

- 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 11.10 [Charset-upgrading], page 524.

  Warning: The preceding operation will convert 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 but 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 16.7.6 [Multiple tablespaces], page 776.
- 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 20.2.3.30 [mysql_info()], page 885.

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 14.5.3.1 [SET OPTION], page 702.

With `MyISAM` tables, if you don't change the `AUTO_INCREMENT` column, the sequence number will not be 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 [ALTER TABLE problems], page 1027.

14.2.3 CREATE DATABASE Syntax

```
CREATE DATABASE [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 10.2 [Legal names], page 495. An error occurs if the database already 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 11 [Charset], page 507.

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).

You can also use the mysqladmin program to create databases. See Section 8.4 [mysqladmin], page 466.

14.2.4 CREATE INDEX Syntax

```
CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name [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 14.2.2 [ALTER TABLE], page 663. 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 14.2.5 [CREATE TABLE], page 670. 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.

Note that 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.

For more information about how MySQL uses indexes, see Section 7.4.5 [MySQL indexes], page 427.

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 13.6 [Fulltext Search], page 597.

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], page 816.

14.2.5 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] [[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
  | TEXT
  | MEDIUMTEXT
  | LONGTEXT
  | 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]
               [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} = {BDB|HEAP|ISAM|InnoDB|MERGE|MRG_MYISAM|MYISAM}
  | 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 }
  | 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 10.2 [Legal names], page 495. By default, the table is created in the current database. An error occurs if the table already 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 already 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 15 [Table types], page 737.

For general information on the properties of the various column types, see Chapter 12 [Column types], page 533. For information about spatial column types, see Chapter 18 [Spatial extensions in MySQL], page 816.

- 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 20.2.3.32 [mysql_insert_id()], page 886.

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 [Server options], page 228.

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 will work 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 [example-AUTO_INCREMENT], page 203.

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 11 [Charset], page 507.

  ```
  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. 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 will contain 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.

- A 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.

  If no DEFAULT value is specified for a column, MySQL automatically assigns one, as follows.

  If the column can take NULL as a value, the default value is NULL.

  If the column is declared as NOT NULL, the default value depends on the column type:

  - 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 12.3 [Date and time types], page 541.
  - 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.
- 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. This is a compatibility feature.
- 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 indexed columns allow 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 will declare 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 will mark only 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 14.5.3.7 [Show database info], page 708.
- 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, INDEX USING BTREE (id))
    ENGINE = MEMORY;
```

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 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 will use the available type.

- 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 [Indexes], page 426.

  The MyISAM and (as of MySQL 4.0.14) InnoDB table types 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 using only the initial number of bytes indicated by the max_sort_length system variable. See Section 12.4.2 [BLOB], page 548.
- 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 13.6 [Fulltext Search], page 597 for details of operation.

- In MySQL 4.1 or later, you can create special 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], page 816.
- In MySQL 3.23.44 or later, InnoDB tables support checking of foreign key constraints. See Chapter 16 [InnoDB], page 758. 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 16.7.4 [InnoDB foreign key constraints], page 772.

  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 and ignored for all storage engines. See Section 1.8.5.5 [ANSI diff Foreign Keys], page 48.
- 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 will be 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**
BDB	Transaction-safe tables with page locking. See Section 15.4 [BDB], page 751.
BerkeleyDB	An alias for BDB.
HEAP	The data for this table is only stored in memory. See Section 15.3 [HEAP], page 749.
ISAM	The original MySQL storage engine. See Section 15.5 [ISAM], page 756.
InnoDB	Transaction-safe tables with row locking. See Chapter 16 [InnoDB], page 758.

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 15.2 [MERGE], page 746.

MRG_MyISAM — An alias for MERGE.

MyISAM — The binary portable storage engine that is the improved replacement for ISAM. See Section 15.1 [MyISAM], page 738.

See Chapter 15 [Table types], page 737.

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.

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 will be. If you don't specify either option, the maximum size for a table will be 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], page 240.

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 will be 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 will usually only take two bytes (including the pointer to the row). Compare this to the ordinary case where the following keys will take `storage_size_for_key + pointer_size` (where the pointer size is usually 4). Conversely, you will get a big benefit from prefix compression only if you have many numbers that are the same. If all keys are totally different, you will 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 will be 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 15.1.3 [MyISAM table formats], page 742.

RAID_TYPE
: 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. For now, 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 will create RAID_CHUNKS subdirectories named '00', '01', '02', ... '09', '0a', '0b', ... in the database directory. In each of these directories, MyISAM will create 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 15.2 [MERGE], page 746.

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 15.2 [MERGE], page 746.

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-symlink option. Your operating system must also have a working, thread-safe realpath() call. See Section 7.6.1.2 [Symbolic links to tables], page 445.

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 will create new column 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` will 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 a 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.

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 will not allow concurrent inserts during `CREATE TABLE ... SELECT`.

14.2.5.1 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 15 [Table types], page 737.
- From MySQL 4.1.0 on, 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. This is a compatibility feature.
- 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.
- 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 values when the table is created.
- MySQL maps certain column types used by other SQL database vendors to MySQL types. See Section 12.7 [Other-vendor column types], page 554.
- 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 engine will use 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 15.1.3.3 [Compressed format], page 744.

14.2.6 DROP DATABASE Syntax

```
DROP DATABASE [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.

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.
- 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], page 466.

14.2.7 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 14.2.2 [ALTER TABLE], page 663. DROP INDEX doesn't do anything prior to MySQL 3.22.

14.2.8 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 14.5.3.20 [SHOW WARNINGS], page 719.

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 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.

14.2.9 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 will do a reverse rename for all renamed tables to get everything back to the original state.

14.3 MySQL Utility Statements

14.3.1 DESCRIBE Syntax (Get Information About Columns)

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

`DESCRIBE` provides information about a table's columns. It is a shortcut for `SHOW COLUMNS FROM`. See Section 14.5.3.7 [Show database info], page 708.

`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.

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 14.2.5.1 [Silent column changes], page 681.

The DESCRIBE statement is provided for Oracle compatibility.

The SHOW CREATE TABLE and SHOW TABLE STATUS statements also provide information about tables. See Section 14.5.3 [SHOW], page 701.

14.3.2 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.

14.4 MySQL Transactional and Locking Statements

14.4.1 START TRANSACTION, COMMIT, and ROLLBACK Syntax

By default, MySQL runs 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.

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 will be 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 5.8.4 [Binary log], page 347.

You can change the isolation level for transactions with SET TRANSACTION ISOLATION LEVEL. See Section 14.4.6 [SET TRANSACTION], page 688.

14.4.2 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.

14.4.3 Statements That Cause an Implicit Commit

The following statements 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.

14.4.4 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 already 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.

14.4.5 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: LOCK TABLES is not transaction-safe and implicitly commits any active transactions before attempting to lock the tables.

As of MySQL 4.0.2, to use LOCK TABLES you must have the global 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 read from or write to the table. Other threads are blocked.

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.

When you use `LOCK TABLES`, you must lock all tables that you are going to use in your queries. If you are using a table multiple times in a query (with aliases), you must get a lock for each alias. While the locks obtained with a `LOCK TABLES` statement are in effect, you cannot access any tables that were not locked by the statement.

If your queries refer to a table using an alias, then you must lock the table using that same alias. It will 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 will 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 will eventually be a time when no threads will 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 with this schema:

If you are using a `LOW_PRIORITY WRITE` lock for a table, it means only that MySQL will wait 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 will 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 14.5.4.3 [KILL], page 724.

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 tables, it's much faster to lock the tables you are going to use. The downside is that no thread can update a READ-locked table (including the one holding the lock) and no thread can read a WRITE-locked table other than the one holding the lock.

  The reason some operations are faster under LOCK TABLES is that MySQL will not flush the key cache for the locked tables until UNLOCK TABLES is called. Normally, the key cache is flushed after each SQL statement. Locking MyISAM tables speeds up inserting, updating, or deleting on them.
- 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 1.8.5.3 [ANSI diff Transactions], page 45.

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 13.8.4 [Miscellaneous functions], page 616.

See Section 7.3.1 [Internal locking], page 422, 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 14.5.4.2 [FLUSH], page 722. 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 [ALTER TABLE problems], page 1027.

14.4.6 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 16.11.2 [InnoDB transaction isolation], page 783. 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 [Server options], page 228.

14.5 Database Administration Statements

14.5.1 Account Management Statements

14.5.1.1 DROP USER Syntax

```
DROP USER user
```

The DROP USER statement deletes a MySQL account that doesn't have any privileges. It serves to remove the account record from the user table. The 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.

To remove a MySQL user 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 14.5.3.10 [SHOW GRANTS], page 710.
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 14.5.1.2 [GRANT], page 689.
3. Delete the account by using DROP USER to remove the user table record.

The DROP USER statement was added in MySQL 4.1.1. Before 4.1.1, 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;
```

14.5.1.2 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]]

    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 [MySQL Database Administration], page 218, which you should consult for additional details.

Privileges can be granted at four 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 tables 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.

To make it easy to revoke all privileges, MySQL 4.1.2 has added the following syntax, which drops all 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 FROM user [, user] ...
REVOKE GRANT OPTION 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
CREATE	Allows use of CREATE TABLE
CREATE TEMPORARY TABLES	Allows use of CREATE TEMPORARY TABLE
DELETE	Allows use of DELETE
DROP	Allows use of DROP TABLE
EXECUTE	Allows the user to run stored procedures (MySQL 5.0)
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 yet implemented
RELOAD	Allows use of FLUSH
REPLICATION CLIENT	Gives the right to the user to ask where the 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
SHUTDOWN	Allows use of mysqladmin shutdown
SUPER	Allows use of CHANGE MASTER, KILL thread, 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

USAGE can be used when you want to create a user that has no privileges.

The privileges CREATE TEMPORARY TABLES, EXECUTE, LOCK TABLES, REPLICATION ..., SHOW DATABASES and SUPER are new for in MySQL 4.0.2. To use these new privileges after upgrading to 4.0.2, you must run the mysql_fix_privilege_tables script. See Section 2.5.8 [Upgrading-grant-tables], page 139.

In older MySQL versions that do not have the SUPER privilege, the PROCESS privilege can be used instead.

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 a current database, the privileges will be granted in that database. (**Warning:** If you specify ON * and you *don't* have a current database, the privileges granted will be 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.

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 will be granted.

MySQL allows you to create 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 create table-level privileges if the table doesn't exist.

MySQL does not automatically revoke any privileges even if you drop a table or drop a database.

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 you want to specify a user_name string containing special characters (such as '-'), or a host_name string containing special characters or wildcard characters (such as '%'), you can quote the username or hostname (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' ...
```

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 will be used when users try to log in to the MySQL server from the local machine! (This 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;
```

For the moment, GRANT only supports host, table, database, and column names up to 60 characters long. A username 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 5.4 [Privilege system], page 275.

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` will 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 already 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.

Passwords can also be set with the `SET PASSWORD` statement. See Section 14.5.1.3 [SET PASSWORD], page 696.

If you don't want to send the password in clear text, you can use the `PASSWORD` keyword followed by a scrambled password from the `PASSWORD()` SQL function or 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). If a user has no privileges for a database, the database name is not displayed by `SHOW DATABASES` unless the user has the `SHOW DATABASES` privilege.

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 already 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. See Section 5.5.5 [User resources], page 307. Note: To specify any of these options for an existing user without affecting existing privileges, use GRANT USAGE ON *.* ... WITH MAX_....

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 5.5.8 [Secure connections], page 311.)

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 unneccessary.

```
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 will be 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.

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.

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 5.4.7 [Privilege changes], page 290.

The biggest differences between the standard SQL and MySQL versions of GRANT are:

- In MySQL, privileges are given for a username/hostname combination and not only for 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. In MySQL, the granted privileges are not automatically revoked; you must revoke them yourself.

- 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 will be set to their default values. Standard SQL requires you to have the `INSERT` privilege on all columns.

14.5.1.3 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 access to 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` fields 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;
```

14.5.2 Table Maintenance Statements

14.5.2.1 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. 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 14.5.3.7 [Show database info], page 708.

If the table hasn't changed since the last `ANALYZE TABLE` statement, the table will not be 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.

14.5.2.2 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 5.6.1 [Backup], page 319.

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.

14.5.2.3 CHECK TABLE Syntax

```
CHECK TABLE tbl_name [, tbl_name] ... [option] ...

option = {QUICK | FAST | MEDIUM | EXTENDED | CHANGED}
```

`CHECK TABLE` works only on `MyISAM` and `InnoDB` tables. On `MyISAM` tables, This is the same thing as running `myisamchk --medium-check tbl_name` on the table.

If you don't specify any option, `MEDIUM` is used.

Checks the table or tables for errors. For `MyISAM` tables, the key statistics are updated. The 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 will have 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 5.6.2 [Table maintenance], page 321. 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 types are as follows:

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 will take 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. 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 will change 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 will change 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.

14.5.2.4 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 14.2.5 [CREATE TABLE], page 670.

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.

This statement is implemented in MySQL 4.1.1.

14.5.2.5 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 and BDB tables. For BDB tables, OPTIMIZE TABLE is currently mapped to ANALYZE TABLE. See Section 14.5.2.1 [ANALYZE TABLE], page 696.

You can get OPTIMIZE TABLE to work on other table types by starting mysqld with the --skip-new or --safe-mode option, but 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.

14.5.2.6 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 on a table. 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.4.1 [Crashing], page 1013. See Section 15.1.4 [MyISAM table problems], page 744.

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 will have 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 will re-create the '.MYI' file using information from the '.frm' file. This kind of repair cannot be done with myisamchk.

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.

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.

14.5.2.7 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 will not be overwritten; if you try to restore over an existing table, you will 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 index. The more indexes the table has, the longer it will take.

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

14.5.3 SET and SHOW Syntax

SET allows you to set variables and options.

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 `SET PASSWORD` statement for assigning account passwords is described in See Section 14.5.1.3 [SET PASSWORD], page 696.
- 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 14.6 [Replication SQL], page 726.

14.5.3.1 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.

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], page 260.

In older versions of MySQL, we allowed the use of the `SET OPTION` syntax, but this is now deprecated; just leave out the word `OPTION`.

The following examples shows 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 10.3 [Variables], page 498.

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 databases.

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 [Option files], page 212.

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 14.5.3.19 [SHOW VARIABLES], page 718. 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], page 240. Although these variables are not displayed by SHOW VARIABLES, you can obtain their values with SELECT (with the exception of CHARACTER SET). 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. Note that when 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 Section 14.4.1 [COMMIT], page 684.

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 will 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 16.7.4 [InnoDB foreign key constraints], page 772.

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 theh 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.

  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_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 will take a very long time (that is, statements for which the optimizer estimates that the number of of examined rows will exceed 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 will be 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 C.1.2 [News-5.0.0], page 1042).

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 will

work for tables with table and column names that require quoting. This variable was added in MySQL 3.23.26. Section 14.5.3.6 [SHOW CREATE TABLE], page 708.

`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_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, no uniqueness checks are done. This variable was added in MySQL 3.23.52. See Section 16.7.4 [InnoDB foreign key constraints], page 772.

14.5.3.2 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.

14.5.3.3 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.

14.5.3.4 SHOW COLUMNS Syntax

```
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [LIKE 'pattern']
```

SHOW COLUMNS lists the columns in a given table. 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 14.2.5.1 [Silent column changes], page 681.

The FULL keyword can be used from MySQL 3.23.32 on. It causes the 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 14.3.1 [DESCRIBE], page 683.

14.5.3.5 SHOW CREATE DATABASE Syntax

```
SHOW CREATE DATABASE db_name
```

Shows a CREATE DATABASE statement that will create the given database. It was added in MySQL 4.1.

```
mysql> SHOW CREATE DATABASE test\G
*************************** 1. row ***************************
       Database: test
Create Database: CREATE DATABASE `test`
                 /*!40100 DEFAULT CHARACTER SET latin1 */
```

14.5.3.6 SHOW CREATE TABLE Syntax

```
SHOW CREATE TABLE tbl_name
```

Shows a CREATE TABLE statement that will create the given table. It was added in MySQL 3.23.20.

```
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 14.5.3.1 [SET SQL_QUOTE_SHOW_CREATE], page 702.

14.5.3.7 SHOW DATABASES Syntax

```
SHOW DATABASES [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 will 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.

14.5.3.8 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 ***************************
   Type: MyISAM
Support: DEFAULT
Comment: Default type from 3.23 with great performance
*************************** 2. row ***************************
   Type: HEAP
Support: YES
Comment: Hash based, stored in memory, useful for temporary tables
*************************** 3. row ***************************
   Type: MEMORY
Support: YES
Comment: Alias for HEAP
*************************** 4. row ***************************
   Type: MERGE
Support: YES
Comment: Collection of identical MyISAM tables
*************************** 5. row ***************************
   Type: MRG_MYISAM
Support: YES
Comment: Alias for MERGE
*************************** 6. row ***************************
   Type: ISAM
Support: NO
Comment: Obsolete table type; Is replaced by MyISAM
*************************** 7. row ***************************
   Type: MRG_ISAM
Support: NO
Comment: Obsolete table type; Is replaced by MRG_MYISAM
*************************** 8. row ***************************
   Type: InnoDB
Support: YES
Comment: Supports transactions, row-level locking and foreign keys
*************************** 9. row ***************************
   Type: INNOBASE
Support: YES
Comment: Alias for INNODB
*************************** 10. row ***************************
   Type: BDB
Support: YES
Comment: Supports transactions and page-level locking
*************************** 11. row ***************************
   Type: BERKELEYDB
Support: YES
Comment: Alias for BDB
```

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 will have the value `DEFAULT`.

14.5.3.9 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 14.1.7 [SELECT], page 643.

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 14.5.3.20 [SHOW WARNINGS], page 719.

14.5.3.10 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 13.8.3 [CURRENT_USER()], page 611.

`SHOW GRANTS` is available as of MySQL 3.23.4.

14.5.3.11 SHOW INDEX Syntax

```
SHOW INDEX FROM tbl_name [FROM db_name]
```

SHOW INDEX returns table index information in a format that closely 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.

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.

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 Comments 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.

14.5.3.12 SHOW INNODB STATUS Syntax

```
SHOW INNODB STATUS
```

This statement shows extensive information about the state of the InnoDB storage engine.

14.5.3.13 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)

14.5.3.14 SHOW PRIVILEGES Syntax

```
SHOW PRIVILEGES
```

SHOW PRIVILEGES shows the list of system privileges that the underlying MySQL server supports. This statement is implemented as of MySQL 4.1.0.

```
mysql> SHOW PRIVILEGES\G
*************************** 1. row ***************************
Privilege: Select
  Context: Tables
  Comment: To retrieve rows from table
*************************** 2. row ***************************
Privilege: Insert
  Context: Tables
  Comment: To insert data into tables
*************************** 3. row ***************************
Privilege: Update
  Context: Tables
  Comment: To update existing rows
*************************** 4. row ***************************
Privilege: Delete
  Context: Tables
  Comment: To delete existing rows
*************************** 5. row ***************************
Privilege: Index
  Context: Tables
  Comment: To create or drop indexes
*************************** 6. row ***************************
Privilege: Alter
  Context: Tables
  Comment: To alter the table
*************************** 7. row ***************************
Privilege: Create
  Context: Databases,Tables,Indexes
```

```
  Comment: To create new databases and tables
*************************** 8. row ***************************
Privilege: Drop
  Context: Databases,Tables
  Comment: To drop databases and tables
*************************** 9. row ***************************
Privilege: Grant
  Context: Databases,Tables
  Comment: To give to other users those privileges you possess
*************************** 10. row ***************************
Privilege: References
  Context: Databases,Tables
  Comment: To have references on tables
*************************** 11. row ***************************
Privilege: Reload
  Context: Server Admin
  Comment: To reload or refresh tables, logs and privileges
*************************** 12. row ***************************
Privilege: Shutdown
  Context: Server Admin
  Comment: To shutdown the server
*************************** 13. row ***************************
Privilege: Process
  Context: Server Admin
  Comment: To view the plain text of currently executing queries
*************************** 14. row ***************************
Privilege: File
  Context: File access on server
  Comment: To read and write files on the server
```

14.5.3.15 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 14.5.4.3 [KILL], page 724. 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 now 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.

`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 now 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 around 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.

14.5.3.16 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], page 262.

```
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  |
+--------------------+----------+
```

14.5.3.17 SHOW TABLE STATUS Syntax

```
SHOW TABLE STATUS [FROM db_name] [LIKE 'pattern']
```

SHOW TABLE STATUS (new in MySQL 3.23) 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.

SHOW TABLE STATUS returns the following fields:

Name
: The name of the table.

Type
: The type of the table. See Chapter 15 [Table types], page 737.

Row_format
: The row storage format (Fixed, Dynamic, or Compressed).

Rows
: The number of rows.

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. For fixed-row formats, this is the maximum number of rows in the table. For dynamic-row formats, 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 will 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.

14.5.3.18 SHOW TABLES Syntax

```
SHOW [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.

Note: If you have no privileges for a table, the table will 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 [Table cache], page 435. 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.

14.5.3.19 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 will get the values that will be used for new connections to MySQL. With SESSION, you will get the values that are in effect for the current connection. If you use neither option, the default 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 [Server options], page 228 and Section 14.5.3.1 [SET], page 702.

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], page 240. Information about tuning them is provided in Section 7.5.2 [Server parameters], page 437.

```
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      |
+--------------------+----------+
```

14.5.3.20 `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 14.5.3.9 [SHOW ERRORS], page 710.

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 14.1.7 [SELECT], page 643.

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 20.2.3.58 [mysql_warning_count()], page 908.

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.

14.5.4 Other Administrative Statements

14.5.4.1 CACHE INDEX Syntax

```
CACHE INDEX
  table_index_list [, table_index_list] ...
  IN key_cache_name

table_index_list:
  tbl_name [[INDEX] (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 10.4.1 [Structured System Variables], page 501.

A key cache must exist before you can assign indexes to it:

```
mysql> CACHE INDEX t1 in non_existent_cache;
ERROR 1283 (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.

14.5.4.2 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 connection requests. Flushing the host tables allows the host to attempt to connect again. See Section A.2.6 [Blocked host], page 1003. 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 will be incremented by one relative to the previous file. If you have used an extension in the file name, MySQL will close and reopen the update log or binary log file. See Section 5.8.3 [Update log], page 346. On Unix, this is the same thing as sending a SIGHUP signal to the mysqld server.

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 1.7.1.3 [Bug reports], page 34.

{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 user resources to zero. This enables clients that have reached their hourly connection, query, or update limits to resume activity. See Section 14.5.1.2 [GRANT], page 689.

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 14.5.4.5 [RESET], page 725.

14.5.4.3 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 will not be rolled back!
- GET_LOCK() will abort and return NULL.
- An INSERT DELAYED thread will quickly flush all rows it has in memory and terminate.
- If the thread is in the table lock handler (state: Locked), the table lock will be 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.

14.5.4.4 LOAD INDEX INTO CACHE Syntax

```
LOAD INDEX INTO CACHE
  table_index_list [, table_index_list] ...

table_index_list:
  tbl_name
    [[INDEX] (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.

14.5.4.5 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 14.5.4.2 [FLUSH], page 722.

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 14.6.1 [Replication Master SQL], page 726.

`QUERY CACHE`
Removes all query results from the query cache.

`SLAVE` Makes the slave forget its replication position in the master binary logs. Previously named `FLUSH SLAVE`. See Section 14.6.2 [Replication Slave SQL], page 728.

14.6 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.

14.6.1 SQL Statements for Controlling Master Servers

Replication can be controlled through the SQL interface. This section discusses statements for managing master replication servers. Section 14.6.2 [Replication Slave SQL], page 728 discusses statements for managing slave servers.

14.6.1.1 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 will be 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 will be 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.

14.6.1.2 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.

14.6.1.3 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.)

14.6.1.4 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 will be displayed.

The LIMIT clause has the same syntax as for the SELECT statement. See Section 14.1.7 [SELECT], page 643.

This statement is available as of MySQL 4.0.

14.6.1.5 SHOW MASTER LOGS Syntax

```
SHOW MASTER LOGS
```

Lists the binary log files on the master. This statement is used as part of the procedure described in Section 14.6.1.1 [PURGE MASTER LOGS], page 726 for determining which logs can be purged.

14.6.1.6 SHOW MASTER STATUS Syntax

```
SHOW MASTER STATUS
```

Provides status information on the binary log files of the master.

14.6.1.7 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 will not be visible in that list.

14.6.2 SQL Statements for Controlling Slave Servers

Replication can be controlled through the SQL interface. This section discusses statements for managing slave replication servers. Section 14.6.1 [Replication Master SQL], page 726 discusses statements for managing master servers.

14.6.2.1 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 is 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 last coordinates of the *slave SQL thread* before CHANGE MASTER was issued are used. 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 will be kept; as of MySQL 4.1.1 the relay_log_purge global variable will silently be set 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, slave-of-nobody 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 will read and execute 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.

14.6.2.2 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 will start 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 like this: --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 the non-blocking online backup feature.

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], page 240.

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 that the replication account that is used to connect to the master has 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 will hide 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 14.5.3.7 [Show database info], page 708. 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.

14.6.2.3 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.

14.6.2.4 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 13.8.4 [Miscellaneous functions], page 616 for a full description.

14.6.2.5 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.

14.6.2.6 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.

14.6.2.7 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], page 364. 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 will make you aware of the connection problem. On the opposite, the state of the SQL thread is not copied, because things are simpler for this thread: If it is running, there is no problem; if it is not, you will 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 will also appear 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 will stop 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 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`

The number of seconds that have elapsed since the timestamp of the last master's event executed by the slave SQL thread. This will be `NULL` when no event has been executed yet, or after `CHANGE MASTER` and `RESET SLAVE`. This field can be used to know how "late" your slave is. It will work even though your master and slave don't have identical clocks.

This field is present beginning with MySQL 4.1.1.

14.6.2.8 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` will 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 already 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.

14.6.2.9 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.

15 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 will be 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 now is known as 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.

This chapter describes each of the MySQL storage engines except for `InnoDB`, which is covered in Chapter 16 [InnoDB], page 758.

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 table type is usually `MyISAM`. This can be changed by setting the `table_type` system variable.

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 14.2.5 [`CREATE TABLE`], page 670 and Section 14.2.2 [`ALTER TABLE`], page 663.

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 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 your changes will be restored. (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 16.4 [`InnoDB` configuration], page 759.

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.

15.1 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 5.6.2.7 [Crash recovery], page 329. You can compress MyISAM tables with myisampack to take up much less space. See Section 8.2 [myisampack], page 449.

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 will use less system resources than ISAM, but will need 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 deleted values 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 14.2.5 [CREATE TABLE], page 670.
- 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 (and optionally repaired) when opened 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 will allow you to have UNIQUE on any combination of columns in a table. (You can't search on a UNIQUE computed index, however.)

15.1.1 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 will lead to index corruption.

  Using --external-lock will not help for tables that use --delay-key-write.

See Section 5.2.1 [Server options], page 228.

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], page 240.

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.

15.1.2 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 will also be 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.

15.1.3 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 14.2.5 [CREATE TABLE], page 670.

15.1.3.1 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.

15.1.3.2 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 will be 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 will be at least 20 bytes, so the next enlargement will probably go in the same link. If not, there will be another link. You may check how many links there are with myisamchk -ed. All links may be removed with myisamchk -r.

15.1.3.3 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 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.

15.1.4 `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.

15.1.4.1 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 14.5.2.3 [CHECK TABLE], page 697, Section 14.5.2.6 [REPAIR TABLE], page 700, and Section 5.6.2.1 [myisamchk syntax], page 322.

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 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.4.1 [Crashing], page 1013 and Section D.1.6 [Reproduceable test case], page 1204.

15.1.4.2 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 5.9 [Multiple servers], page 351 for additional discussion.

15.2 The MERGE Storage Engine

The MERGE storage engine was introduced in MySQL 3.23.25. It is also known as the MRG_MyISAM engine. The code is now reasonably stable.

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 packed differently, 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], page 449.

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. (You can also use a RAID table to get the same kind of benefits.)
- 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 will save 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 will need 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], page 402 for more information about eq_ref and ref.

15.2.1 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 will get unexpected results. The easiest way to work around this deficiency is to issue 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 will be 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 will be 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. We will fix this at the same time we introduce views.

15.3 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. 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 will be 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 [MySQL indexes], page 427.
- 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 will be 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.
- 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).
- The MEMORY table property that table contents are stored in memory is one that is shared with internal tables that the server creates on the fly while processing queries. However, internal tables also have the property that the server converts them to on-disk tables automatically if they become too large. The size limit is determined by the value of the tmp_table_size system variable.

  MEMORY tables are not 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 [Server options], page 228.
- 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 will return 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_KEYS(max_length_of_key + 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.

15.4 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 the MySQL source distribution 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 will be forwarded to Sleepycat, which in turn will help 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. See Section 1.4.1 [Support], page 16.

For general information about Berkeley DB, please visit the Sleepycat Web site, `http://www.sleepycat.com/`.

15.4.1 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 will 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.

15.4.2 Installing `BDB`

If you have downloaded a binary version of MySQL that includes support for Berkeley DB, simply follow 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 2.2.5 [Installing binary], page 95, Section 5.1.2 [`mysqld-max`], page 219, and See Section 2.3 [Installing source], page 98.

15.4.3 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 [Server options], page 228.

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], page 240.

If you use the --skip-bdb option, MySQL will not initialize the Berkeley DB library and this will save a lot of memory. However, if you use this option, you cannot use BDB tables. If you try to create a BDB table, MySQL will create 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 2.4.2.3 [Starting server], page 124.

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 5.8.4 [Binary log], page 347.

15.4.4 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 14.4.1 [COMMIT], page 684.

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 will be 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 14.5.2.1 [ANALYZE TABLE], page 696 and Section 14.5.2.5 [OPTIMIZE TABLE], page 699.
- 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 will be 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 5.6.1 [Backup], page 319.

  Warning: If you delete old log files that are still in use, BDB will not be 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 will get an error (probably error 28) and the transaction should roll back. This contrasts with MyISAM and ISAM tables, for which mysqld will wait for enough free disk before continuing.

15.4.5 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.

15.4.6 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.

15.4.7 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.)

15.5 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 now is deprecated. As of MySQL 4.1, it's included in the source but not enabled in binary distributions. It will disappear 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 MySAM 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 15.1 [MyISAM], page 738.

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 5.6.2.7 [Crash recovery], page 329.

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.

16 The InnoDB Storage Engine

16.1 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 16.3 [InnoDB in MySQL 3.23], page 759.

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. If you distribute MySQL/InnoDB, and your application does not satisfy the provisions of the GPL license, you must purchase a commercial **MySQL Pro** license from https://order.mysql.com/?sub=pg&pg_no=1.

16.2 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
```

16.3 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 already 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 2.2.5 [Installing binary], page 95 and Section 5.1.2 [mysqld-max], page 219.

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 2.3 [Installing source], page 98.

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 will not start. (From MySQL 4.0 on, InnoDB uses default parameters if you do not specify any. However, it is still recommended that to get best performance, you should use parameters appropriate for your system, as discussed in Section 16.4 [InnoDB configuration], page 759.)

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 10MB auto-extending data file, place the following setting in the [mysqld] section of your option file:

```
[mysqld]
innodb_data_file_path=ibdata1:10M:autoextend
```

InnoDB will create the 'ibdata1' file in the MySQL data directory by default. To specify the location explicitly, specify an innodb_data_home_dir setting. See Section 16.4 [InnoDB configuration], page 759.

16.4 InnoDB Configuration

To enable InnoDB tables in MySQL 3.23, see Section 16.3 [InnoDB in MySQL 3.23], page 759.

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 will 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 16.8 [Adding and removing], page 778.

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 makes use of 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 will then print 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 16.6 [InnoDB init], page 767 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.

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.
- 'COMPILATION_DATADIR/my.cnf'

  Server-specific options.
- 'defaults-extra-file'

  The file specified with the --defaults-extra-file option.
- '~/.my.cnf'

  User-specific options.

COMPILATION_DATADIR is the MySQL data directory that was specified as a configure option when mysqld was compiled (typically '/usr/local/mysql/data' for a binary installation or '/usr/local/var' for a source installation).

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 will fill the tablespace beginning with the first data file. In some cases, it will improve 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 16.15.2 [InnoDB Raw Devices], page 801.

Warning: On GNU/Linux x86, you must be careful not to set memory usage too high. glibc will allow the process heap to grow over thread stacks, which will crash 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 will use 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, 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 16.5 [InnoDB start], page 764.

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=...
```

16.5 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 will need to allocate here. If `InnoDB` runs out of memory in this pool, it will start to allocate memory from the operating system, and write warning messages to the MySQL error log. The default value is 1MB.

`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 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_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 16.15.2 [InnoDB Raw Devices], page 801.

`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_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`
: 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 16.7.6 [Multiple tablespaces], page 776. This option is available as of MySQL 4.1.1.

`innodb_flush_log_at_trx_commit`
: Normally you set this to 1, meaning that at a transaction commit, the log is flushed to disk, and the modifications made by the transaction become permanent and survive a database crash. If you are willing to compromise this safety, and you are running small transactions, you may set this to 0 or 2 to reduce disk I/O to the logs. A value of 0 means that the log is only written to the log file and the log file flushed to disk approximately once per second. A value of 2 means the log is written to the log file at each commit, but the log file is only flushed to disk approximately once per second. The default value is 1 (prior to MySQL 4.0.13, the default 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 16.9.1 [Forcing recovery], page 780. 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. 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_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 will save 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 will be 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_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_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.

16.6 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 will print 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 now 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 will be 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 now look at the data file and log directories and you will see the files created. The log directory will also contain 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 will already have been created, so the output will be much briefer:

```
InnoDB: Started
mysqld: ready for connections
```

16.6.1 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 or log directories.
- `mysqld` does not have access rights to create files in those directories.
- `mysqld` does 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 data files, all log files, and the small archived log file. In case you already 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.

16.7 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 will 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.

16.7.1 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.

16.7.2 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 data has 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 will start 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 16.9.1 [Forcing recovery], page 780.

16.7.3 How an AUTO_INCREMENT Column Works in InnoDB

If you specify an AUTO_INCREMENT column for a table, the InnoDB table handle in the data dictionary will contain 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.

16.7.4 `FOREIGN KEY` Constraints

Starting from MySQL 3.23.43b, `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 {CASCADE | SET NULL | NO ACTION | RESTRICT}]
    [ON UPDATE {CASCADE | SET NULL | NO ACTION | RESTRICT}]
```

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.

`InnoDB` does not automatically create indexes on foreign keys or referenced keys: You must create them explicitly. The indexes are needed so that foreign key checks can be fast and not require a table scan.

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.

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 already 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.

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 {CASCADE | SET NULL | NO ACTION | RESTRICT}]
    [ON UPDATE {CASCADE | SET NULL | NO ACTION | RESTRICT}]
```

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 will confuse 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 allows you to drop any table, even though that would break the foreign key constraints that reference the table. 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.

16.7.5 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 16.10 [Moving], page 782.

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.

16.7.6 Using Per-Table Tablespaces

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.

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 earlier versions of `InnoDB` are not aware of multiple tablespaces.

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. A direct downgrade is not recommended, however, because it has not been extensively tested.

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 will store 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.

16.8 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` will increase the size of that file automatically in 8MB increments when it runs out of space.

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 already 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 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 so that `InnoDB` creates it and initializes it 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 will 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` will see that the no log files exist at startup and tell you that it is creating new ones.

16.9 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 into a safe place.
3. Copy all your `InnoDB` log files 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` will automatically check the logs and perform a roll-forward of the database to the present. `InnoDB` will automatically roll back uncommitted transactions that were present at the time of the crash. During recovery, `mysqld` will display 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, though `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.

16.9.1 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 will be 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` will treat 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 these options! 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.

16.9.2 Checkpoints

`InnoDB` implements a checkpoint mechanism called a "fuzzy checkpoint." `InnoDB` will flush 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 already 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 already 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 will be more logged information to apply to the database.

16.10 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 16.9 [Backing up], page 779. 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 big import transaction will generate. Do the commit only after importing a whole table or a segment of a table.

16.11 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.

16.11.1 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 think that a user always has a transaction open. An SQL COMMIT or ROLLBACK statement ends the current transaction and a new one starts. Both statements will 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.

16.11.2 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 16.11.3 [InnoDB consistent read], page 784.

- `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 16.11.3 [InnoDB consistent read], page 784.

- `SERIALIZABLE`

  This level is like `REPEATABLE READ`, but all plain `SELECT` statements are implicitly converted to `SELECT ... LOCK IN SHARE MODE`.

16.11.3 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 will 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 will see 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.

16.11.4 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 already 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 now 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 will 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 will occur 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 will end 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.

16.11.5 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 will scan that index starting from the first record where `id` is bigger than 100. Now, 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 now 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 will prevent 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.

16.11.6 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` will give 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 will 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;
```

16.11.7 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 will become 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 16.11.1 [`InnoDB` and `AUTOCOMMIT`], page 783.

  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. The automatic deadlock detection of `InnoDB` cannot detect deadlocks where such table locks are involved. See Section 16.11.9 [InnoDB Deadlock detection], page 789.

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 16.17 [InnoDB restrictions], page 807.

16.11.8 When Does MySQL Implicitly Commit or Roll Back a Transaction?

MySQL begins each client connection with autocommit mode enabled by default. When autocommit 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 will roll back your transaction.

If an error is returned by an SQL statement, the commit/rollback behavior depends on the error. See Section 16.16 [InnoDB Error handling], page 802.

The following SQL statements 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. It means that a `ROLLBACK` from the user does not undo `CREATE TABLE` statements the user made during that transaction.

16.11.9 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.

`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.

16.11.10 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. For example, if you need to write table `t1` and read table `t2`, you can do this:

  ```
  LOCK TABLES t1 WRITE, t2 READ, ...;
  [do something with tables t1 and t2 here];
  UNLOCK TABLES;
  ```

  Table-level locks make your transactions queue nicely, and deadlocks are avoided. Note that `LOCK TABLES` implicitly starts a transaction, just like the statement `BEGIN`, and `UNLOCK TABLES` implicitly ends the transaction in a `COMMIT`.
- 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.

16.12 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 167/second if the disk does not fool the operating system.
- If you can afford the loss of some 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 will cause many unnecessary disk writes. The drawback of big log files is that recovery time will be 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, though 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 will 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 will not help because the rollback will start 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 16.9.1 [Forcing recovery], page 780.

- Beware also of other big disk-bound operations. Use `DROP TABLE` or `TRUNCATE TABLE` (from MySQL 4.0 up) 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.

16.12.1 `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 information 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 database 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, 'hgjhjgghggjgjgjgjgjggjgjgjgjgjgggjgjgjlhh
gghggggghhjhghgggggghjhghghghghghhhhghghghjhhjghjghjkghjghjghjghjfhjfh
---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, 'hgjhjgghggjgjgjgjgjggjgjgjgjgjgggjgjgjlh
hgghggggghhjhghgggggghjhghghghghghhhhghghghjhhjghjghjkghjghjghjghjfhjf
---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.

16.13 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.

16.14 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 will be 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 will be physically in the insertion order.

Accessing a row through the clustered index is fast because the row data will be 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.

16.14.1 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 will be about 15/16 full. If records are inserted in a random order, the pages will be 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.

16.14.2 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 already 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.

16.14.3 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 will build hash indexes on demand for those pages of the index that are often accessed.

In a sense, InnoDB adapts itself through the adaptive hash index mechanism to ample main memory, coming closer to the architecture of main memory databases.

16.14.4 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.
- 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 14.2.5.1 [Silent column changes], page 681.
- An SQL NULL value reserves zero bytes if stored in a variable length column. In a fixed-length column, it reserves the fixed length of the column. 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.

16.15 File Space Management and Disk I/O

16.15.1 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 later will find a good copy of the page from the doublewrite buffer during recovery.

16.15.2 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 will reinitialize the partition and your changes will be 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 will allow 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.

16.15.3 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 will be 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 will contract 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 will be physically removed only in an (automatic) purge operation after they are no longer needed in transaction rollback or consistent read.

16.15.4 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.

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 will not occur.

16.16 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 will 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 only the insert of that particular row, even in a statement like INSERT INTO ... SELECT. This will probably change so that the SQL statement will be rolled back 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.

16.16.1 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 16.18.1 [InnoDB troubleshooting datadict], page 809.

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.

16.16.2 Operating System Error Codes

In Unix, 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/e` `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.

`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.

16.17 Restrictions on InnoDB Tables

- A table cannot contain more than 1000 columns.
- The maximum key length is 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 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.
- 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 will 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 will take some time if the table 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 16.12 [InnoDB tuning], page 791.
- 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 about a million years before BIGINT reached its upper bound. With all other integer type columns, a duplicate-key error will result. This is similar to how MyISAM works, as 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 will 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 will create an index over the whole column.

  These restrictions are removed starting from 4.0.14 and 4.1.1.
- INSERT DELAYED is not supported for InnoDB tables.
- The MySQL LOCK TABLES operation does not know about InnoDB row-level locks set by already 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 will 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.

- 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.

16.18 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 on Windows '`mysql.err`'.
- 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 will 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.

16.18.1 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 an '.InnoDB' file:

```
ERROR 1016: Can't open file: 'child2.InnoDB'. (errno: 1)
```

In the error log you will 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...', but since MySQL does not allow accessing any table with such a name, you cannot dump or drop it. The solution is to use a special 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;
```

The backticks around the table name are needed because a temporary table name contains the character '-'.

The table definition must be similar to that of the temporary table. If you do not know the definition of the temporary table, you can use an arbitrary definition in the preceding CREATE TABLE statement, and after that replace the file 'rsql_id.frm' by the file '#sql_id.frm' of the temporary table. Note that to copy or rename a file in the shell, you need to put the file name in double quotes if the file name contains '#'. Then you can dump and drop the renamed table.

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 G [GPL license], page 1216), 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 G [GPL license], page 1216). 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 (Section 1.4.3 [MySQL licenses], page 17). Thus, MaxDB will be available under the GNU General Public License (Appendix G [GPL license], page 1216), and a commercial license (Section 1.4 [Licensing and Support], page 16).

MySQL will offer 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/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

The following features will be included in MaxDB versions to be released shortly after the first 7.5.00 version. These features will allow interoperation between MaxDB and MySQL.

- There will be a MySQL proxy enabling connections to MaxDB using the MySQL protocol. This makes it possible to use MySQL client programs for MaxDB, such as the `mysql` command-line user interface, the `mysqldump` dump utility, or the `mysqlimport` import program. Using `mysqldump`, you can easily dump data from one database system and export (or even pipe) those data to the other database system.

- Replication between MySQL and MaxDB will be supported in both directions. That is, either MySQL or MaxDB can be used as the master replication server. The long-range plan is to converge and extend the replication syntax so that both database systems understand the same syntax. See Section 6.1 [Replication Intro], page 363.

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 10.6 [Reserved words], page 503.

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, DROP SYNONYM	Nothing comparable
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	mysql command-line user interface command	USE
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

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 ® 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.
- 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))
  ```
- The quality of being **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.
- The quality of being **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.

- The quality of being **empty** or **not 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 using WKB `LineString` arguments. If any argument is not a `LineString`, the return value is `NULL`.

`MultiPoint(pt1,pt2,...)`
: Constructs a WKB `MultiPoint` value using WKB `Point` arguments. If any argument is not a `WKBPoint`, 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 rerurn 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 20.2.3.44 [mysql_real_escape_string()], page 898.

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 [GIS WKT Functions], page 825.

`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 [GIS WKB Functions], page 826.

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 [GIS class geometry], page 818.)

```
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 will correspond 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`.

```
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 property functions], page 831, we've already 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 already 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 16000,30000
+-----+---------------------------------------------------------------------------
| 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)
```

Now let's use `EXPLAIN` to check the way this query is executed:

```
mysql> EXPLAIN SELECT fid,AsText(g) FROM geom WHERE
mysql> MBRContains(GeomFromText('Polygon((30000 15000,31000 15000,31000 16000,30000
+----+-------------+-------+-------+---------------+------+---------+------+------+
| id | select_type | table | type  | possible_keys | key  | key_len | ref  | rows |
+----+-------------+-------+-------+---------------+------+---------+------+------+
|  1 | SIMPLE      | geom  | range | g             | g    |      32 | NULL |   50 |
+----+-------------+-------+-------+---------------+------+---------+------+------+
1 row in set (0.00 sec)
```

Now let's 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 16000,30000
+----+-------------+-------+------+---------------+------+---------+------+-------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows  |
+----+-------------+-------+------+---------------+------+---------+------+-------+
|  1 | SIMPLE      | geom  | ALL  | NULL          | NULL |    NULL | NULL | 32376 |
+----+-------------+-------+------+---------------+------+---------+------+-------+
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 16000,30000
+-----+---------------------------------------------------------------------------
| 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], page 838.

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.

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.

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. And, often, there are multiple client machines (such as Web servers) but only one or a few database servers.

Stored procedures also allow you to have libraries of functions in the database server. However, modern application languages already allow such design internally with, for example, classes. Using these client application language features is beneficial for the programmer even outside the scope of database use.

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.

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.

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 2.5.8 [Upgrading-grant-tables], page 139.

19.1 Stored Procedure Syntax

Stored procedures and functions are routines that are created with `CREATE PROCEDURE` and `CREATE FUNCTION` statements. A procedure is invoked using a `CALL` statement, and can only pass back values using output variables. Functions may return a scalar value and can be called from inside a statement just like any other function (that is, by invoking the function's name). Stored routines may call other stored routines. A routine is either a procedure or a function.

At present, MySQL only preserves context for the default database. That is, if you say `USE db_name` within a procedure, the original default database is restored upon routine exit. A routine inherits the default database from the caller, so generally routines should either issue a USE db_name statement, or specify all tables with an explicit database reference; for example, db_name.tbl_name.

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.

This following section describes the syntax used to create, alter, drop, and query stored procedures and functions.

19.1.1 Maintaining Stored Procedures

19.1.1.1 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 statement(s)
```

The RETURNS clause may be specified only for a FUNCTION. 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 22.2 [Adding functions], page 987. 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 will be PHP because the core PHP engine is small, thread-safe, and can easily be embedded. Because the framework will be public, it is expected that many other languages will also be supported.

A function is considered "deterministic" if it always returns 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.

MySQL does not yet use the `GRANT EXECUTE` privilege.

MySQL stores the `sql_mode` system variable setting that is in effect at the time a routine is created, and will always execute 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 is therefore 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)
```

19.1.1.2 ALTER PROCEDURE and ALTER FUNCTION

```
ALTER {PROCEDURE | FUNCTION} sp_name [characteristic ...]

characteristic:
    NAME new_name
  | SQL SECURITY {DEFINER | INVOKER}
  | COMMENT string
```

This statement can be used to rename a stored procedure or function, and to change its characteristics. More than one change may be specified in an `ALTER PROCEDURE` or `ALTER FUNCTION` statement.

19.1.1.3 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.

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`.

19.1.1.4 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.

19.1.2 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 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.

19.1.3 CALL

```
CALL sp_name([parameter[,...]])
```

The CALL statement is used to invoke a routine that was defined previously with CREATE PROCEDURE.

19.1.4 BEGIN ... END Compound Statement

```
[begin_label:] BEGIN
    statement(s)
END [end_label]
```

Stored routines may contain multiple statements, using a BEGIN ... END compound statement.

begin_label and end_label must be the same, if both are specified.

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.

19.1.5 DECLARE Statement

The DECLARE statement is used to define various items local to a routine: local variables (see Section 19.1.6 [Variables in Stored Procedures], page 850), conditions and handlers (see Section 19.1.7 [Conditions and Handlers], page 850) and cursors (see Section 19.1.8 [Cursors], page 852). 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.

19.1.6 Variables in Stored Procedures

You may declare and use variables within a routine.

19.1.6.1 DECLARE Local Variables

```
DECLARE var_name[,...] type [DEFAULT value]
```

This statement is used to declare local variables. The scope of a variable is within the `BEGIN ... END` block.

19.1.6.2 Variable SET Statement

```
SET var_name = expression [,...]
```

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, server 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 global/system variables; in that case, the options are accepted but ignored.

19.1.6.3 SELECT ... INTO Statement

```
SELECT col_name[,...] INTO var_name[,...] table_expression
```

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;
```

19.1.7 Conditions and Handlers

Certain conditions may require specific handling. These conditions can relate to errors, as well as general flow control inside a routine.

19.1.7.1 DECLARE Conditions

```
DECLARE condition_name CONDITION FOR condition_value

condition_value:
    SQLSTATE [VALUE] sqlstate_value
  | mysql_error_code
```

This statement specifies conditions that will 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 19.1.7.2 [DECLARE Handlers], page 851.

In addition to SQLSTATE values, MySQL error codes are also supported.

19.1.7.2 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`.

19.1.8 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.

For example:

```
CREATE PROCEDURE curdemo()
BEGIN
  DECLARE done INT DEFAULT 0;
  DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET done = 1;
  DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
  DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
  DECLARE a CHAR(16);
  DECLARE b,c INT;

  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
```

19.1.8.1 Declaring Cursors

```
DECLARE cursor_name CURSOR FOR sql_statement
```

Multiple cursors may be defined in a routine, but each must have a unique name.

19.1.8.2 Cursor `OPEN` Statement

```
OPEN cursor_name
```

This statement opens a previously declared cursor.

19.1.8.3 Cursor `FETCH` Statement

```
FETCH cursor_name
```

This statement fetches the next row (if a row exists) using the specified open cursor, and advances the cursor pointer.

19.1.8.4 Cursor `CLOSE` Statement

```
CLOSE cursor_name
```

This statement closes a previously opened cursor.

19.1.9 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.

19.1.9.1 `IF` Statement

```
IF search_condition THEN statement(s)
    [ELSEIF search_condition THEN statement(s)]
    ...
    [ELSE statement(s)]
END IF
```

`IF` implements a basic conditional construct. If the `search_condition` evaluates to true, the corresponding SQL statement is executed. If no `search_condition` matches, the statement in the `ELSE` clause is executed.

Please note that there is also an `IF()` function. See Section 13.2 [Control flow functions], page 562.

19.1.9.2 CASE Statement

```
CASE case_value
    WHEN when_value THEN statement
    [WHEN when_value THEN statement ...]
    [ELSE statement]
END CASE
```

Or:

```
CASE
    WHEN search_condition THEN statement
    [WHEN search_condition THEN statement ...]
    [ELSE statement]
END CASE
```

`CASE` 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 inside a stored procedure differs slightly from that of the SQL `CASE` expression. The `CASE` statement cannot have an `ELSE NULL` clause, and the construct is terminated with `END CASE` instead of `END`. See Section 13.2 [Control flow functions], page 562.

19.1.9.3 LOOP Statement

```
[begin_label:] LOOP
    statement(s)
END LOOP [end_label]
```

`LOOP` implements a simple loop construct, enabling repeated execution of a particular statement or group of statements. The statements within the loop are repeated until the loop is exited; usually this is accomplished with a `LEAVE` statement.

`begin_label` and `end_label` must be the same, if both are specified.

19.1.9.4 LEAVE Statement

```
LEAVE label
```

This statement is used to exit any flow control construct.

19.1.9.5 ITERATE Statement

```
ITERATE label
```

`ITERATE` can only appear within `LOOP`, `REPEAT`, and `WHILE` statements. `ITERATE` means “do the loop iteration 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
```

19.1.9.6 REPEAT Statement

```
[begin_label:] REPEAT
    statement(s)
UNTIL search_condition
END REPEAT [end_label]
```

The statements within a `REPEAT` statement are repeated until the `search_condition` is true.

`begin_label` and `end_label` must be the same, if both are specified.

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)
```

19.1.9.7 WHILE Statement

```
[begin_label:] WHILE search_condition DO
    statement(s)
END WHILE [end_label]
```

The statements within a `WHILE` statement are repeated as long as the `search_condition` is true.

`begin_label` and `end_label` must be the same, if both are specified.

For example:

```
CREATE PROCEDURE dowhile()
BEGIN
  DECLARE v1 INT DEFAULT 5;

  WHILE v1 > 0 DO
    ...
    SET v1 = v1 - 1;
  END WHILE;
END
```

20 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, as it was developed by the MySQL team, and is the basis for most of the other APIs.

20.1 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.

20.1.1 `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 [replace utility], page 488.

20.1.2 `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 subsitute its output into the surrounding command.

20.2 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 [Client-Side Scripts], page 448, 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 16M). 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 [Server parameters], page 437.

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 20.2.14 [Threaded clients], page 950. 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 20.2.15 [libmysqld], page 951.

20.2.1 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 counted 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 will 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 will evaluate 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.

20.2.2 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 20.2.3 [C API functions], page 866.

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 command `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 command `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 query.
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_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_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 (new in 4.1).
mysql_rollback()	Rolls back the transaction (new in 4.1).
mysql_autocommit()	Toggles autocommit mode on/off (new in 4.1).
mysql_more_results()	Checks whether any more results exist (new in 4.1).
mysql_next_result()	Returns/Initiates the next result in multi-query executions (new in 4.1).

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`. 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 already 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.

20.2.3 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()`.

20.2.3.1 `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 one specifies the flag CLIENT_FOUND_ROWS when connecting to mysqld, mysql_affected_rows() will return the number of rows matched by the WHERE statement for UPDATE statements.

Note that when one uses a REPLACE command, mysql_affected_rows() will return 2 if the new row replaced and old row. This is because in this case one row was inserted after the duplicate was deleted.

20.2.3.2 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 will always ROLLBACK any active transactions, close all temporary tables, unlock all locked tables and reset the state as if one had done a new connect. This will happen 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));
}
```

20.2.3.3 `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.

20.2.3.4 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.

20.2.3.5 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()`.

20.2.3.6 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));
}
```

20.2.3.7 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(stmt)-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.

20.2.3.8 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 D.1 [Debugging server], page 1199. See Section D.2 [Debugging client], page 1204.

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");
```

20.2.3.9 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));
```

20.2.3.10 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.

20.2.3.11 `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()` will always return 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 now 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));
}
```

20.2.3.12 `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 Section 21.1 [Error-returns], page 969.

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 will 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.

20.2.3.13 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 will reset mysql_error() if they succeed.

For functions that resets 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 5.7.2 [Languages], page 341.

Return Values

A null-terminated character string that describes the error. An empty string if no error occurred.

Errors

None.

20.2.3.14 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.

20.2.3.15 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);
}
```

20.2.3.16 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);
}
```

20.2.3.17 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);
}
```

20.2.3.18 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]);
        }
    }
```

20.2.3.19 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");
}
```

20.2.3.20 `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 20.2.12.1 [`NULL mysql_store_result()`], page 947.

Return Values

An unsigned integer representing the number of fields 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`.

20.2.3.21 `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()` will retrieve 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.

20.2.3.22 `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.

20.2.3.23 `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.

20.2.3.24 `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.

20.2.3.25 `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`.

Return Values

An integer that represents the MySQL client library version.

Errors

None.

20.2.3.26 `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.

20.2.3.27 `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.

20.2.3.28 `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.

20.2.3.29 `mysql_get_server_version()`

`unsigned long mysql_get_server_version(MYSQL *mysql)`

Description

Returns the version number of the server as an integer (new in 4.1).

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.

20.2.3.30 `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 will contain 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.

20.2.3.31 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 will be 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.

20.2.3.32 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 13.8.3 [Information functions], page 611.

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.

20.2.3.33 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.

20.2.3.34 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.

20.2.3.35 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.

20.2.3.36 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.

20.2.3.37 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.

20.2.3.38 `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 20.2.12.1 [`NULL mysql_store_result()`], page 947.

Return Values

An unsigned integer representing the number of fields 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 will happen only if something went wrong.

20.2.3.39 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() will 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.

20.2.3.40 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 options values:

Option	Argument Type	Function
MYSQL_OPT_CONNECT_TIMEOUT	unsigned int *	Connect timeout in seconds.
MYSQL_OPT_READ_TIMEOUT	unsigned int *	Timeout for reads from server (works currently only on Windows on TCP/IP connections)
MYSQL_OPT_WRITE_TIMEOUT	unsigned int *	Timeout for writes to server (works currently only on Windows on TCP/IP connections)

MYSQL_OPT_COMPRESS	Not used	Use the compressed client/server protocol.
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_INIT_COMMAND	char *	Command to execute when connecting to the MySQL server. Will automatically be re-executed when reconnecting.
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_OPT_PROTOCOL	unsigned int *	Type of protocol to use. Should be one of the enum values of mysql_protocol_type defined in 'mysql.h'.
MYSQL_SHARED_MEMORY_BASE_NAME	char*	Named of 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.

Note that the group client 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 20.2.3.43 [mysql_real_connect], page 896.

`local-infile[=(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.
`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=name`	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` will still work for a while.

For more information about option files, see Section 4.3.2 [Option files], page 212.

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.

20.2.3.41 `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.

20.2.3.42 `mysql_query()`

`int mysql_query(MYSQL *mysql, const char *query)`

Description

Executes the SQL query pointed to by the null-terminated string `query`. The query must consist of a single SQL statement. You should not add a terminating semicolon ('`;`') or `\g` to the statement.

`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 20.2.3.20 [`mysql_field_count()`], page 880.

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.

20.2.3.43 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 of the other API functions, with the exception of mysql_get_client_info().

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 20.2.3.40 [mysql_options], page 892.
- 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 20.3.2 [ODBC administrator], page 958.
- 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 will be 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 will set the default database to this value.
- If port is not 0, the value will be 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 Nescription**
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 will be 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-row-queries (separated by ';'). If this flag is not set, multiple-row-queries are disabled. New in 4.1.
CLIENT_MULTI_RESULTS	Tell the server that the client can handle multiple-result sets from multi-queries 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 will read the `[client]` and `[your_prog_name]` sections in the '`my.cnf`' file which will ensure that your program will work, 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`. 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.

20.2.3.44 `mysql_real_escape_string()`

`unsigned long mysql_real_escape_string(MYSQL *mysql, 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 10.1.1 [String syntax], page 491.

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 10.1 [Literals], page 491). (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` will be 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.

20.2.3.45 `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. The query must consist of a single SQL statement. You should not add a terminating semicolon (‘`;`’) or `\g` to the statement.

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 20.2.3.20 [mysql_field_count], page 880.

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.

20.2.3.46 `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.

20.2.3.47 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.

20.2.3.48 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.

20.2.3.49 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.

20.2.3.50 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.

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.

20.2.3.51 `mysql_shutdown()`

`int mysql_shutdown(MYSQL *mysql)`

Description

Asks the database server to shut down. The connected user must have `SHUTDOWN` privileges.

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.

20.2.3.52 `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 Section 21.1 [Error-returns], page 969.

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 20.2.3.12 [mysql_errno], page 874. See Section 20.2.3.13 [mysql_error], page 875. See Section 20.2.7.22 [mysql_stmt_sqlstate], page 939.

20.2.3.53 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()` will return an error when you attempt to connect.

20.2.3.54 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.

20.2.3.55 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).

You don't have to call mysql_store_result() or mysql_use_result() for other queries, but it will not do any harm or cause any notable performance 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 20.2.3.20 [mysql_field_count], page 880.

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 20.2.12.1 [NULL mysql_store_result()], page 947.

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.

20.2.3.56 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 will change. 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.

20.2.3.57 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 will allocate 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 will tie 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 will be returned as part of the result set for your next query. The C API will give 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() will accurately return 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.

20.2.3.58 `mysql_warning_count()`

`unsigned int mysql_warning_count(MYSQL *mysql)`

Description

Returns the number of warnings generated during execution of the previous SQL statement. Available from MySQL 4.1.

Return Values

The warning count.

Errors

None.

20.2.3.59 `mysql_commit()`

`my_bool mysql_commit(MYSQL *mysql)`

Description

Commits the current transaction. Available from MySQL 4.1.

Return Values

Zero if successful. Non-zero if an error occurred.

Errors

None.

20.2.3.60 mysql_rollback()

`my_bool mysql_rollback(MYSQL *mysql)`

Description

Rolls back the current transaction. Available from MySQL 4.1.

Return Values

Zero if successful. Non-zero if an error occurred.

Errors

None.

20.2.3.61 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. Available from MySQL 4.1.

Return Values

Zero if successful. Non-zero if an error occurred.

Errors

None.

20.2.3.62 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. Available from MySQL 4.1.

Return Values

TRUE (1) if more results exist. FALSE (0) if no more results exist.

Note that in most cases one instead call `mysql_next_result()` to test whether more results exist and initiate the next result set if it existed.

See Section 20.2.8 [C API multiple queries], page 942. See Section 20.2.3.63 [mysql_next_result], page 910.

Errors

None.

20.2.3.63 `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. Available from MySQL 4.1.

Note that 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()` for the next query. This means that you can now call `mysql_store_result()`, `mysql_warning_count()`, `mysql_affected_rows()` ... on the connection.

If `mysql_next_result()` returns an error, no other statements will be executed and there is no more results to fetch.

See Section 20.2.8 [C API multiple queries], page 942.

Return Values

0 if successful and there was more results -1 if no more results > 0 if 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.

20.2.4 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. 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 prepare function.

Prepared execution is faster than direct execution for statements executed more than once, primarly 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. Prepared statements also can support input and output binding for multiple query execution.

20.2.5 C API Prepared Statement Data types

Note: The API for prepared statements is still subject to revision. This information is provided for early adopters, but please be aware that the API may change. Some incompatible changes were made in MySQL 4.1.2. See Section 20.2.7 [C API Prepared statement functions], page 916 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.

  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 query 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, though 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 query 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 query 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.) 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()`. If `length` is a null pointer, the protocol assumes that all character and binary data are null-terminated. 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 query parameter. For output, this value will be set to true after you fetch a row if the result value returned from the query is `NULL`.

`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 20.2.9 [C API date handling], page 943.

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	long 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.

20.2.6 C API Prepared Statement Function Overview

Note: The API for prepared statements is still subject to revision. This information is provided for early adopters, but please be aware that the API may change. Some incompatible changes were made in MySQL 4.1.2. See Section 20.2.7 [C API Prepared statement functions], page 916 for details.

The functions available for prepared statement processing are summarized here and described in greater detail in a later section. See Section 20.2.7 [C API Prepared statement functions], page 916.

Function	Description
mysql_stmt_init()	Allocates memory for `MYSQL_STMT` structure and initializes it.
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_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_result_metadata()	Returns prepared statement metadata in the form of a 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_send_long_data()	Sends long data in chunks to server.
mysql_stmt_affected_rows()	Returns the number of rows changes, deleted, or inserted by the last `UPDATE`, `DELETE`, or `INSERT` query.
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_free_result()	Free the resources allocated to the statement handle.
mysql_stmt_num_rows()	Returns total rows from the statement buffered result set.
mysql_stmt_reset()	Reset the statement buffers in the server.
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_sqlstate()	Returns the SQLSTATE error code for the last statement execution.
mysql_stmt_store_result()	Retrieves the complete result set to the client.
mysql_stmt_attr_set()	Sets an attribute for a prepared statement.
mysql_stmt_attr_get()	Get value of an attribute for a prepared statement.

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 query. You can repeat the `mysql_stmt_execute()` by changing parameter values in the respective buffers supplied through `mysql_stmt_bind_param()`.

If the query is a `SELECT` statement or any other query that produces a result set, `mysql_stmt_prepare()` will also return 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()` will automatically return 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 20.2.7.11 [mysql_stmt_send_long_data], page 932.

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 it using `mysql_stmt_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()`. Call `mysql_stmt_prepare()` and pass it a string containing the SQL statement to prepare the statement on the server.
2. If the query 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, query execution will return an error or produce unexpected results.
4. Call `mysql_stmt_execute()` to execute the statement.
5. If the query 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 query 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 meta information if it is a result set oriented query. All syntax and semantics of the query 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 query. If the query 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.

You can get the statement error code, error message, and SQLSTATE value using `mysql_stmt_errno()`, `mysql_stmt_error()`, and `mysql_stmt_sqlstate()`, respectively.

20.2.7 C API Prepared Statement Function Descriptions

To prepare and execute queries, use the functions in the following sections.

Note: The API for prepared statements is still subject to revision. This information is provided for early adopters, but please be aware that the API may change.

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()`

Now 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.

20.2.7.1 `mysql_stmt_init()`

`MYSQL_STMT *mysql_stmt_init(MYSQL *mysql)`

Description

Create a MYSQL_STMT handle.

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.

20.2.7.2 `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 20.2.5 [C API Prepared statement datatypes], page 911.

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 20.2.7.4 [mysql_stmt_execute()], page 919.

20.2.7.3 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.

Note that 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. Otherwise, mysql_stmt_fetch() simply ignores the data fetch. Also, 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 20.2.5 [C API Prepared statement datatypes], page 911.

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 20.2.7.5 [mysql_stmt_fetch()], page 923.

20.2.7.4 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 20.2.7.5 [mysql_stmt_fetch()], page 923.

Return Values

Zero if execution was successful. Non-zero if an error occurred. The error code and message can be obtained by calling mysql_stmt_errno() and mysql_stmt_error().

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_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; it will */
/* be set 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(mysql, 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 */

/* 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_VAR_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): %ld\n", 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): %ld\n", 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/client_test.c'. This file can be obtained from a MySQL source distribution or from the BitKeeper source repository.

20.2.7.5 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.

Note that 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_TIMESTAMP	sizeof(MYSQL_TIME)
MYSQL_TYPE_STRING	data length
MYSQL_TYPE_VAR_STRING	data_length
MYSQL_TYPE_TINY_BLOB	data_length
MYSQL_TYPE_BLOB	data_length
MYSQL_TYPE_MEDIUM_BLOB	data_length
MYSQL_TYPE_LONG_BLOB	data_length

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

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 20.2.7.4 [`mysql_stmt_execute()`], page 919.) 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 */
```

```
/* 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_VAR_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);
}
```

20.2.7.6 mysql_stmt_fetch_column()

int mysql_stmt_fetch_column(MYSQL_STMT *stmt, MYSQL_BIND *bind, unsigned int column, unsigned long offset)

Description

Return Values

Errors

20.2.7.7 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_rseult_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()`.

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 20.2.7.5 [`mysql_stmt_fetch()`], page 923.

20.2.7.8 `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.

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 20.2.7.4 [`mysql_stmt_execute()`], page 919.

20.2.7.9 mysql_stmt_param_metadata()

`MYSQL_RES *mysql_stmt_param_metadata(MYSQL_STMT *stmt)`

To be added.

Description

Return Values

Errors

20.2.7.10 mysql_stmt_prepare()

`int mysql_stmt_prepare(MYSQL_STMT *stmt, const char *query, unsigned long length)`

Description

Prepares the SQL query pointed to by the null-terminated string `query`, and returns a statement handle to be used for further operations on the statement. The query 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), in the select list that names the columns to be returned by a `SELECT` statement), 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 Languange (DML) statements, and not in Data Defination Language (DDL) statements.

The parameter markers must be bound to application variables using `mysql_stmt_bind_param()` before executing the statement.

Return Values

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 is not successful (that is, `mysql_stmt_prepare()` returns `NULL`), the error message can be obtained by calling `mysql_error()`.

Example

For the usage of `mysql_stmt_prepare()`, refer to the Example from Section 20.2.7.4 [mysql_stmt_execute()], page 919.

20.2.7.11 mysql_stmt_send_long_data()

`my_bool mysql_stmt_send_long_data(MYSQL_STMT *stmt, unsigned int parameter_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 will ignore 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 20.2.7.19 [mysql_stmt_reset], page 938.

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",40
{
  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);
}
```

20.2.7.12 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 immediatlely after mysql_stmt_execute() for UPDATE, DELETE, or INSERT statements. For SELECT statements, mysql_stmt_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_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).

Errors

None.

Example

For the usage of mysql_stmt_affected_rows(), refer to the Example from Section 20.2.7.4 [mysql_stmt_execute()], page 919.

20.2.7.13 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.

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 20.2.7.4 [mysql_stmt_execute()], page 919.

20.2.7.14 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()`.

Return Values

None.

Errors

None.

20.2.7.15 `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 Section 21.1 [Error-returns], page 969.

Return Values

An error code value. Zero if no error occurred.

Errors

None.

20.2.7.16 `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.

Return Values

A character string that describes the error. An empty string if no error occurred.

Errors

None.

20.2.7.17 `mysql_stmt_free_result()`

`my_bool mysql_stmt_free_result(MYSQL_STMT *stmt)`

To be added.

Description

Return Values

Errors

20.2.7.18 `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.

Return Values

The number of rows in the result set.

Errors

None.

20.2.7.19 `mysql_stmt_reset()`

`my_bool mysql_stmt_reset(MYSQL_STMT *stmt)`

Description

Reset prepared statement on client and server to state after prepare. For now 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()`.

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.

20.2.7.20 `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()`.

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.

20.2.7.21 `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()`.

Return Values

The current offset of the row cursor.

Errors

None.

20.2.7.22 `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 Section 21.1 [Error-returns], page 969.

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.

20.2.7.23 `mysql_stmt_store_result()`

`int mysql_stmt_store_result(MYSQL_STMT *stmt)`

Description

You must call `mysql_stmt_store_result()` for every query 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 queries, but if you do, it will not harm or cause any notable performance. You can detect whether the query produced a result set by checking if `mysql_stmt_result_metadata()` returns `NULL`. For more information, refer to Section 20.2.7.7 [`mysql_stmt_result_metadata()`], page 929.

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 20.2.7.24 [mysql_stmt_attr_set], page 941.

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.

20.2.7.24 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()`.

Return Values

`0` if ok. `1` if `attr_type` is unknown.

Errors

none

20.2.7.25 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 20.2.7.24 [mysql_stmt_attr_set], page 941.

Return Values

`0` if ok. `1` if `attr_type` is unknown.

Errors

none

20.2.8 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 a 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));
```

20.2.9 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 20.2.5 [C API Prepared statement datatypes], page 911.

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 will 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);
}

/* setup 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);
..
```

20.2.10 C API Threaded Function Descriptions

You need to use the following functions when you want to create a threaded client. See Section 20.2.14 [Threaded clients], page 950.

20.2.10.1 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 will also call `mysql_thread_init()` for this thread.

This is automatically called by `mysql_init()`, `mysql_server_init()` and `mysql_connect()`.

Return Values

None.

20.2.10.2 `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.

20.2.10.3 `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.

20.2.10.4 `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.

20.2.11 C API Embedded Server Function Descriptions

You must use the following functions if you want to allow your application to be linked against the embedded MySQL server library. See Section 20.2.15 [libmysqld], page 951.

If the program is linked with `-lmysqlclient` instead of `-lmysqld`, these functions do nothing. This makes it possible to choose between using the embedded MySQL server and a standalone server without modifying any code.

20.2.11.1 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 program will crash. 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.

The `NULL`-terminated list of strings in `groups` selects which groups in the option files will be active. See Section 4.3.2 [Option files], page 212. For convenience, `groups` may be `NULL`, in which case the `[server]` and `[emedded]` groups will be 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.

20.2.11.2 `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.

20.2.12 Common questions and problems when using the C API

20.2.12.1 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().

20.2.12.2 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 20.2.3.32 [mysql_insert_id()], page 886.
- 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.

20.2.12.3 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 whether a value was stored into an AUTO_INCREMENT column by executing the following code. This also checks whether the query was an INSERT with an AUTO_INCREMENT index:

```
if (mysql_error(&mysql)[0] == 0 &&
    mysql_num_fields(result) == 0 &&
    mysql_insert_id(&mysql) != 0)
{
    used_id = mysql_insert_id(&mysql);
}
```

For more information, see Section 20.2.3.32 [mysql_insert_id()], page 886.

When a new AUTO_INCREMENT value has been generated, you can also obtain it by executing a SELECT LAST_INSERT_ID() statement 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 will not be changed by another client. It will not even be 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 is an explicit value. LAST_INSERT_ID() returns 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().

20.2.12.4 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.

20.2.13 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 20.1.2 [mysql_config], page 857.

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.

20.2.14 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 20.2 [MySQL C API], page 858. 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 20.2.10 [C Thread functions], page 944.

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 will create 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 will create 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 will not 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 will free 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.

20.2.15 libmysqld, the Embedded MySQL Server Library

20.2.15.1 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 will access 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 20.2.11.1 [`mysql_server_init()`], page 946.

20.2.15.2 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.

20.2.15.3 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 OS has nowadays support for 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.

20.2.15.4 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 [Option files], page 212.

- Put common options in the `[server]` section. These will be 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.

20.2.15.5 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.

20.2.15.6 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 will be 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 will
     * initialize 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 will 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
```

20.2.15.7 Licensing the Embedded Server

The MySQL source code is covered by the GNU GPL license (see Appendix G [GPL license], page 1216). One result of this is that any program which includes, by linking with libmysqld, the MySQL source code must be released as free software (under a license compatible with the GPL).

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 Section 1.4.3 [MySQL licenses], page 17.

20.3 MySQL ODBC Support

MySQL provides support for ODBC by means of the MyODBC program. This chapter will teach you how to install MyODBC, and how to use it. Here, you will also find a list of common programs that are known to work with MyODBC.

20.3.1 How to Install MyODBC

MyODBC 2.50 is a 32-bit ODBC 2.50 specification level 0 (with level 1 and level 2 features) driver for connecting an ODBC-aware application to MySQL. MyODBC works on Windows 9x, Me, NT, 2000, and XP, and on most Unix platforms. MyODBC 3.51 is an enhanced version with ODBC 3.5x specification level 1 (complete core API + level 2 features).

MyODBC is Open Source, and you can find the newest version at http://www.mysql.com/downloads/api-my 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 only need to install MyODBC on Windows machines. You only need MyODBC for Unix if you have a program like ColdFusion that is running on the Unix machine and uses ODBC to connect to the databases.

If you want to install MyODBC on a Unix box, you will also need an ODBC manager. MyODBC is known to work with most of the Unix ODBC managers.

To install MyODBC on Windows, you should download the appropriate MyODBC '.zip' file, unpack it with WinZip or some similar program, and execute the 'SETUP.EXE' file.

On Windows, NT, and XP you may get the following error when trying to install MyODBC:

```
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 in this case is that some other program is using ODBC and because of how Windows is designed, you may not in this case be able to install a new ODBC drivers with Microsoft's ODBC setup program. In most cases you can continue by just pressing

`Ignore` to copy the rest of the MyODBC files and the final installation should still work. If this doesn't work, the solution is to restart your computer in "safe mode" (Choose this by pressing F8 just before your machine starts Windows while restarting), install `MyODBC`, and restart to normal mode.

- 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 the access privileges to the MySQL server on the Unix machine. This is set up with the `GRANT` command. See Section 14.5.1.2 [`GRANT`], page 689.
- You must create an ODBC DSN entry as follows:
  - Open the Control Panel on the Windows machine.
  - Double-click the ODBC Data Sources 32-bit icon.
  - Click the tab User DSN.
  - Click the button Add.
  - Select MySQL in the screen Create New Data Source and click the Finish button.
  - The MySQL Driver default configuration screen is shown. See Section 20.3.2 [ODBC administrator], page 958.
- Now start your application and select the ODBC driver with the DSN you specified in the ODBC administrator.

Notice that there are other configuration options on the screen of MySQL (trace, don't prompt on connect, etc) that you can try if you run into problems.

20.3.2 How to Fill in the Various Fields in the ODBC Administrator Program

There are three possibilities for specifying the server name on Windows:

- Use the IP address of the server.
- Add a file '`\windows\lmhosts`' with the following information:

  ```
  ip hostname
  ```

  For example:

  ```
  194.216.84.21 my_hostname
  ```
- Configure the PC to use DNS.

An example of how to fill in the `ODBC setup`:

```
Windows DSN name:   test
Description:        This is my test database
MySQL Database:     test
Server:             194.216.84.21
User:               monty
Password:           my_password
Port:
```

The value for the `Windows DSN name` field is any name that is unique in your Windows ODBC setup.

You don't have to specify values for the `Server`, `User`, `Password`, or `Port` fields in the ODBC setup screen. However, if you do, the values will be used as the defaults later when you attempt to make a connection. You have the option of changing the values at that time.

If the port number is not given, the default port (3306) is used.

If you specify the option `Read options from C:\my.cnf`, the groups `client` and `odbc` will be read from the '`C:\my.cnf`' file. You can use all options that are usable by `mysql_options()`. See Section 20.2.3.40 [`mysql_options()`], page 892.

20.3.3 Connect parameters for MyODBC

One can specify the following parameters for `MyODBC` on the `[Servername]` 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	A integer by which you can specify how `MyODBC` should work. See below.
port	3306	The TCP/IP port to use if `server` is not `localhost`.
stmt		A statement that will be executed when connecting to MySQL
password		The password for the `server` `user` combination.
socket		The socket or Windows pipe to connect to.

The option argument is used to tell `MyODBC` that the client isn't 100% ODBC compliant. On Windows, one normally sets the option flag by toggling the different options on the connection screen but one can also set this in the option argument. The following options are listed in the same order as they appear in the `MyODBC` connect screen:

Bit	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 then 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`'
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	Simulate a ODBC 1.0 driver in some context.
64	Ignore use of database name in 'database.table.column'.
128	Force use of ODBC manager cursors (experimental).
256	Disable the use of extended fetch (experimental).
512	Pad CHAR fields to full column length.

1024	SQLDescribeCol() will return 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 will make 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 bee needed but...)

If you want to have many options, you should add together their values. For example setting option to 12 (4+8) gives you debugging without package limits!

The default '`MYODBC.DLL`' is compiled for optimal performance. If you want to debug `MyODBC` (for example to enable tracing), you should instead use '`MYODBCD.DLL`'. To install this file, copy '`MYODBCD.DLL`' over the installed '`MYODBC.DLL`' file.

20.3.4 How to Report Problems with MyODBC

`MyODBC` has been tested with Access, Admndemo.exe, C++-Builder, Borland Builder 4, Centura Team Developer (formerly Gupta SQL/Windows), ColdFusion (on Solaris and NT with svc pack 5), Crystal Reports, DataJunction, Delphi, ERwin, Excel, iHTML, FileMaker Pro, FoxPro, Notes 4.5/4.6, SBSS, Perl DBD-ODBC, Paradox, Powerbuilder, Powerdesigner 32 bit, VC++, and Visual Basic.

If you know of any other applications that work with `MyODBC`, please send mail to the `myodbc` mailing list about this! See Section 1.7.1.1 [Mailing-list], page 32.

With some programs you may get an error like: `Another user has modifies 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 there isn't one already.
- Add a timestamp column if there isn't one already.
- Only use double float fields. Some programs may fail when they compare single floats.

If these strategies don't help, you should generate a `MyODBC` trace file and try to figure out why things go wrong.

20.3.5 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:

Program **Comment**

Access
To make Access work:

- If you are using Access 2000, you should get and install the newest (version 2.6 or above) Microsoft MDAC (`Microsoft Data Access Components`) from `http://www.microsoft.com/data/`. This will fix the following bug in Access: when you export data to MySQL, the table and column names aren't specified. Another way to around this bug is to upgrade to MyODBC 2.50.33 and MySQL 3.23.x, which together provide a workaround for this bug!

You should also get and apply the Microsoft Jet 4.0 Service Pack 5 (SP5) which can be found here `http://support.microsoft.com/support/kb/articles/Q239/1/14.ASP`. This will fix some cases where columns are marked as `#deleted#` in Access.

Note that 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 above to work around this problem.

- For all Access versions, you should enable the MyODBC option flag `Return matching rows`. For Access 2.0, you should additionally enable `Simulate ODBC 1.0`.
- You should have a timestamp in all tables you want to be able to update. For maximum portability `TIMESTAMP(14)` or simple `TIMESTAMP` is recommended instead of other `TIMESTAMP(X)` variations.
- You should have a primary key in the table. If not, new or updated rows may show up as `#DELETED#`.
- Only use `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 linking a table through MyODBC, which has `BIGINT` as one of the column, then the results will be displayed as `#DELETED`. The work around solution is:
  - Have one more dummy column with `TIMESTAMP` as the data type, preferably `TIMESTAMP(14)`.
  - Check the `'Change BIGINT columns to INT'` in connection options dialog in ODBC DSN Administrator
  - Delete the table link from access and re-create it.

  It still displays the previous records as `#DELETED#`, but newly added/updated records will be 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 `table` data sheet view. Create instead 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 queries that MySQL can't understand. You can fix this by selecting `"Query|SQLSpecific|Pass-Through"` from the Access menu.
- Access on NT will report `BLOB` columns as `OLE OBJECTS`. If you want to have `MEMO` columns instead, you should change the column 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 will try to export this as `TINYINT` instead of `TINYINT UNSIGNED`. This will give you problems if you have values > 127 in the column!

ADO
When you are coding with the ADO API and `MyODBC` you need to put attention in some default properties that aren't supported by the MySQL server. For example, using the `CursorLocation Property` as `adUseServer` will return for the `RecordCount Property` a result of -1. To have the right value, you need to set this property to `adUseClient`, like is showing 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 use the option flag `Return matching rows`.

BDE applications
To get these to work, you should set the option flags `Don't optimize column widths` and `Return matching rows`.

Borland Builder 4
When you start a query you can use the property `Active` or use the method `Open`. Note that `Active` will start by automatically issuing a `SELECT * FROM ...` query that may not be a good thing if your tables are big!

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://www.mysql.com/downloads/api-myodbc.html`

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 will appear 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 grief.

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 plain stupid 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 db with a table containing 2 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 db above.
- Open the Word application.
- Create a blank new documentation.
- Using the tool bar called Database, press the button insert database.
- Press the button Get Data.
- At the right hand of the screen Get Data, press the button Ms Query.
- In the Ms Query create a New Data Source using the DSN file my.
- 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. Set the `Don't optimize column width` option field 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 PRIMARY, though this has not been a problem.

Vision
: You should use the option flag `Return matching rows`.

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` will not work properly. The fix is to set the option `OPTION=16384` in the ODBC connect string or to set the `Change BIGINT columns to INT` option in the MyODBC connect screen. You may also want to set the `Return matching rows` option.

VisualInterDev
: If you get the error `[Microsoft][ODBC Driver Manager] Driver does not support this parameter` the reason may be that you have a `BIGINT` in your result. Try setting the `Change BIGINT columns to INT` option in the MyODBC connect screen.

Visual Objects
: You should use the option flag `Don't optimize column widths`.

20.3.6 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`. With ODBC, you can do something like this (assuming that `auto` is an `AUTO_INCREMENT` field):

```
INSERT INTO foo (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 foo (auto,text) VALUES(NULL,'text');
INSERT INTO foo2 (id,text) VALUES(LAST_INSERT_ID(),'text');
```

See Section 20.2.12.3 [Getting unique ID], page 948.

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_name WHERE auto IS NULL;
```

20.3.7 Reporting Problems with MyODBC

If you encounter difficulties with MyODBC, 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 get a MyODBC log, you need to do the following:

1. Ensure that you are using 'myodbcd.dll' and not 'myodbc.dll'. The easiest way to do this is to get 'myodbcd.dll' from the MyODBC distribution and copy it over the '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 it is a lot faster than 'myodbcd.dll'.
2. Select the "Trace MyODBC" option in the MyODBC connect/configure screen. The log will be written to the file 'C:\myodbc.log'.

   If the trace option is not remembered the next time you visit this screen, it means that you are not using the myodbcd.dll driver. Reread the previous step to verify that you have installed 'myodbcd.dll'.
3. Start your application and try to get it to fail.

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 will help 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 1.7.1.1 [Mailing-list], page 32. 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 make 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://support.mysql.com/pub/mysql/secret/. Only we at MySQL AB will have access to the files you upload, and we will be 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!

20.4 MySQL Java Connectivity (JDBC)

There are 2 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://www.mysql.com/products/connector-j/.
- The Resin JDBC driver, which can be found at http://www.caucho.com/projects/jdbc-mysql/inde

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://www.mysql.com/documentation/`.

20.5 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/`).

20.5.1 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`.

20.6 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 now 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 2.7 [Perl support], page 166.

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/`.
- 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/`.

20.7 MySQL C++ API

`MySQL++` is the MySQL API for C++. More information can be found at `http://www.mysql.com/products/mysql++/`.

20.7.1 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 will run into problems if you try to use the default `libmysql.dll` libraries (that was compiled with VC++) with Borland C++. You can do one of the following to avoid this problem.

- You can use the static MySQL libraries for Borland C++ that you can find on `http://www.mysql.com/downloads/os-win32.html`.
- Only call `mysql_init()` with `NULL` as an argument, not a pre-allocated `MYSQL` structure.

20.8 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/`.

20.9 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/`.

20.10 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`.

21 Error Handling in MySQL

This chapter lists the errors that MySQL can return.

21.1 Error Returns

Following are error codes that may appear when you call MySQL from any host language.

The `Name` and `Error Code` columns correspond to definitions in the '`include/mysqld_error.h`' MySQL source file.

The `SQLSTATE` column corresponds to definitions in the '`include/sql_state.h`' MySQL source file.

The `SQLSTATE` error code is displayed only if you use MySQL version 4.1 and up. `SQLSTATE` codes were added for compatibility with X/Open, ANSI, and ODBC behavior.

A suggested text for each error code can be found in the '`share/english/errmsg.txt`' error-message file.

Because updates are frequent, it is possible that these files will contain additional error codes.

Name	**Error Code**	**SQLSTATE**
ER_HASHCHK	1000	HY000
ER_NISAMCHK	1001	HY000
ER_NO	1002	HY000
ER_YES	1003	HY000
ER_CANT_CREATE_FILE	1004	HY000
ER_CANT_CREATE_TABLE	1005	HY000
ER_CANT_CREATE_DB	1006	HY000
ER_DB_CREATE_EXISTS	1007	HY000
ER_DB_DROP_EXISTS	1008	HY000
ER_DB_DROP_DELETE	1009	HY000
ER_DB_DROP_RMDIR	1010	HY000
ER_CANT_DELETE_FILE	1011	HY000
ER_CANT_FIND_SYSTEM_REC	1012	HY000
ER_CANT_GET_STAT	1013	HY000
ER_CANT_GET_WD	1014	HY000
ER_CANT_LOCK	1015	HY000
ER_CANT_OPEN_FILE	1016	HY000
ER_FILE_NOT_FOUND	1017	HY000
ER_CANT_READ_DIR	1018	HY000
ER_CANT_SET_WD	1019	HY000
ER_CHECKREAD	1020	HY000
ER_DISK_FULL	1021	HY000
ER_DUP_KEY	1022	23000
ER_ERROR_ON_CLOSE	1023	HY000
ER_ERROR_ON_READ	1024	HY000

ER_ERROR_ON_RENAME	1025	HY000
ER_ERROR_ON_WRITE	1026	HY000
ER_FILE_USED	1027	HY000
ER_FILSORT_ABORT	1028	HY000
ER_FORM_NOT_FOUND	1029	HY000
ER_GET_ERRNO	1030	HY000
ER_ILLEGAL_HA	1031	HY000
ER_KEY_NOT_FOUND	1032	HY000
ER_NOT_FORM_FILE	1033	HY000
ER_NOT_KEYFILE	1034	HY000
ER_OLD_KEYFILE	1035	HY000
ER_OPEN_AS_READONLY	1036	HY000
ER_OUTOFMEMORY	1037	HY001
ER_OUT_OF_SORTMEMORY	1038	HY001
ER_UNEXPECTED_EOF	1039	HY000
ER_CON_COUNT_ERROR	1040	08004
ER_OUT_OF_RESOURCES	1041	08004
ER_BAD_HOST_ERROR	1042	08S01
ER_HANDSHAKE_ERROR	1043	08S01
ER_DBACCESS_DENIED_ERROR	1044	42000
ER_ACCESS_DENIED_ERROR	1045	42000
ER_NO_DB_ERROR	1046	42000
ER_UNKNOWN_COM_ERROR	1047	08S01
ER_BAD_NULL_ERROR	1048	23000
ER_BAD_DB_ERROR	1049	42000
ER_TABLE_EXISTS_ERROR	1050	42S01
ER_BAD_TABLE_ERROR	1051	42S02
ER_NON_UNIQ_ERROR	1052	23000
ER_SERVER_SHUTDOWN	1053	08S01
ER_BAD_FIELD_ERROR	1054	42S22
ER_WRONG_FIELD_WITH_GROUP	1055	42000
ER_WRONG_GROUP_FIELD	1056	42000
ER_WRONG_SUM_SELECT	1057	42000
ER_WRONG_VALUE_COUNT	1058	21S01
ER_TOO_LONG_IDENT	1059	42000
ER_DUP_FIELDNAME	1060	42S21
ER_DUP_KEYNAME	1061	42000
ER_DUP_ENTRY	1062	23000
ER_WRONG_FIELD_SPEC	1063	42000
ER_PARSE_ERROR	1064	42000
ER_EMPTY_QUERY	1065	42000
ER_NONUNIQ_TABLE	1066	42000
ER_INVALID_DEFAULT	1067	42000
ER_MULTIPLE_PRI_KEY	1068	42000
ER_TOO_MANY_KEYS	1069	42000
ER_TOO_MANY_KEY_PARTS	1070	42000
ER_TOO_LONG_KEY	1071	42000

ER_KEY_COLUMN_DOES_NOT_EXITS	1072	42000
ER_BLOB_USED_AS_KEY	1073	42000
ER_TOO_BIG_FIELDLENGTH	1074	42000
ER_WRONG_AUTO_KEY	1075	42000
ER_READY	1076	00000
ER_NORMAL_SHUTDOWN	1077	00000
ER_GOT_SIGNAL	1078	00000
ER_SHUTDOWN_COMPLETE	1079	00000
ER_FORCING_CLOSE	1080	08S01
ER_IPSOCK_ERROR	1081	08S01
ER_NO_SUCH_INDEX	1082	42S12
ER_WRONG_FIELD_TERMINATORS	1083	42000
ER_BLOBS_AND_NO_TERMINATED	1084	42000
ER_TEXTFILE_NOT_READABLE	1085	HY000
ER_FILE_EXISTS_ERROR	1086	HY000
ER_LOAD_INFO	1087	HY000
ER_ALTER_INFO	1088	HY000
ER_WRONG_SUB_KEY	1089	HY000
ER_CANT_REMOVE_ALL_FIELDS	1090	42000
ER_CANT_DROP_FIELD_OR_KEY	1091	42000
ER_INSERT_INFO	1092	HY000
ER_UPDATE_TABLE_USED	1093	HY000
ER_NO_SUCH_THREAD	1094	HY000
ER_KILL_DENIED_ERROR	1095	HY000
ER_NO_TABLES_USED	1096	HY000
ER_TOO_BIG_SET	1097	HY000
ER_NO_UNIQUE_LOGFILE	1098	HY000
ER_TABLE_NOT_LOCKED_FOR_WRITE	1099	HY000
ER_TABLE_NOT_LOCKED	1100	HY000
ER_BLOB_CANT_HAVE_DEFAULT	1101	42000
ER_WRONG_DB_NAME	1102	42000
ER_WRONG_TABLE_NAME	1103	42000
ER_TOO_BIG_SELECT	1104	42000
ER_UNKNOWN_ERROR	1105	HY000
ER_UNKNOWN_PROCEDURE	1106	42000
ER_WRONG_PARAMCOUNT_TO_PROCEDURE	1107	42000
ER_WRONG_PARAMETERS_TO_PROCEDURE	1108	HY000
ER_UNKNOWN_TABLE	1109	42S02
ER_FIELD_SPECIFIED_TWICE	1110	42000
ER_INVALID_GROUP_FUNC_USE	1111	42000
ER_UNSUPPORTED_EXTENSION	1112	42000
ER_TABLE_MUST_HAVE_COLUMNS	1113	42000
ER_RECORD_FILE_FULL	1114	HY000
ER_UNKNOWN_CHARACTER_SET	1115	42000
ER_TOO_MANY_TABLES	1116	HY000
ER_TOO_MANY_FIELDS	1117	HY000
ER_TOO_BIG_ROWSIZE	1118	42000

ER_STACK_OVERRUN	1119	HY000
ER_WRONG_OUTER_JOIN	1120	42000
ER_NULL_COLUMN_IN_INDEX	1121	42000
ER_CANT_FIND_UDF	1122	HY000
ER_CANT_INITIALIZE_UDF	1123	HY000
ER_UDF_NO_PATHS	1124	HY000
ER_UDF_EXISTS	1125	HY000
ER_CANT_OPEN_LIBRARY	1126	HY000
ER_CANT_FIND_DL_ENTRY	1127	HY000
ER_FUNCTION_NOT_DEFINED	1128	HY000
ER_HOST_IS_BLOCKED	1129	HY000
ER_HOST_NOT_PRIVILEGED	1130	HY000
ER_PASSWORD_ANONYMOUS_USER	1131	42000
ER_PASSWORD_NOT_ALLOWED	1132	42000
ER_PASSWORD_NO_MATCH	1133	42000
ER_UPDATE_INFO	1134	HY000
ER_CANT_CREATE_THREAD	1135	HY000
ER_WRONG_VALUE_COUNT_ON_ROW	1136	21S01
ER_CANT_REOPEN_TABLE	1137	HY000
ER_INVALID_USE_OF_NULL	1138	42000
ER_REGEXP_ERROR	1139	42000
ER_MIX_OF_GROUP_FUNC_AND_FIELDS	1140	42000
ER_NONEXISTING_GRANT	1141	42000
ER_TABLEACCESS_DENIED_ERROR	1142	42000
ER_COLUMNACCESS_DENIED_ERROR	1143	42000
ER_ILLEGAL_GRANT_FOR_TABLE	1144	42000
ER_GRANT_WRONG_HOST_OR_USER	1145	42000
ER_NO_SUCH_TABLE	1146	42S02
ER_NONEXISTING_TABLE_GRANT	1147	42000
ER_NOT_ALLOWED_COMMAND	1148	42000
ER_SYNTAX_ERROR	1149	42000
ER_DELAYED_CANT_CHANGE_LOCK	1150	HY000
ER_TOO_MANY_DELAYED_THREADS	1151	HY000
ER_ABORTING_CONNECTION	1152	08S01
ER_NET_PACKET_TOO_LARGE	1153	08S01
ER_NET_READ_ERROR_FROM_PIPE	1154	08S01
ER_NET_FCNTL_ERROR	1155	08S01
ER_NET_PACKETS_OUT_OF_ORDER	1156	08S01
ER_NET_UNCOMPRESS_ERROR	1157	08S01
ER_NET_READ_ERROR	1158	08S01
ER_NET_READ_INTERRUPTED	1159	08S01
ER_NET_ERROR_ON_WRITE	1160	08S01
ER_NET_WRITE_INTERRUPTED	1161	08S01
ER_TOO_LONG_STRING	1162	42000
ER_TABLE_CANT_HANDLE_BLOB	1163	42000
ER_TABLE_CANT_HANDLE_AUTO_INCREMENT	1164	42000
ER_DELAYED_INSERT_TABLE_LOCKED	1165	HY000

ER_WRONG_COLUMN_NAME	1166	42000
ER_WRONG_KEY_COLUMN	1167	42000
ER_WRONG_MRG_TABLE	1168	HY000
ER_DUP_UNIQUE	1169	23000
ER_BLOB_KEY_WITHOUT_LENGTH	1170	42000
ER_PRIMARY_CANT_HAVE_NULL	1171	42000
ER_TOO_MANY_ROWS	1172	42000
ER_REQUIRES_PRIMARY_KEY	1173	42000
ER_NO_RAID_COMPILED	1174	HY000
ER_UPDATE_WITHOUT_KEY_IN_SAFE_MODE	1175	HY000
ER_KEY_DOES_NOT_EXITS	1176	HY000
ER_CHECK_NO_SUCH_TABLE	1177	42000
ER_CHECK_NOT_IMPLEMENTED	1178	42000
ER_CANT_DO_THIS_DURING_AN_TRANSACTION	1179	25000
ER_ERROR_DURING_COMMIT	1180	HY000
ER_ERROR_DURING_ROLLBACK	1181	HY000
ER_ERROR_DURING_FLUSH_LOGS	1182	HY000
ER_ERROR_DURING_CHECKPOINT	1183	HY000
ER_NEW_ABORTING_CONNECTION	1184	08S01
ER_DUMP_NOT_IMPLEMENTED	1185	HY000
ER_FLUSH_MASTER_BINLOG_CLOSED	1186	HY000
ER_INDEX_REBUILD	1187	HY000
ER_MASTER	1188	HY000
ER_MASTER_NET_READ	1189	08S01
ER_MASTER_NET_WRITE	1190	08S01
ER_FT_MATCHING_KEY_NOT_FOUND	1191	HY000
ER_LOCK_OR_ACTIVE_TRANSACTION	1192	HY000
ER_UNKNOWN_SYSTEM_VARIABLE	1193	HY000
ER_CRASHED_ON_USAGE	1194	HY000
ER_CRASHED_ON_REPAIR	1195	HY000
ER_WARNING_NOT_COMPLETE_ROLLBACK	1196	HY000
ER_TRANS_CACHE_FULL	1197	HY000
ER_SLAVE_MUST_STOP	1198	HY000
ER_SLAVE_NOT_RUNNING	1199	HY000
ER_BAD_SLAVE	1200	HY000
ER_MASTER_INFO	1201	HY000
ER_SLAVE_THREAD	1202	HY000
ER_TOO_MANY_USER_CONNECTIONS	1203	42000
ER_SET_CONSTANTS_ONLY	1204	HY000
ER_LOCK_WAIT_TIMEOUT	1205	HY000
ER_LOCK_TABLE_FULL	1206	HY000
ER_READ_ONLY_TRANSACTION	1207	25000
ER_DROP_DB_WITH_READ_LOCK	1208	HY000
ER_CREATE_DB_WITH_READ_LOCK	1209	HY000
ER_WRONG_ARGUMENTS	1210	HY000
ER_NO_PERMISSION_TO_CREATE_USER	1211	42000
ER_UNION_TABLES_IN_DIFFERENT_DIR	1212	HY000

ER_LOCK_DEADLOCK	1213	40001
ER_TABLE_CANT_HANDLE_FULLTEXT	1214	HY000
ER_CANNOT_ADD_FOREIGN	1215	HY000
ER_NO_REFERENCED_ROW	1216	23000
ER_ROW_IS_REFERENCED	1217	23000
ER_CONNECT_TO_MASTER	1218	08S01
ER_QUERY_ON_MASTER	1219	HY000
ER_ERROR_WHEN_EXECUTING_COMMAND	1220	HY000
ER_WRONG_USAGE	1221	HY000
ER_WRONG_NUMBER_OF_COLUMNS_IN_SELECT	1222	21000
ER_CANT_UPDATE_WITH_READLOCK	1223	HY000
ER_MIXING_NOT_ALLOWED	1224	HY000
ER_DUP_ARGUMENT	1225	HY000
ER_USER_LIMIT_REACHED	1226	42000
ER_SPECIFIC_ACCESS_DENIED_ERROR	1227	HY000
ER_LOCAL_VARIABLE	1228	HY000
ER_GLOBAL_VARIABLE	1229	HY000
ER_NO_DEFAULT	1230	42000
ER_WRONG_VALUE_FOR_VAR	1231	42000
ER_WRONG_TYPE_FOR_VAR	1232	42000
ER_VAR_CANT_BE_READ	1233	HY000
ER_CANT_USE_OPTION_HERE	1234	42000
ER_NOT_SUPPORTED_YET	1235	42000
ER_MASTER_FATAL_ERROR_READING_BINLOG	1236	HY000
ER_SLAVE_IGNORED_TABLE	1237	HY000
ER_WRONG_FK_DEF	1238	42000
ER_KEY_REF_DO_NOT_MATCH_TABLE_REF	1239	HY000
ER_CARDINALITY_COL	1240	21000
ER_SUBSELECT_NO_1_ROW	1241	21000
ER_UNKNOWN_STMT_HANDLER	1242	HY000
ER_CORRUPT_HELP_DB	1243	HY000
ER_CYCLIC_REFERENCE	1244	HY000
ER_AUTO_CONVERT	1245	HY000
ER_ILLEGAL_REFERENCE	1246	42S22
ER_DERIVED_MUST_HAVE_ALIAS	1247	42000
ER_SELECT_REDUCED	1248	01000
ER_TABLENAME_NOT_ALLOWED_HERE	1249	42000
ER_NOT_SUPPORTED_AUTH_MODE	1250	08004
ER_SPATIAL_CANT_HAVE_NULL	1251	42000
ER_COLLATION_CHARSET_MISMATCH	1252	42000
ER_SLAVE_WAS_RUNNING	1253	HY000
ER_SLAVE_WAS_NOT_RUNNING	1254	HY000
ER_TOO_BIG_FOR_UNCOMPRESS	1255	HY000
ER_ZLIB_Z_MEM_ERROR	1256	HY000
ER_ZLIB_Z_BUF_ERROR	1257	HY000
ER_ZLIB_Z_DATA_ERROR	1258	HY000
ER_CUT_VALUE_GROUP_CONCAT	1259	HY000

ER_WARN_TOO_FEW_RECORDS	1260	01000
ER_WARN_TOO_MANY_RECORDS	1261	01000
ER_WARN_NULL_TO_NOTNULL	1262	01000
ER_WARN_DATA_OUT_OF_RANGE	1263	01000
ER_WARN_DATA_TRUNCATED	1264	01000
ER_WARN_USING_OTHER_HANDLER	1265	01000
ER_CANT_AGGREGATE_COLLATIONS	1266	42000
ER_DROP_USER	1267	42000
ER_REVOKE_GRANTS	1268	42000
ER_CANT_AGGREGATE_3COLLATIONS	1269	42000
ER_CANT_AGGREGATE_NCOLLATIONS	1270	42000
ER_VARIABLE_IS_NOT_STRUCT	1271	HY000
ER_UNKNOWN_COLLATION	1272	HY000
ER_SLAVE_IGNORED_SSL_PARAMS	1273	HY000
ER_SERVER_IS_IN_SECURE_AUTH_MODE	1274	HY000
ER_WARN_FIELD_RESOLVED	1275	HY000
ER_BAD_SLAVE_UNTIL_COND	1276	HY000
ER_MISSING_SKIP_SLAVE	1277	HY000
ER_UNTIL_COND_IGNORED	1278	HY000

21.2 Error Messages

Following are error messages that may appear when you call MySQL from any host language.

Error Code	**Error Message**
1000	hashchk
1001	isamchk
1002	NO
1003	YES
1004	Can't create file '%s' (errno: %d)
1005	Can't create table '%s' (errno: %d)
1006	Can't create database '%s' (errno: %d)
1007	Can't create database '%s'; database exists
1008	Can't drop database '%s'; database doesn't exist
1009	Error dropping database (can't delete '%s', errno: %d)
1010	Error dropping database (can't rmdir '%s', errno: %d)
1011	Error on delete of '%s' (errno: %d)
1012	Can't read record in system table
1013	Can't get status of '%s' (errno: %d)
1014	Can't get working directory (errno: %d)
1015	Can't lock file (errno: %d)
1016	Can't open file: '%s' (errno: %d)
1017	Can't find file: '%s' (errno: %d)
1018	Can't read dir of '%s' (errno: %d)
1019	Can't change dir to '%s' (errno: %d)

1020	Record has changed since last read in table '%s'
1021	Disk full (%s). Waiting for someone to free some space...
1022	Can't write, duplicate key in table '%s'
1023	Error on close of '%s' (errno: %d)
1024	Error reading file '%s' (errno: %d)
1025	Error on rename of '%s' to '%s' (errno: %d)
1026	Error writing file '%s' (errno: %d)
1027	'%s' is locked against change
1028	Sort aborted
1029	View '%s' doesn't exist for '%s'
1030	Got error %d from storage engine
1031	Table storage engine for '%s' doesn't have this option
1032	Can't find record in '%s'
1033	Incorrect information in file: '%s'
1034	Incorrect key file for table: '%s'; try to repair it
1035	Old key file for table '%s'; repair it!
1036	Table '%s' is read only
1037	Out of memory. Restart daemon and try again (needed %d bytes)
1038	Out of sort memory. Increase daemon sort buffer size
1039	Unexpected eof found when reading file '%s' (errno: %d)
1040	Too many connections
1041	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
1042	Can't get hostname for your address
1043	Bad handshake
1044	Access denied for user: '%s'@'%s' to database '%s'
1045	Access denied for user: '%s'@'%s' (Using password: %s)
1046	No Database Selected
1047	Unknown command
1048	Column '%s' cannot be null
1049	Unknown database '%s'
1050	Table '%s' already exists
1051	Unknown table '%s'
1052	Column: '%s' in %s is ambiguous
1053	Server shutdown in progress
1054	Unknown column '%s' in '%s'
1055	'%s' isn't in GROUP BY
1056	Can't group on '%s'
1057	Statement has sum functions and columns in same statement
1058	Column count doesn't match value count
1059	Identifier name '%s' is too long
1060	Duplicate column name '%s'
1061	Duplicate key name '%s'
1062	Duplicate entry '%s' for key %d
1063	Incorrect column specifier for column '%s'
1064	%s near '%s' at line %d

1065	Query was empty
1066	Not unique table/alias: '%s'
1067	Invalid default value for '%s'
1068	Multiple primary key defined
1069	Too many keys specified. Max %d keys allowed
1070	Too many key parts specified. Max %d parts allowed
1071	Specified key was too long. Max key length is %d
1072	Key column '%s' doesn't exist in table
1073	BLOB column '%s' can't be used in key specification with the used table type
1074	Too big column length for column '%s' (max = %d). Use BLOB instead
1075	Incorrect table definition; There can only be one auto column and it must be defined as a key
1076	%s: ready for connections. Version: '%s' socket: '%s' port: %d
1077	%s: Normal shutdown
1078	%s: Got signal %d. Aborting!
1079	%s: Shutdown Complete
1080	%s: Forcing close of thread %ld user: '%s'
1081	Can't create IP socket
1082	Table '%s' has no index like the one used in CREATE INDEX. Recreate the table
1083	Field separator argument is not what is expected. Check the manual
1084	You can't use fixed rowlength with BLOBs. Please use 'fields terminated by'
1085	The file '%s' must be in the database directory or be readable by all
1086	File '%s' already exists
1087	Records: %ld Deleted: %ld Skipped: %ld Warnings: %ld
1088	Records: %ld Duplicates: %ld
1089	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
1090	You can't delete all columns with ALTER TABLE. Use DROP TABLE instead
1091	Can't DROP '%s'. Check that column/key exists
1092	Records: %ld Duplicates: %ld Warnings: %ld
1093	You can't specify target table '%s' for update in FROM clause
1094	Unknown thread id: %lu
1095	You are not owner of thread %lu
1096	No tables used
1097	Too many strings for column %s and SET
1098	Can't generate a unique log-filename %s.(1-999)
1099	Table '%s' was locked with a READ lock and can't be updated
1100	Table '%s' was not locked with LOCK TABLES
1101	BLOB/TEXT column '%s' can't have a default value
1102	Incorrect database name '%s'
1103	Incorrect table name '%s'

1104	The SELECT would examine more rows than MAX_JOIN_SIZE. Check your WHERE and use SET SQL_BIG_SELECTS=1 or SET SQL_MAX_JOIN_SIZE=# if the SELECT is ok
1105	Unknown error
1106	Unknown procedure '%s'
1107	Incorrect parameter count to procedure '%s'
1108	Incorrect parameters to procedure '%s'
1109	Unknown table '%s' in %s
1110	Column '%s' specified twice
1111	Invalid use of group function
1112	Table '%s' uses an extension that doesn't exist in this MySQL version
1113	A table must have at least 1 column
1114	The table '%s' is full
1115	Unknown character set: '%s'
1116	Too many tables. MySQL can only use %d tables in a join
1117	Too many columns
1118	Too big row size. The maximum row size for the used table type, not counting BLOBs, is %ld. You have to change some fields to TEXT or BLOBs
1119	Thread stack overrun: Used: %ld of a %ld stack. Use 'mysqld -O thread_stack=#' to specify a bigger stack if needed
1120	Cross dependency found in OUTER JOIN. Examine your ON conditions
1121	Column '%s' is used with UNIQUE or INDEX but is not defined as NOT NULL
1122	Can't load function '%s'
1123	Can't initialize function '%s'; %s
1124	No paths allowed for shared library
1125	Function '%s' already exist
1126	Can't open shared library '%s' (errno: %d %s)
1127	Can't find function '%s' in library'
1128	Function '%s' is not defined
1129	Host '%s' is blocked because of many connection errors. Unblock with 'mysqladmin flush-hosts'
1130	Host '%s' is not allowed to connect to this MySQL server
1131	You are using MySQL as an anonymous users and anonymous users are not allowed to change passwords
1132	You must have privileges to update tables in the `mysql` database to be able to change passwords for others
1133	Can't find any matching row in the user table
1134	Rows matched: %ld Changed: %ld Warnings: %ld
1135	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
1136	Column count doesn't match value count at row %ld
1137	Can't reopen table: '%s'
1138	Invalid use of NULL value
1139	Got error '%s' from regexp
1140	Mixing of GROUP columns (MIN(),MAX(),COUNT()...) with no GROUP columns is illegal if there is no GROUP BY clause

1141	There is no such grant defined for user '%s' on host '%s'
1142	%s command denied to user: '%s'@'%s' for table '%s'
1143	%s command denied to user: '%s'@'%s' for column '%s' in table '%s'
1144	Illegal GRANT/REVOKE command. Please consult the manual which privileges can be used
1145	The host or user argument to GRANT is too long
1146	Table '%s.%s' doesn't exist
1147	There is no such grant defined for user '%s' on host '%s' on table '%s'
1148	The used command is not allowed with this MySQL version
1149	You have an error in your SQL syntax. Check the manual that corresponds to your MySQL server version for the right syntax to use
1150	Delayed insert thread couldn't get requested lock for table %s
1151	Too many delayed threads in use
1152	Aborted connection %ld to db: '%s' user: '%s' (%s)
1153	Got a packet bigger than 'max_allowed_packet'
1154	Got a read error from the connection pipe
1155	Got an error from fcntl()
1156	Got packets out of order
1157	Couldn't uncompress communication packet
1158	Got an error reading communication packets
1159	Got timeout reading communication packets
1160	Got an error writing communication packets
1161	Got timeout writing communication packets
1162	Result string is longer than max_allowed_packet
1163	The used table type doesn't support BLOB/TEXT columns
1164	The used table type doesn't support AUTO_INCREMENT columns
1165	INSERT DELAYED can't be used with table '%s' because it is locked with LOCK TABLES
1166	Incorrect column name '%s'
1167	The used storage engine can't index column '%s'
1168	All tables in the MERGE table are not identically defined
1169	Can't write, because of unique constraint, to table '%s'
1170	BLOB/TEXT column '%s' used in key specification without a key length
1171	All parts of a PRIMARY KEY must be NOT NULL; If you need NULL in a key, use UNIQUE instead
1172	Result consisted of more than one row
1173	This table type requires a primary key
1174	This version of MySQL is not compiled with RAID support
1175	You are using safe update mode and you tried to update a table without a WHERE that uses a KEY column
1176	Key '%s' doesn't exist in table '%s'
1177	Can't open table
1178	The storage engine for the table doesn't support %s
1179	You are not allowed to execute this command in a transaction
1180	Got error %d during COMMIT
1181	Got error %d during ROLLBACK
1182	Got error %d during FLUSH_LOGS

1183	Got error %d during CHECKPOINT
1184	Aborted connection %ld to db: '%s' user: '%s' host: '%s' (%s)
1185	The storage engine for the table does not support binary table dump
1186	Binlog closed, cannot RESET MASTER
1187	Failed rebuilding the index of dumped table '%s'
1188	Error from master: '%s'
1189	Net error reading from master
1190	Net error writing to master
1191	Can't find FULLTEXT index matching the column list
1192	Can't execute the given command because you have active locked tables or an active transaction
1193	Unknown system variable '%s'
1194	Table '%s' is marked as crashed and should be repaired
1195	Table '%s' is marked as crashed and last (automatic?) repair failed
1196	Some non-transactional changed tables couldn't be rolled back
1197	Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage. Increase this mysqld variable and try again
1198	This operation cannot be performed with a running slave, run STOP SLAVE first
1199	This operation requires a running slave, configure slave and do START SLAVE
1200	The server is not configured as slave, fix in config file or with CHANGE MASTER TO
1201	Could not initialize master info structure, more error messages can be found in the MySQL error log
1202	Could not create slave thread, check system resources
1203	User %s has already more than 'max_user_connections' active connections
1204	You may only use constant expressions with SET
1205	Lock wait timeout exceeded; Try restarting transaction
1206	The total number of locks exceeds the lock table size
1207	Update locks cannot be acquired during a READ UNCOMMITTED transaction
1208	DROP DATABASE not allowed while thread is holding global read lock
1209	CREATE DATABASE not allowed while thread is holding global read lock
1210	Wrong arguments to %s
1211	'%s'@'%s' is not allowed to create new users
1212	Incorrect table definition; all MERGE tables must be in the same database
1213	Deadlock found when trying to get lock; Try restarting transaction
1214	The used table type doesn't support FULLTEXT indexes
1215	Cannot add foreign key constraint
1216	Cannot add or update a child row: a foreign key constraint fails
1217	Cannot delete or update a parent row: a foreign key constraint fails
1218	Error connecting to master: %s
1219	Error running query on master: %s
1220	Error when executing command %s: %s

1221	Wrong usage of %s and %s
1222	The used SELECT statements have a different number of columns
1223	Can't execute the query because you have a conflicting read lock
1224	Mixing of transactional and non-transactional tables is disabled
1225	Option '%s' used twice in statement
1226	User '%s' has exceeded the '%s' resource (current value: %ld)
1227	Access denied. You need the %s privilege for this operation
1228	Variable '%s' is a LOCAL variable and can't be used with SET GLOBAL
1229	Variable '%s' is a GLOBAL variable and should be set with SET GLOBAL
1230	Variable '%s' doesn't have a default value
1231	Variable '%s' can't be set to the value of '%s'
1232	Wrong argument type to variable '%s'
1233	Variable '%s' can only be set, not read
1234	Wrong usage/placement of '%s'
1235	This version of MySQL doesn't yet support '%s'
1236	Got fatal error %d: '%s' from master when reading data from binary log
1237	Slave SQL thread ignored the query because of replicate-*-table rules
1238	Wrong foreign key definition for '%s': %s
1239	Key reference and table reference doesn't match
1240	Operand should contain %d column(s)
1241	Subquery returns more than 1 row
1242	Unknown prepared statement handler (%ld) given to %s
1243	Help database is corrupt or does not exist
1244	Cyclic reference on subqueries
1245	Converting column '%s' from %s to %s
1246	Reference '%s' not supported (%s)
1247	Every derived table must have it's own alias
1248	Select %u was reduced during optimisation
1249	Table '%s' from one of SELECT's can not be used in %s
1250	Client does not support authentication protocol requested by server; consider upgrading MySQL client
1251	All parts of a SPATIAL KEY must be NOT NULL
1252	COLLATION '%s' is not valid for CHARACTER SET '%s'
1253	Slave is already running
1254	Slave has already been stopped
1255	Too big size of uncompressed data. The maximum size is %d. (probably, length of uncompressed data was corrupted)
1256	ZLIB: Not enough memory available for zlib
1257	ZLIB: Not enough room in the output buffer for zlib (probably, length of uncompressed data was corrupted)
1258	ZLIB: Input data was corrupted for zlib
1259	%d line(s) was(were) cut by group_concat()
1260	Row %ld doesn't contain data for all columns
1261	Row %ld was truncated; It contained more data than there where input columns

1262	Data truncated, NULL supplied to NOT NULL column '%s' at row %ld
1263	Data truncated, out of range for column '%s' at row %ld
1264	Data truncated for column '%s' at row %ld
1265	Using storage engine %s for table '%s'
1266	Illegal mix of collations (%s,%s) and (%s,%s) for operation '%s'
1267	Can't drop one or more of the requested users
1268	Can't revoke all privileges, grant for one or more of the requested users
1269	Illegal mix of collations (%s,%s), (%s,%s), (%s,%s) for operation '%s'
1270	Illegal mix of collations for operation '%s'
1271	Variable '%s' is not a variable component (Can't be used as XXXX.variable_name)
1272	Unknown collation: '%s'
1273	SSL parameters in CHANGE MASTER are ignored because this MySQL slave was compiled without SSL support. They can be used later when MySQL slave with SSL will be started.
1274	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
1275	Field or reference '%s%s%s%s%s' of SELECT #%d was resolved in SELECT #%d
1276	Wrong parameter or combination of parameters for START SLAVE UNTIL
1277	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
1278	SQL thread is not to be started so UNTIL options are ignored
1279	Incorrect index name '%s'
1280	Incorrect catalog name '%s'
1281	Query cache failed to set size %lu, new query cache size is %lu
1282	Column '%s' cannot be part of FULLTEXT index
1283	Unknown key cache '%s'
1284	MySQL is started in –skip-name-resolve mode. You need to restart it without this switch for this grant to work
1285	Unknown table engine '%s'
1286	'%s' is deprecated. Use '%s' instead.
1287	The target table %s of the %s is not updatable.
1288	The '%s' feature was disabled. You need MySQL built with '%s' define to have it working
1289	MySQL is started in –skip-grant-tables mode. You can't use this command
1290	Can't create a %s from within another stored routine
1291	%s %s already exists
1292	%s %s does not exist
1293	Failed to DROP %s %s
1294	Failed to CREATE %s %s
1295	%s with no matching label: %s
1296	Redefining label %s
1297	End-label %s without match

1298	Referring to uninitialized variable %s
1299	SELECT in a stored procedure must have INTO
1300	RETURN is only allowed in a FUNCTION
1301	Statements like SELECT, INSERT, UPDATE (and others) are not allowed in a FUNCTION
1302	The update log is deprecated and replaced by the binary log. SET SQL_LOG_UPDATE has been ignored.
1303	The update log is deprecated and replaced by the binary log. SET SQL_LOG_UPDATE has been translated to SET SQL_LOG_BIN.
1304	Query execution was interrupted
1305	Wrong number of arguments for %s %s, expected %u, got %u
1306	Undefined CONDITION: %s
1307	No RETURN found in FUNCTION %s
1308	FUNCTION %s ended without RETURN
1309	Cursor statement must be a SELECT
1310	Cursor SELECT must not have INTO
1311	Undefined CURSOR: %s
1312	Cursor is already open
1313	Cursor is not open
1314	Undeclared variable: %s
1315	Wrong number of FETCH variables
1316	No data to FETCH
1317	Duplicate parameter: %s
1318	Duplicate variable: %s
1319	Duplicate condition: %s
1320	Duplicate cursor: %s
1321	Failed to ALTER %s %s
1322	Subselect value not supported

22 Extending MySQL

22.1 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 2.3.3 [Installing source tree], page 104. 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 1.7.1.1 [Mailing-list], page 32. 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!

22.1.1 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 will be started to read and apply updates from the master.

`mysqladmin processlist` only shows the connection, `INSERT DELAYED`, and replication threads.

22.1.2 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 Windows 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 will ensure that all future MySQL releases will work well with your applications.

22.1.2.1 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 22.1.2.3 [Reporting mysqltest bugs], page 986.

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.

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.

22.1.2.2 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 `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 will 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 `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 on the line before the statement specify with the `--error error-number`. 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 1.7.1.1 [Mailing-list], page 32. As this list does not accept attachments, you should ftp all the relevant files to: `ftp://support.mysql.com/pub/mysql/Incoming/`

22.1.2.3 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 1.7.1.3 [Bug reports], page 34.
- 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://support.mysql.com/pub/mysql/secret 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 `http://www.mysql.com/downloads/`. All our standard binaries should pass the test suite !
- If you get an error, like `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 mysqld version produces slight 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 D.1.2 [Making trace files], page 1200.

  If you have not compiled MySQL for debugging you should probably do that. Just specify the `--with-debug` options to `configure`! See Section 2.3 [Installing source], page 98.

22.2 Adding New Functions to MySQL

There are two ways to add new functions to MySQL:

- You can add the function through the user-defined function (UDF) interface. User-defined functions are added and removed dynamically using the `CREATE FUNCTION` and `DROP FUNCTION` statements. See Section 22.2.1 [CREATE FUNCTION], page 988.
- You can add the function as a native (built in) MySQL function. Native functions are compiled into the `mysqld` server and become available on a permanent basis.

Each method has advantages and disadvantages:

- If you write a user-defined function, you must install the object file 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. For native functions, you must repeat your modifications each time you upgrade.

Whichever method you use to add new functions, they may be used just like native functions such as `ABS()` or `SOUNDEX()`.

22.2.1 CREATE FUNCTION/DROP FUNCTION Syntax

```
CREATE [AGGREGATE] FUNCTION function_name RETURNS {STRING|REAL|INTEGER}
       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 native (built in) MySQL function such as `ABS()` and `CONCAT()`.

`AGGREGATE` is a new option for MySQL 3.23. An `AGGREGATE` function works exactly like a native MySQL `GROUP` function like `SUM` or `COUNT()`.

`CREATE FUNCTION` saves the function's name, type, and shared library name in the `mysql.func` system table. You must have the `INSERT` and `DELETE` privileges for the `mysql` database to create and drop functions.

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. (An active function is one that has been loaded with `CREATE FUNCTION` and not removed with `DROP FUNCTION`.)

For instructions on writing user-defined functions, see Section 22.2 [Adding functions], page 987. 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).

Note that to make `AGGREGATE` work, you must have a `mysql.func` table that contains the column `type`. If you do not have this table, you should run the script `mysql_fix_privilege_tables` to create it.

22.2.2 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.

For `mysqld` to be able to use UDF functions, you should configure MySQL with `--with-mysqld-ldflags=-rdynamic` The reason is that to on many platforms (including Linux) you can load a dynamic library (with `dlopen()`) from a static linked program, which you would get if you are using `--with-mysqld-ldflags=-all-static` If you want to use an UDF that needs to access symbols from `mysqld` (like the `metaphone` example in '`sql/udf_example.cc`' that uses `default_charset_info`), you must link the program with `-rdynamic` (see `man dlopen`).

If you are using a precompiled version of the server, use MySQL-Max, which 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 type and 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 (like `SUM()`), you must also provide the following functions:

`xxx_reset()` (required)
: Reset sum and insert the argument as the initial value for a new group.

`xxx_add()` (required)
: Add the argument to the old sum.

When using aggregate UDFs, MySQL works the following way:

1. Call `xxx_init()` to let the aggregate function allocate the memory it will need to store results.
2. Sort the table according to the `GROUP BY` expression.
3. For the first row in a new group, call the `xxx_reset()` function.

4. For each new row that belongs in the same group, call the `xxx_add()` function.
5. When the group changes or after the last row has been processed, call `xxx()` to get the result for the aggregate.
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 (not just the main function, but the initialization and deinitialization functions as well). This means 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()`.

22.2.2.1 UDF Calling Sequences for simple functions

The main function should be declared as shown here. Note that the return type and parameters differ, depending on whether you will 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 string result. The default 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, you can set this to 65KB or 16MB; this memory is not allocated but used to decide which column type to use if there is a need to temporary 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 between functions. In `xxx_init()`, 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.

22.2.2.2 UDF Calling Sequences for aggregate functions

Here follows a description of the different functions you need to define when you want to create an aggregate UDF function.

Note that the following function is NOT needed or used by MySQL 4.1.1. You can keep still have define this function if you want to have your code work with both MySQL 4.0 and MySQL 4.1.1

```
char *xxx_reset(UDF_INIT *initid, UDF_ARGS *args,
                char *is_null, char *error);
```

This function is called when MySQL finds the first row in a new group. In the function you should reset any internal summary variables and then set the given argument as the first argument in the group.

In many cases this is implemented internally by reseting all variables (for example by calling `xxx_clear()` and then calling `xxx_add()`.

The following function is only required by MySQL 4.1.1 and above:

```
char *xxx_clear(UDF_INIT *initid, char *is_null, char *error);
```

This function is called when MySQL needs to reset the summary results. This will be called at the beginning for each new group but can also be called to reset the values for a query where there was no matching rows. `is_null` will be set to point to `CHAR(0)` before calling `xxx_clear()`.

You can use the `error` pointer to store a byte if something went wrong .

```
char *xxx_add(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

This function is called for all rows that belongs to the same group, except for the first row. In this you should add the value in UDF_ARGS to your internal summary variable.

The `xxx()` function should be declared identical as when you define a simple UDF function. See Section 22.2.2.1 [UDF calling], page 990.

This function is called when all rows in the group has been processed. You should normally never access the `args` variable here but return your value based on your internal summary variables.

All argument processing in `xxx_reset()` and `xxx_add()` should be done identically as for normal UDFs. See Section 22.2.2.3 [UDF arguments], page 992.

The return value handling in `xxx()` should be done identically as for a normal UDF. See Section 22.2.2.4 [UDF return values], page 993.

The pointer argument to `is_null` and `error` is 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 if the `xxx()` function should return `NULL`. Note that you should not store a string into `*error`! This is just a 1 byte flag!

`is_null` is reset for each group (before calling `xxx_clear()`). `error` is never reset.

If `is_null` or `error` are set after `xxx()`, then MySQL will return `NULL` as the result for the group function.

22.2.2.3 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 want 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`

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 coercion of the first two arguments to string and integer, 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 your function was called with. 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).

22.2.2.4 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 will be 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 functions should return a pointer to the result and store the length of the string in the `length` arguments.

Set these 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 calc function is 255 byte big. 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 22.2.2.1 [UDF calling], page 990.

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 the `error` parameter 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()` will not even be called for subsequent rows.) **Note:** In MySQL versions prior to 3.22.10, you should set both `*error` and `*is_null`:

```
*error = 1;
*is_null = 1;
```

22.2.2.5 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 '`udf_example.cc`' that is included in the MySQL source distribution. This 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 an 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 with a string `'xxx.xxx.xxx.xxx'` or 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 myfunc.cc
```

You can easily find out 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.)

Once 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 searched by the dynamic linker ld, such as '/usr/lib' or add the directory in which you placed the shared object to the linker configuration file (for example, '/etc/ld.so.conf').

On many systems, you can also set the LD_LIBRARY or LD_LIBRARY_PATH environment variable to point at the directory where you have your UDF function files. 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.

After the library is installed, notify mysqld about the new functions with these commands:

```
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 system table func 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 already 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 will continue to use the old version.

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. (An active function is one that has been loaded with `CREATE FUNCTION` and not removed with `DROP FUNCTION`.)

22.2.3 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 will 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#) (where # 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 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!

22.3 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 you to take a look at mylua. With this you can use the LUA language to load a procedure at runtime into mysqld.

22.3.1 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 will 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 memory analyse should allocate per column while trying to find all distinct values.

```
SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max elements,[max memory]])
```

22.3.2 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 MySQL users have encountered. It describes how to figure out what the causes of the problems are, 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 5.8 [Log Files], page 345.
- If you don't think you have hardware problems, you should try to find out which program is causing problems. Try using `top`, `ps`, `taskmanager`, 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 1.7.1.2 [Asking questions], page 34.

A.2 Common Errors When Using MySQL

This section lists some errors that users frequently encounter. You will find descriptions of the errors, and how to solve the problem here.

A.2.1 `Access denied`

See Section 5.4.8 [Access denied], page 290. See Section 5.4.2 [Privileges], page 276.

A.2.2 `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. 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], page 240.

If you have a script, you just have to issue the query again for the client to do an automatic reconnection.

You normally get one of the the following error codes in this case (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.

You will also get an error if someone has killed the running thread with a `KILL` statement or a `mysqladmin kill` command.

Another common reason the `MySQL server has gone away` error occurs within an application program is that 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 can check whether the MySQL server died and restarted by executing `mysqladmin version` and examining the uptime. If the problem is that `mysqld` crashed, 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.4.1 [Crashing], page 1013.

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.2.10 [Packet too large], page 1005.

You will 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.

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.4.1 [Crashing], page 1013.
- 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 D.1.6 [Reproduceable test case], page 1204.
- 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 Section 1.7.1.2 [Asking questions], page 34.

A.2.3 `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 `mysqld` server is running on Windows 9x or Me, you can connect only via TCP/IP. If the server is running on Windows NT, 2000, or XP and `mysqld` is started with the `--enable-named-pipe` option, you can also connect with named pipes if you run the client on the same 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 will try to connect to the named pipe. If that doesn't work, it will connect 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 `mysqld` server.

Start by checking whether there is a process named `mysqld` running on your server host. (Use `ps` on Unix or the Task Manager on Windows.) If there is no `mysqld` process, you should start one. See Section 2.4.2.3 [Starting server], page 124.

If a `mysqld` process is running, you can check the server 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.
- 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 [Which OS], page 60. 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 (default '`/tmp/mysql.sock`'). 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 [Problems with '`mysql.sock`'], page 1018.
- 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. See Section A.4.5 [Problems with '`mysql.sock`'], page 1018.
- 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 the `mysql_zap` script) before you can restart the MySQL server. See Section A.4.1 [Crashing], page 1013.

- 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 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 will not be able to connect to it using TCP/IP.

A.2.4 `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.
- Use an account that still has a pre-4.1-style password when connecting to the server with a pre-4.1 client program.
- 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:

  ```
  mysql> SET PASSWORD FOR
      -> 'some_user'@'some_host' = OLD_PASSWORD('mypass');
  ```

  Alternatively, use `UPDATE` and `FLUSH PRIVILEGES`:

  ```
  mysql> UPDATE mysql.user SET Password = OLD_PASSWORD('mypass')
      -> WHERE Host = 'some_host' AND User = 'some_user';
  mysql> FLUSH PRIVILEGES;
  ```

  Substitute the password you want to use for "mypass" in the preceding example. 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 such account, use the `Host` and `User` values and assign a password using the `OLD_PASSWORD()` function and either `SET PASSWORD` or `UPDATE`, as described earlier.

For additional background on password hashing and authentication, see Section 5.4.9 [Password hashing], page 295.

A.2.5 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). To work around this, change your password to a value that is eight or fewer characters long.

A.2.6 `Host '...' 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], page 240.

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 &
```

Note that 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 won't do you any good to increase the value of the `max_connect_errors` variable.

A.2.7 Too many connections

If you get a `Too many connections` error when you try to connect to the `mysqld` server, this means that there are already `max_connections` clients connected to it.

If you need to support more than the default maximum number of connections (100), you should restart `mysqld` with a bigger value for the `max_connections` variable.

Note that `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 clients already are connected. See Section 14.5.3.15 [SHOW PROCESSLIST], page 713.

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.

A.2.8 Some non-transactional changed tables couldn't be rolled back

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 will not be 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 will instead create 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 14.5.3.17 [SHOW TABLE STATUS], page 717 and Section 14.5.3.6 [SHOW CREATE TABLE], page 708.

You can check the storage engines that your `mysqld` server supports by using this statement:

```
SHOW ENGINES;
```

Before MySQL 4.1.2, 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 14.5.3.8 [SHOW ENGINES], page 708 and Section 14.5.3.19 [SHOW VARIABLES], page 718.

A.2.9 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()` to retrieve the result set, which places less of a load on the client (but more on the server).

A.2.10 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.

Note that both the client and the server have their own `max_allowed_packet` variable. If you want to handle big packets, you have to 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 more 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 restart `mysqld`.

A.2.11 Communication Errors / Aborted Connection

The server error log can be a useful source of information about connection problems. See Section 5.8.1 [Error log], page 345. 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 missing any requests to the server. See Section 5.2.3 [Server system variables], page 240.
- 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 a wrong 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], page 240.

Note that if these kinds of things happen, it might indicate that someone is trying to break into your database!

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.2.10 [Packet too large], page 1005.

A.2.12 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 14.5.3.1 [SET], page 702.

  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 solution is to extend the `InnoDB` tablespace. See Section 16.8 [Adding and removing], page 778.
- 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], page 240.

  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.2.13 Can't create/write to file

If you get an error for some queries of the following type, it means that MySQL can't 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 error message for Windows; the Unix error message is similar. The 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 already exist. See Section 4.3.2 [Option files], page 212.

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.2.14 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.2.15 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('your password')
      -> WHERE User='some_user' AND Host='some_host';
  ```

A.2.16 `Table 'xxx' doesn't exist`

If you get either of the following errors, it usually means that no table exists in the current database with the name `xxx`:

```
Table 'xxx' doesn't exist
Can't find file: 'xxx' (errno: 2)
```

In some cases, it may be that the table does exist but that you are not referring to it correctly:

- 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 you have in the current database with `SHOW TABLES`. See Section 14.5.3 [SHOW], page 701.

A.2.17 `Can't initialize character set xxx`

You might see an error like this if you have character set problems:

```
MySQL Connection Failed: Can't initialize character set xxx
```

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=xxx` or `--with-extra-charsets=xxx` option. See Section 2.3.2 [configure options], page 101.

  All standard MySQL binaries are compiled with `--with-extra-character-sets=complex`, which enables support for all multi-byte character sets. See Section 5.7.1 [Character sets], page 340.
- 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 2.3.2 [configure options], page 101.
  - 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.2.18 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
File table overflow
shell> perror 24
Too many open files
shell> perror 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 will reduce 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 the `open_files_limit` system variable. See Section 5.2.3 [Server system variables], page 240. The easiest way to set these values is to add an option to your option file. See Section 4.3.2 [Option files], page 212. 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 affect the number of file descriptors 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 only be overridden 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). 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` will not work! `tcsh` will also report 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 When Linking with 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 find the correct directory, try this command:

```
shell> mysql_config --libs
```

The command output 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` functions, add `-lz` to the end of your link command and try again.

If you get `undefined reference` errors for functions 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 2.3 [Installing source], page 98.

You might get undefined reference errors at runtime when you try to execute a MySQL program. If these errors indicate symbols that start with `mysql_` or 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 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 will switch 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
   ```

At this point, your mysqld process should be running as the Unix user user_name. One thing hasn't changed, though: the contents of the grant table in the mysql database. By default, the grant tables are initialized such that only the MySQL root accounts have permission to access the mysql database or to create or drop databases. Unless you have changed those permissions, they still hold. This does not stop you from accessing MySQL as the MySQL root user when you're logged in as a Unix user other than root; just start your client programs with the --user=root option.

Note that accessing the MySQL server as the MySQL root user from a client program by supplying --user=root on the command line has *nothing* to do with MySQL running as the Unix root user, or, indeed, as any other Unix user. The access permissions and usernames of MySQL accounts are completely separate from those of Unix login accounts. The only connection with Unix usernames is that if you don't provide a --user option when you invoke a client program, the client will try to connect using your Unix login name as your MySQL username.

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 mysql 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 2.4 [Post-installation], page 115.

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> /path/to/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> /path/to/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 E [Environment variables], page 1211.

A.4 Administration-Related Issues

A.4.1 What to Do If MySQL Keeps Crashing

All MySQL versions are tested on many platforms before they are 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 will always help if you try to find out exactly what crashes your system, because you will 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 5.8.1 [Error log], page 345.

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 D.1.4 [Using stack trace], page 1202. 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 will update 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 the data is written but not the indexes.) This means that the data is safe even if `mysqld` crashes, because the operating system will ensure 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 made mysqld confused.
- 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 will ensure that you are running from a clean state. See Chapter 5 [MySQL Database Administration], page 218.
- 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 will be one of the last queries in the log file just before the server restarts. See Section 5.8.2 [Query log], page 346. 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 1.7.1.3 [Bug reports], page 34.
- Try to make a test case that we can use to repeat the problem. See Section D.1.6 [Reproduceable test case], page 1204.
- Try running the tests in the 'mysql-test' directory and the MySQL benchmarks. See Section 22.1.2 [MySQL test suite], page 984. 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 'bench' directory in the 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 will be 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 D.1 [Debugging server], page 1199.
- Have you 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 responding? 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 will be 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 D.1.3 [Using `gdb` on `mysqld`], page 1201.
  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 1.7.1.3 [Bug reports], page 34. 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 will force 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.

A.4.2 How to Reset the Root Password

If you have never set a `root` password for MySQL, the server will not require a password at all for connecting as `root`. However, it is recommended to set a password for each account. See Section 5.3.1 [Security guidelines], page 269.

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 will prompt you to enter the new password for access. Enter the password that you assigned in the first command.
6. Now you can start the MySQL server 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.

In a Unix environment, the procedure 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 depends 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.

   Now you can stop the MySQL server by sending a normal `kill` (not `kill -9`) to the `mysqld` process, using the name of the '`.pid`' file in the following command:
   ```
   shell> kill `cat /mysql-data-directory/hostname.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 now be able to connect using the new password.

Alternatively, 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 now be able to connect using the new password.

A.4.3 How MySQL Handles a Full Disk

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 six 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 send a `mysqladmin kill` to the thread. The thread will be aborted the next time it checks the disk (in one minute).
- Note that other threads may 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 will allow 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 will remove the big temporary files and mark the table as crashed. The exception is that for `ALTER TABLE`, the old table will be 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 between several physical disks, these paths should end up 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 will fail.

MySQL creates all temporary files as hidden files. This ensures that the temporary files will be removed if `mysqld` is terminated. The disadvantage of using hidden files is that you will 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 [Option files], page 212.

- 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 2.3.2 [`configure` options], page 101.

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 E [Environment variables], page 1211.

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 will 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 is a binary string. You can do this with the `BINARY` operator. Write the condition as either `BINARY col_name LIKE 'a%'` or `col_name LIKE BINARY 'a%'`.

If you want a column always to be treated in case-sensitive fashion, declare it as `BINARY`. See Section 14.2.5 [`CREATE TABLE`], page 670.

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.

If you are using Chinese data in the so-called `big5` encoding, you want to make all character columns `BINARY`. This works because the sorting order of `big5` encoding characters is based on the order of ASCII codes. As of MySQL 4.1, you can explicitly declare that a column should use the `big5` character set:

```
CREATE TABLE t (name CHAR(40) CHARACTER SET big5);
```

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 a `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 Connector/ODBC, it is automatically converted to `NULL` in Connector/ODBC 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 will 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.

The MySQL server packs dates for storage, so it can't store a given date if it would not fit onto the result buffer. Note that MySQL does very limited checking of whether the date is correct. If you store an incorrect date, such as '2004-2-31', MySQL stores it as given. The rules for accepting a date are:

- If MySQL can store and retrieve a given date as given, the date is accepted for DATE and DATETIME columns even if it is not strictly legal.
- Day values from 0 to 31 are accepted for any date. This makes it very convenient for Web applications where you ask year, month, and day in three different fields.
- The day or month value may be zero. This is convenient if you want to store a birthdate in a DATE column and you only know part of the date.

If the date cannot be converted to any reasonable value, a 0 is stored in the DATE column, which will be retrieved as '0000-00-00'. This is both a speed and 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.

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 the =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 only if you are using MySQL 3.23.2 or newer and are using the MyISAM, InnoDB, or BDB storage engine. In earlier versions and with other storage engines, you must declare indexed columns NOT NULL. This also means you cannot then insert NULL into an indexed 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 14.1.5 [LOAD DATA], page 635.

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 AUTO_INCREMENT column, 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 rt FROM tbl_name GROUP BY rt HAVING rt > 0;
SELECT id, COUNT(*) AS cnt FROM tbl_name GROUP BY id HAVING cnt > 0;
SELECT id AS 'Customer identity' FROM tbl_name;
```

Note that 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 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 will 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.5.6 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], page 402.
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 will fix this. See Section A.5.7 [Problems with float], page 1023.
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 too many of them), and add your SELECT statement at the end of the file.

   Verify that the test file demonstrates your 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 1.7.1.1 [Mailing-list], page 32.

A.5.7 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 what is the desired tolerance 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, if we wanted 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.

This section is intended for the cases when MySQL doesn't get it right.

The tools you have available to help MySQL do the "right" things are:

- `EXPLAIN`. See Section 7.2.1 [`EXPLAIN`], page 402.
- `ANALYZE TABLE`. See Section 14.5.2.1 [`ANALYZE TABLE`], page 696.
- `USE INDEX`, `FORCE INDEX`, and `IGNORE INDEX`. See Section 14.1.7 [`SELECT`], page 643.
- Global and table level `STRAIGHT JOIN`. See Section 14.1.7 [`SELECT`], page 643.
- Setting thread-specific system variables. See Section 5.2.3 [Server system variables], page 240 and Section 10.4 [System Variables], page 499.

A.6.1 How to Avoid Table Scans

The output from `EXPLAIN` will show `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 constants 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 [Where optimisations], page 410.
- You are using a key with low cardinality (many rows match the key value) through another column. MySQL will in this case assume that by using the key it will probably do 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 to avoid having the optimizer incorrectly choose a table scan:

- Use `ANALYZE TABLE` to update the key distributions for the scanned table. See Section 14.5.2.1 [`ANALYZE TABLE`], page 696.
- Use `FORCE INDEX` for the scanned table to tell MySQL that table scans are very expensive compared to using the given index. See Section 14.1.7 [`SELECT`], page 643.

  ```
  SELECT * FROM t1,t2 force index(index_for_column)
  WHERE t1.column=t2.column;
  ```

- 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 will cause more than 1,000 key seeks. See Section 5.2.3 [Server system variables], page 240.

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-something`' or '`B-something`' lying around:

```
Error on rename of './database/name.frm'
to './database/B-a.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 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`', but 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` will `UNLOCK` 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_table-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 will not remain the same if you add, move, or delete columns. A simple change to your table structure will then 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 (more than 10 years old now). 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
- Connector/ODBC (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.

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_setpermissions` 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 UDF 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.

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. Will try 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 is 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 Change History

This appendix lists the changes from version to version in the MySQL source code.

We are now working actively on MySQL 4.1 and 5.0, and will 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 4.1 & 5.0. See Section 1.6 [TODO], page 26.

Note that we tend to update the manual at the same time we make changes to MySQL. If you find a version listed here that you can't find on the MySQL download page (`http://www.mysql.com/downloads/`), this means that the version has not yet been released!

The date mentioned with a release version is the date of the last BitKeeper ChangeSet that this particular release has been based on, not the date when the packages have been made available. The binaries are usually made available a few days after the date of the tagged ChangeSet - building and testing all packages takes some time.

C.1 Changes in release 5.0.x (Development)

The following changelog shows what has already been done in the 5.0 tree:

- Basic support for stored procedures (SQL:2003 style). See Chapter 19 [Stored Procedures], page 845.
- Added `SELECT INTO list_of_vars`, which can be of mixed, that is, global and local type. See Section 19.1.6.3 [SELECT INTO Statement], page 850.
- Removed the update log. It is fully replaced by the binary log. If the MySQL server is started with `--log-update`, it will be translated to `--log-bin` (or ignored if the server is explicitly started with `--log-bin`), and a warning message will be written to the error log. Setting `SQL_LOG_UPDATE` will silently set `SQL_LOG_BIN` instead (or do nothing if the server is explicitly started with `--log-bin`).
- User variable names are now case insensitive: If you do `SET @a=10;` then `SELECT @A;` will now return `10`. Case sensitivity of a variable's value depends on the collation of the value.

For a full list of changes, please refer to the changelog sections for each individual 5.0.x release.

C.1.1 Changes in release 5.0.1 (not released yet)

Functionality added or changed:

- 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)

- 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 binlog cache is allocated for connections (this should save `binlog_cache_size` bytes of memory (32 kilobytes by default) for every connection).
- Added option `--replicate-same-server-id`.

Bugs fixed:

- Strange results with index (x, y) ... WHERE x=val_1 AND y>=val_2 ORDER BY pk; (Bug #3155)
- Subquery and order by (Bug #3118)
- `ALTER DATABASE` caused the client to hang if the database did not exist. (Bug #2333)
- `SLAVE START` (which is a deprecated syntax, `START SLAVE` should be used instead) could crash the slave. (Bug #2516)
- Multiple-table `DELETE` statements were never replicated by the slave if there were any `replicate-*-table` options. (Bug #2527)
- The MySQL server did not report any error if the query (submitted through `mysql_real_query()` or `mysql_prepare()`) was terminated by garbage characters (which can happen if you pass a wrong `length` parameter to `mysql_real_query()` or `mysql_prepare()`); the result was that the garbage characters were written into the binary log. (Bug #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)
- Made clearer the error message which one gets when an update is refused because of the `read-only` option. (Bug #2757)
- Fixed that `replicate-wild-*-table` rules apply to `ALTER DATABASE` when the table pattern is '%', like it is already the case for `CREATE DATABASE` and `DROP DATABASE`. (Bug #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)
- Corrected the master's binary log position that `InnoDB` reports when it is doing a crash recovery on a slave server. (Bug #3015)

- Changed the column `Seconds_Behind_Master` in `SHOW SLAVE STATUS` to never show a value of -1. (Bug #2826)
- Changed that when a `DROP TEMPORARY TABLE` statement is automatically written to the binlog 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)
- 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)
- Fixed deadlock when two `START SLAVE` commands were run at the same time. (Bug #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)
- 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)
- `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 was it does when reading a local binary log. (Bug #3204)
- Fixed `mysqlbinlog --read-from-remote-server` to print the exact positions of events in the "at #" lines. (Bug #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)
- 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)
- Fixed a memory corruption when replicating a `LOAD DATA INFILE` when the master had version 3.23. (Bug #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)
- Fixed a crash of the MySQL slave server when it was built with `--with-debug` and replicating itself. (BUG #3568)

C.1.2 Changes in release 5.0.0 (22 Dec 2003: Alpha)

Functionality added or changed:

- The `KILL` statement now takes `CONNECTION` and `QUERY` variants. 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 DATE_ADD() and DATE_SUB() functions.
- New binary log format that enables replication of those session variables: sql_mode, SQL_AUTO_IS_NULL, FOREIGN_KEY_CHECKS (that one was already replicated since 4.0.14 but here it's done more efficiently: takes less space in the binary logs), UNIQUE_CHECKS. Other variables (like character sets, SQL_SELECT_LIMIT...) will be replicated in next 5.0.x releases.
- Implemented Index Merge optimization for OR clauses. See Section 7.2.5 [OR optimizations], page 412.
- Basic support for stored procedures (SQL:2003 style). See Chapter 19 [Stored Procedures], page 845.
- Added SELECT INTO list_of_vars, which can be of mixed, that is, global and local type. See Section 19.1.6.3 [SELECT INTO Statement], page 850.
- Easier replication upgrade (5.0.0 masters can read older binary logs, 5.0.0 slaves can read older relay logs; see Section 6.5 [Replication Compatibility], page 374 for more details). The format of the binary log and relay log is changed compared to the one of MySQL 4.1 and older.
- **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.

Bugs fixed:

C.2 Changes in release 4.1.x (Alpha)

Version 4.1 of the MySQL server includes many enhancements and new features. Binaries for this version are available for download at http://www.mysql.com/downloads/mysql-4.1.html.

- Subqueries and derived tables (unnamed views). See Section 14.1.8 [Subqueries], page 652.
- 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 14.1.4 [INSERT], page 630.
- A newly designed GROUP_CONCAT() aggregate function. See Section 13.9 [Group by functions and modifiers], page 618.
- Extensive Unicode (UTF8) support.
- 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], page 816.
- SHOW WARNINGS shows warnings for the last command. See Section 14.5.3.20 [SHOW WARNINGS], page 719.

- Faster binary protocol with prepared statements and parameter binding. See Section 20.2.4 [C API Prepared statements], page 911.
- You can now issue multiple statements with a single C API call and then read the results in one go. See Section 20.2.8 [C API multiple queries], page 942.
- 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.

C.2.1 Changes in release 4.1.2 (not released yet)

Functionality added or changed:

- New `myisam_data_pointer_size` system variable. See Section 5.2.3 [Server system variables], page 240.
- 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)
- '`mysqld_multi`' now creates the log in datadir (from `[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 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 will have to repair this table. `CHECK TABLES` will tell you if this problem exists. (Bug #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 will 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.
- `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)
- MySQL now issues a warning when a `SET` or `ENUM` column with duplicate values in the list is created. (Bug #1427)
- Now `SQL_SELECT_LIMIT` variable has no influence on subqueries. (Bug #2600)
- `UNHEX()` function implemented. See Section 13.3 [`UNHEX(str)`], page 564.
- History in command line client does not store multiple copies of identical queries that are run consecutively.
- Multi-line queries in the command line client now are stored as a single line.
- `UUID()` function implemented. Note that it does not work with replication yet. See Section 13.8.4 [`UUID()`], page 616.
- Prepared statements with all types of subqueries fixed.
- MySQL now supports up to 64 keys 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], page 240.

- `REVOKE ALL PRIVILEGES, GRANT FROM user_list` is changed to a more consistent `REVOKE ALL PRIVILEGES, GRANT OPTION FROM user_list`. (Bug #2642)
- Internal string-to-number conversion now supports only SQL:2003 compatible syntax for numbers. In particular, `'0x10'+0` will 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)
- 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`.
- Added `init_connect` and `init_slave` server 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)
- 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 binlog cache is allocated for connections (this should save `binlog_cache_size` bytes of memory (32 kilobytes 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 buffer in memory. They can be used for tuning the `binlog_cache_size` system variable.

- Added option `--replicate-same-server-id`.

Bugs fixed:

- `UNION`s 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 multi table `UPDATE`s which resulted in 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)
- Fixed crash on second execution of prepared statement with `UNION`. (Bug #3577)
- Fixed incorrect results of aggregate functions in subquery with empty result set. (Bug #3505)
- One 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).
- Added support for unsigned integer types to prepared statement API (Bug #3035).
- Fixed crash in prepared statements when subquery in the `FROM` clause with parameter used. (Bug #3020)
- Fixed unknown error when negative value bind to unsigned. (Bug #3223)
- Fixed aggregate function in prepared statements. (Bug #3360)
- Incorrect error message when wrong table used in `multi-delete` statement in prepared statements. (Bug #3411)
- Requiring `UPDATE` privilege for tables which will not be updated in `multi-update` query 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, Bug #3406, Bug #2811)
- Fixed a lot of bugs in `GROUP_CONCAT()`. (Bug #2695, Bug #3381, Bug #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)
- 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)
- Fixed a problem resulting from setting the `character_set_results` variable to `NULL`. (Bug #3296)
- Fixed query cache statistics.
- Fixed bug in `ANALYZE TABLE` on a `BDB` table inside a transaction that hangs server thread. (Bug #2342)
- Fixed a symlink vulnerability in '`mysqlbug`' script. (Bug #3284)

- Fixed a bug in parallel repair (`myisamchk -p`, `myisam_repair_threads`); sometimes the repair process failed to repair a table. (Bug #1334)
- A query that uses both `UNION [DISTINCT]` and `UNION ALL` now works correctly. (Bug #1428)
- Table default character set affects `LONGBLOB` columns. (Bug #2821)
- `CONCAT_WS()` makes the server die in case of illegal mix of collations. (Bug #3087)
- UTF8 charset breaks joins with mixed column/string constant. (Bug #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)
- Fixed server segfaults when processing malformed prepared statements commands. (Bug #2795, Bug #2274)
- Fixed using subqueries with `OR` and `AND` functions. (Bug #2838)
- Fixed comparison of tables/database names with `--lower_case_table_names` option. (Bug #2880)
- Removed try to check `NULL` if index built on column where `NULL` is impossible in `IN` subquery optimization. (Bug #2393)
- Fixed incorrect parsing of subqueries in the `FROM` clause. (Bug #2421)
- Fixed processing of `RAND()` in subqueries with static tables. (bug #2645)
- Fixed bug with quoting of table names in `mysqldump` for various values of `sql_mode` of server. (Bug #2591)
- Fixed bug with storing values that are out of range for `DOUBLE` and `FLOAT` columns. (Bug #2082)
- Fixed bug with compiling `--with-pstack` with binutils 2.13.90. (Bug #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)
- 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 "Uppsåla"). (Bug #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)
- Fixed `mysql_stmt_affected_rows()` call to always return number of rows affected by given statement. (Bug #2247)
- Fixed crash in `MATCH ... AGAINST()` on a phrase search operator with a missing closing double quote. (Bug #2708)
- Fixed output of `mysqldump --tab`. (Bug #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)

- Fix for a bug in UNION operations with InnoDB storage engine, when some columns from one table where used in one SELECT statement and some were used in another SELECT statement. (Bug #2552)
- Fixed a few years old bug in the range optimizer that caused a segmentation fault on some very rare queries. (Bug #2698)
- Fixed bug with SHOW CREATE TABLE ... which didn't properly double quotes. (Bug #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)
- Fixed bug with mysqldump not quoting "tricky" names correctly. (Bug #2592)
- Fix for a bug that prevented table / column privileges from being loaded on startup. (Bug #2546)
- Fixed bug in replication with CREATE TABLE ... LIKE ... that resulted in a statement not being written to the binary log. (Bug #2557)
- Fixed memory leak in INSERT ... ON DUPLICATE KEY UPDATE .... (Bug #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)
- 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)
- Compile the MySQL-client RPM package against libreadline instead of libedit. (Bug #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)
- Fix for a bug that caused mysql client program to erroneously cache the value of the current database. (Bug #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)
- Fix for a bug that caused wong results when CAST() was applied on NULL to signed or unsigned integer column. (Bug #2219)
- Fix for a crashing bug that occurred in the mysql client program when database name was longer then expected. (Bug #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)
- Fixed bug in GRANT with table-level privilege handling. (Bug #2178)
- Fixed bug in ORDER BY on a small column. (Bug #2147)
- Fixed a bug with the INTERVAL() function when 8 or more comparison arguments are provided. (Bug #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)
- Fixed a hang in full-text indexing of strings in multi-byte (all besides utf8) charsets. (Bug #2065)
- Fixed a crash in full-text indexing of UTF8 data. (Bug #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)
- 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)
- 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)
- 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 [Full disk], page 1017); 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)
- 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)
- Fixed a crash when the replication slave was unable to create the first relay log. (Bug #2145)
- ALTER DATABASE caused the client to hang if the database did not exist. (Bug #2333)
- Multiple-table DELETE statements were never replicated by the slave if there were any replicate-*-table options. (Bug #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)
- The MySQL server did not report any error if the query (submitted through mysql_real_query() or mysql_prepare()) was terminated by garbage characters (which can happen if you pass a wrong length parameter to mysql_real_query() or mysql_prepare()); the result was that the garbage characters were written into the binary log. (Bug #2703)
- Fixed bug in client library which caused mysql_fetch and mysql_stmt_store_result() to hang if they were called without prior call of mysql_execute(). Now they give an error instead. (Bug #2248)

- Made clearer the error message which one gets when an update is refused because of the `read-only` option. (Bug #2757)
- Fixed that `replicate-wild-*-table` rules apply to `ALTER DATABASE` when the table pattern is '%', like it is already the case for `CREATE DATABASE` and `DROP DATABASE`. (Bug #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)
- Corrected the master's binary log position that `InnoDB` reports when it is doing a crash recovery on a slave server. (Bug #3015)
- Changed the column `Seconds_Behind_Master` in `SHOW SLAVE STATUS` to never show a value of -1. (Bug #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)
- 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)
- Fixed deadlock when two `START SLAVE` commands were run at the same time. (Bug #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)
- 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)
- `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 was it does when reading a local binary log. (Bug #3204)
- Fixed `mysqlbinlog --read-from-remote-server` to print the exact positions of events in the "at #" lines. (Bug #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)
- 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)
- 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)
- 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], page 376. (Bug #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)
- Fixed a crash of the MySQL slave server when it was built with `--with-debug` and replicating itself. (BUG #3568)

C.2.2 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:

- 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 14.5.4.1 [CACHE INDEX], page 722. See Section 14.5.4.4 [LOAD INDEX], page 725. Structured system variables are introduced as a means of grouping related key cache parameters. See Section 10.4.1 [Structured System Variables], page 501.
- 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.
- **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.
- One can revoke all privileges from a user with `REVOKE ALL PRIVILEGES, GRANT FROM user_list`.
- Added `IGNORE` option for `DELETE` statement.
- The MySQL source distribution now also includes 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 will continue 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 server 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 will take 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.
- **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.
- **Warning: Incompatible change!** Renamed the C API mysql_prepare_result() function to mysql_get_metadata() as the old name was confusing.
- Added DROP USER 'user_name'@'host_name' statement to drop an account that has no privileges.

- The interface to aggregated UDF functions has changed a bit. You must now declare a `xxx_clear()` function for each aggregate function `XXX()`.
- 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 (like `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 14.5.4.2 [FLUSH], page 722.
- 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 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` (which had been deleted in version 4.1.0) as an alias for `SHOW BINARY LOGS`.
- 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)
- 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)
- 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)
- Fix for bug in `ROLLUP` when all tables were `const` tables. (Bug #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)
- Fixed memory overrun in subqueries in select list with `WHERE` clause bigger than outer query `WHERE` clause. (Bug #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)
- Fixed name resolution of outer columns of subquery in `INSERT/REPLACE` statements. (Bug #446)
- Fixed bug in marking columns of reduced subqueries. (Bug #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)
- 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)
- 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)
- `CONNECTION_ID()` now is properly replicated. (Bug #177)
- The new `PASSWORD()` function in 4.1 is now properly replicated. (Bug #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)
- 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)
- 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)
- 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)
- `SLAVE START` (which is a deprecated syntax, `START SLAVE` should be used instead) could crash the slave. (Bug #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)

C.2.3 Changes in release 4.1.0 (03 Apr 2003: Alpha)

Functionality added or changed:

- 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.
- 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)
- 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 14.5.4.3 [KILL], page 724.
- `REPAIR TABLE` now can be killed. See Section 14.5.4.3 [KILL], page 724.
- 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 will get a warning in this case.
- One can specify the different `BLOB/TEXT` types with the syntax `BLOB(length)` and `TEXT(length)`. MySQL will automatically change 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`.
- `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], page 816.
- 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 already 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.

C.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 16 [InnoDB], page 758.
- A query cache, offering vastly increased performance for many applications. By caching complete result sets, later identical queries can return instantly. See Section 5.10 [Query Cache], page 358.
- Improved full-text indexing with boolean mode, truncation, and phrase searching. See Section 13.6 [Fulltext Search], page 597.
- Enhanced MERGE tables, now supporting INSERT statements and AUTO_INCREMENT. See Section 15.2 [MERGE], page 746.
- UNION syntax in SELECT. See Section 14.1.7.2 [UNION], page 650.
- Multiple-table DELETE statements. See Section 14.1.1 [DELETE], page 626.
- libmysqld, the embedded server library. See Section 20.2.15 [libmysqld], page 951.
- Additional GRANT privilege options for even tighter control and security. See Section 14.5.1.2 [GRANT], page 689.
- Management of user resources in the GRANT system, particularly useful for ISPs and other hosting providers. See Section 5.5.5 [User resources], page 307.
- Dynamic server variables, allowing configuration changes to be made without having to stop and restart the server. See Section 14.5.3.1 [SET OPTION], page 702.
- Improved replication code and features. See Chapter 6 [Replication], page 363.

- 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.

C.3.1 Changes in release 4.0.19 (not released yet)

Functionality added or changed:

- If length of a timestamp field is defined as 19, the timestamp will be 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 will be set to 255. This was made to ensure that all raid directories are always 2 hex bytes. (Bug #3182)
- Changed that the optimizer will now consider the index specified in `FORCE INDEX` clause as a candidate to resolve `ORDER BY` as well.
- Non-standard behavior of `UNION` statements has changed to the standard ones. So far, a table name in the `ORDER BY` clause was tolerated. From now on a proper error message is issued (Bug #3064).
- Added `max_insert_delayed_threads` system variable as a synonym for `max_delayed_threads`.
- Added `query_cache_wlock_invalidate` system variable. It allow emulation of `MyISAM` table write-locking behavior, even for queries in the query cache. (Bug #2693)
- The keyword `MASTER_SERVER_ID` is not reserved anymore.
- The following is mainly relevant 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 file names in lower case:

  One 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 your have already created `InnoDB` tables. With `lower_case_table_names=0`, `InnoDB` tables were stored in mixed case while setting lower_case_table_names `<>` 0 will now force 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.
- Non-standard behavior of `UNION` statements has changed to the standard ones. So far, a table name in the `ORDER BY` clause was tolerated. From now on a proper error message is issued (Bug #3064).
- Added `max_insert_delayed_threads` system variable as a synonym for `max_delayed_threads`.

- Added `query_cache_wlock_invalidate` system variable. It allow emulation of `MyISAM` table write-locking behavior, even for queries in the query cache. (Bug #2693)
- 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)
- `INTERVAL` now can handle big values for seconds, minutes and hours. (Bug #3498)
- Blank hostname did not work as documented for table and column privileges. Now it's works the same way as `'%'`. (Bug #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)
- 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 (e.g. "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)
- Fixed a symlink vulnerability in '`mysqlbug`' script. (Bug #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)
- 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 multi-table-update and BDB tables. (Bug: #3098)
- Fixed memory leak when dropping database with `RAID` tables. (Bug #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)
- 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)

- 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)
- Rollback `DELETE` and `UPDATE` statements if thread is killed. (Bug #2422)
- Ensure that all rows in an `INSERT DELAYED` statement is written at once if binary logging is enabled. (Bug #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)
- Fixed bugs with names of tables, databases and columns that end to space (Bug #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).
- Fixed all bugs in scripts for creating/upgrading system database (Bug #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)
- `--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)
- Fixed a few years old bug in the range optimizer that caused a segmentation fault on some very rare queries. (Bug #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) 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)
- Corrected the master's binary log position that `InnoDB` reports when it is doing a crash recovery on a slave server. (Bug #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)

- 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)
- Fixed deadlock when two START SLAVE commands were run at the same time. (Bug #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)
- 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)
- 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)
- 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)
- 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)
- 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], page 376. (Bug #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)
- Fixed a crash of the MySQL slave server when it was built with --with-debug and replicating itself. (BUG #3568)

C.3.2 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)
- New utility program 'myisam_ftdump' was added to 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)
- Optimizer is now better tuned for the case where the first used key part (of many) is a constant. (Bug #1679)
- Removed old non-working `--old-rpl-compat` server option, which was a holdover from the very first 4.0.x versions. (Bug #2428)

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)
- Fixed bug in range optimizer when using overlapping ranges. (Bug #2448)
- Limit `wait_timeout` to 2147483 on Windows (OS limit). (Bug #2400)
- Fixed bug when `--init-file` crashes MySQL if it contains a large `SELECT`. (Bug #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)
- Now only the `SELECT` privilege is needed for tables that are only read in multiple-table `UPDATE` statements. (Bug #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)
- `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)
- 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)
- 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 indexed `TEXT` columns that happened to contain values having trailing spaces. This bug was introduced in 4.0.17. (Bug #2295)
- Fixed incorrect path names in some of the manual pages. (Bug #2270)
- Fixed spurious "table corrupted" errors in parallel repair operations. See Section 5.2.3 [Server system variables], page 240.
- Fixed a crashing bug in parallel repair operations. See Section 5.2.3 [Server system variables], page 240.
- Fixed bug in updating `MyISAM` tables for `BLOB` values longer than 16MB. (Bug #2159)

- Fixed bug in `mysqld_safe` when running multiple instances of MySQL. (Bug #2114)
- Fixed a bug in using `HANDLER` statement with tables not from a current database. (Bug #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)
- Fixed a crashing bug that occurred due to `BLOB` column type index size being calculated incorrectly in `MIN()` and `MAX()` optimizations. (Bug #2189)
- Fixed a bug with incorrect syntax for `LOCK TABLES` in `mysqldump`. (Bug #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)
- `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)
- 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 [Full disk], page 1017.) 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)
- 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)
- Fixed a crash when the replication slave was unable to create the first relay log. (Bug #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)
- 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)
- Restored ability to specify default values for `TIMESTAMP` columns that was erroneously disabled in previous release. (Bug #2539) Fixed `SHOW CREATE TABLE` to reflect these values. (Bug #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 will be 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)
- 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) 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)

- 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)
- 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)

C.3.3 Changes in release 4.0.17 (14 Dec 2003)

Functionality added or changed:

- `mysqldump` no longer dumps data for `MERGE` tables. (Bug #1846)
- `lower_case_table_names` is now forced to 1 if the database directory is located on a case-insensitive filesystem. (Bug #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)
- Added four new modes to `WEEK(..., mode)` function. See Section 13.5 [`WEEK(date,mode)`], page 582. (Bug #1178)
- Allow `UNION DISTINCT` syntax.
- `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)
- 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` will read 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)
- One can now configure MySQL as a Windows service as a normal user. (Bug #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)
- `IGNORE ... LINES` option to `LOAD DATA INFILE` didn't work when used with fixed length rows. (Bug #1704)
- Fixed problem with `UNIX_TIMESTAMP()` for timestamps close to 0. (Bug #1998)
- Fixed problem with character values greater than 128 in the `QUOTE()` function. (Bug #1868)
- Fixed searching of `TEXT` with endspace. (Bug #1651)

- Fixed caching bug in multiple-table updates where same table was used twice. (Bug #1711)
- Fixed directory permissions for the MySQL-server RPM documentation directory. (Bug #1672)
- Fixed server crash when updating an `ENUM` column that is set to the empty string (for example, with `REPLACE()`). (Bug #2023)
- `mysql` client program now correctly prints connection identifier returned by `mysql_thread_id()` as unsigned integer rather than as signed integer. (Bug #1951)
- `FOUND_ROWS()` could return incorrect number of rows after a query with an impossible `WHERE` condition. (Bug #1468)
- `SHOW DATABASES` no longer shows `.sym` files (on Windows) that do not point to a valid directory. (Bug #1385)
- Fixed a possible memory leak on Mac OS X when using the shared `libmysql.so` library. (from `pthread_key_create()`). (Bug #2061)
- Fixed bug in `UNION` statement with alias `*`. (Bug #1249)
- Fixed a bug in `DELETE ... ORDER BY ... LIMIT` where the rows where not deleted in the proper order. (Bug #1024, Bug #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, uninitialized 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)
- Filesort was never shown in `EXPLAIN` if query contained an `ORDER BY NULL` clause. (Bug #1335)
- Fixed invalidation of whole query cache on `DROP DATABASE`. (Bug #1898)
- Fixed bug in range optimizer that caused wrong results for some unlikely `AND`/`OR` queries. (Bug #1828)
- Fixed a crash in `ORDER BY` when ordering by expression and identifier. (Bug #1945)
- Fixed a crash in an open `HANDLER` when an `ALTER TABLE` was executed in a different connection. (Bug #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)
- 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)

- Fixed bug when the optimizer did not take `SQL_CALC_FOUND_ROWS` into account if `LIMIT` clause was present. (Bug #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` already 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)
- 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)
- 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)
- 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 14.6.2.1 [`CHANGE MASTER TO`], page 728. (Bug #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)
- 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).
- `BIT_AND()` is now unsigned in all contexts. This means that it will now return 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)
- Fixed bug with `^` (XOR) and `>>` (bit shift) still returning signed value in some cases. (Bug #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)
- 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)
- 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)

C.3.4 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 server 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)
- Fixed a bug that resulted in `thr_alarm queue is full` warnings after increasing the `max_connections` variable with `SET GLOBAL`. (Bug #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).
- 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).
- 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, Bug #1304, and maybe Bug #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 will also be 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 already 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)
- 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)
- `mysqlbinlog` will not reuse temporary file names 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)
- 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)
- On Windows, the slave forgot to delete a `SQL_LOAD-*.info` file from `tmpdir` after successfully replicating a `LOAD DATA INFILE` command. (Bug #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)

- 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, Bug #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)

C.3.5 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` will never send 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)
- Change result set metadata information so that `MIN()` and `MAX()` report that they can return `NULL` (this is true because an empty set will return `NULL`). (Bug #324)
- Produce an error message on Windows if a second `mysqld` server is started on the same TCP/IP port as an already running `mysqld` server.
- The `mysqld` server 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` will 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` server 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 2.2.3 [Mac OS X installation], page 91.
- 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], page 76.

- The manual is now included as an additional `info` file in the binary distributions. (Bug #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 1.5.1.2 [Nutshell Embedded MySQL], page 23.
- MySQL can now use range optimization for `BETWEEN` with non-constant limits. (Bug #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 5.8.4 [Binary log], page 347), 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 Options], page 379).
- 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 already the case for `GRANT` and `REVOKE` since version 4.0.13 (although there was Bug #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], page 364.
- 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)
- `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)

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)
- Fixed table/column grant handling: The proper sort order (from most specific to less specific, see Section 5.4.6 [Request access], page 287) was not honored. (Bug #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)
- Fixed possible deadlock when changing key_buffer_size while the key cache was actively used. (Bug #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)
- Fixed the bug that sometimes prevented a table with a FULLTEXT index from being marked as "analyzed".
- 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)
- Fixed crash in comparisons of strings using the tis620 character set. (Bug #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)
- 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)
- Fixed bug in propagating the version number to the manual included in the distribution files. (Bug #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)
- Fixed compiling of the embedded server library in the RPM spec file. (Bug #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)
- Fixed incorrect XOR evaluation in WHERE clause. (Bug #992)
- Fixed bug with processing in query cache merged tables constructed from more then 255 tables. (Bug #930)

- Fixed incorrect results from outer join query (e.g. `LEFT JOIN`) when `ON` condition is always false, and range search in used. (Bug #926)
- Fixed a bug causing incorrect results from `MATCH ... AGAINST()` in some joins. (Bug #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)
- 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)
- Fixed segmentation fault resulting from `LOAD DATA FROM MASTER` when the master was running without the `--log-bin` option. (Bug #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)
- 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)
- 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)
- `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)
- 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)
- 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)
- `mysqlbinlog` printed superfluous warnings when using `--database`, which caused syntax errors when piped to `mysql`. (Bug #1092)
- Made `mysqlbinlog --database` filter `LOAD DATA INFILE` too (previously, it filtered all queries except `LOAD DATA INFILE`). (Bug #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)
- `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)

- 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)
- 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 already the case for normal `INSERT`. (Bug #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).

C.3.6 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).
- `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], page 388.

- A documentation change: added warning notes about safe use of the CHANGE MASTER command. See Section 14.6.2.1 [CHANGE MASTER TO], page 728.
- 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)
- 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 accidentally 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 will be rotated automatically when its size exceeds `max_relay_log_size`. But if `max_relay_log_size` is 0 (the default), `max_binlog_size` will be 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 already rotated.

Bugs fixed:

- Comparison/sorting for `latin1_de` character set was rewritten. The old algorithm could not handle cases like `"sä" > "ßa"`. See Section 5.7.1.1 [German character set], page 341. In rare cases it resulted in table corruption.
- Fixed a problem with the password prompt on Windows. (Bug #683)
- `ALTER TABLE ... UNION=(...)` for `MERGE` table is now allowed even if some underlying `MyISAM` tables are read-only. (Bug #702)
- Fixed a problem with `CREATE TABLE t1 SELECT x'41'`. (Bug #801)
- Removed some incorrect lock warnings from the error log.
- Fixed memory overrun when doing `REPAIR TABLE` on a table with a multi-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)
- Fixed a crashing bug in `INTERVAL` and `GROUP BY` or `DISTINCT`. (Bug #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)
- Fixed problem with `mysqlhotcopy` failing to read options from option files. (Bug #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)
- Fixed a bug with `LOAD DATA` in `mysqlbinlog`. (Bug #670)
- Fixed that `SET CHARACTER SET DEFAULT` works. (Bug #462)
- Fixed `MERGE` table behavior in `ORDER BY ... DESC` queries. (Bug #515)
- Fixed server crash on `PURGE MASTER LOGS` or `SHOW MASTER LOGS` when the binary log is off. (Bug #733)
- Fixed password-checking problem on Windows. (Bug #464)
- Fixed the bug in comparison of a `DATETIME` column and an integer constant. (Bug #504)
- Fixed remote mode of `mysqlbinlog`. (Bug #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)
- Very small query cache sizes could crash `mysqld`. (Bug #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)

- 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 Options], page 379. (Bug #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 will produce 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)
- 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)
- When using a slave compiled with `--debug`, `CHANGE MASTER TO RELAY_LOG_POS` could cause a debug assertion failure. (Bug #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)
- 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)
- 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)
- After `RESET SLAVE; START SLAVE;`, the `Relay_Log_Space` value displayed by `SHOW SLAVE STATUS` was too big by four bytes. (Bug #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)

C.3.7 Changes in release 4.0.13 (16 May 2003)

Functionality added or changed:

- `PRIMARY KEY` now implies `NOT NULL`. (Bug #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)
- Added multi-threaded `MyISAM` repair optimization and `myisam_repair_threads` variable to enable it. See Section 5.2.3 [Server system variables], page 240.
- 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 will be the default behavior in MySQL 4.1. (Bug #152)
- Enabled `delayed_insert_timeout` on Linux (most modern `glibc` libraries have a fixed `pthread_cond_timedwait()`). (Bug #211)
- Don't create more insert delayed threads than given by `max_delayed_threads`. (Bug #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 16.5 [InnoDB start], page 764.

- 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_debug.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` will not be replicated.

Bugs fixed:

- Logged `Access denied` error message had incorrect `Using password` value. (Bug #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)
- 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)
- Fixed problem with ANSI mode and `GROUP BY` with constants. (Bug #387)
- Fixed bug with `UNION` and `OUTER JOIN`. (Bug #386)
- Fixed bug if one used a multiple-table `UPDATE` and the query required a temporary table bigger than `tmp_table_size`. (Bug #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` will now read `000000` as a zero date instead as `"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)
- Fixed core dump bug when using `FORMAT('nan',#)`. (Bug #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)
- Fixed bug in `LEFT`, `RIGHT` and `MID` when used with multi-byte character sets and some `GROUP BY` queries. (Bug #314)
- Fix problem with `ORDER BY` being discarded for some `DISTINCT` queries. (Bug #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)
- 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)
- 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 links 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)
- 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, Bug #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)

- 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)
- Fixed a deadlock when relay_log_space_limit was set to a too small value. (Bug #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)
- 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)
- Fixed a bug when an updating query using USER() was replicated on the slave; this caused segfault on the slave. (Bug #178). USER() is still badly replicated on the slave (it is replicated to "").

C.3.8 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 99999999999999999999 are now regarded as unsigned longlongs, not as floats. This makes these values work similar to values between 10000000000000000000 and 18446744073709551615.
- SHOW PROCESSLIST will now include the client TCP port after the hostname to make it easier to know from which client the request originated.

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)
- Fixed checking of random part of WHERE clause. (Bug #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)
- 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)
- `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 ok.
- `SET SQL_LOG_BIN=0` was ignored for `INSERT DELAYED` queries. (Bug #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)
- `LOAD DATA INFILE` was not replicated by the slave if `replicate_*_table` was set on the slave. (Bug #86)
- After `RESET SLAVE`, the coordinates displayed by `SHOW SLAVE STATUS` looked un-reset (although they were, but only internally). (Bug #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 [`AUTO_INCREMENT` on secondary column in a multi-column key], page 203), 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)
- 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 (e.g. Mac OS X) due to a too long file name (changed `slave-master-info.opt` to `.slave-mi`).

C.3.9 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`.

C.3.10 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 will force 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], page 240.
- 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` will simply replace `MySQL.rpm`.
- If a slave is configured with `replicate_wild_do_table=db.%` or `replicate_wild_ignore_table=db.%`, these rules will be 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.

C.3.11 Changes in release 4.0.9 (09 Jan 2003)

Functionality added or changed:

- `OPTIMIZE TABLE` will for `MyISAM` tables treat 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 will reduce memory in the 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.

C.3.12 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.
- One 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 uses 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 will 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 ...

C.3.13 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.

C.3.14 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 will restart 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 cache because of low memory).
- Fixed mysqlcheck so it can deal with table names containing dashes.
- Bulk insert optimization (see Section 5.2.3 [bulk_insert_buffer_size], page 240) 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 13.8.3 [CURRENT_USER()], page 611.
- 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() will now reset 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 be both already stopped; these commands will 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 10.2.2 [Name case sensitivity], page 497.
- 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.

C.3.15 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` will now resolve `basedir` to its full path (with `realpath()`). This enables one to use relative symlinks to the MySQL installation directory. This will however cause `show variables` to report different directories on systems where there is a symbolic link in the path.
- Fixed that MySQL will 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 13.1.2 [Comparison Operators], page 556). 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 will 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 C.9 [InnoDB change history], page 1178 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 will 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.

C.3.16 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 will 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.

C.3.17 Changes in release 4.0.3 (26 Aug 2002: Beta)

- Fixed problem with types of user variables. (Bug #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 server 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 will still work until 5.0. See Section 2.5.3 [Upgrading-from-3.23], page 131.
- 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 13.4.2 [Mathematical functions], page 577.
- 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)`.

C.3.18 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 13.8.2 [Encryption functions], page 608.
- Added AES_ENCRYPT() and AES_DECRYPT() functions to perform encryption according to AES standard (Rijndael). See Section 13.8.2 [Encryption functions], page 608.
- Added --single-transaction option to mysqldump, allowing a consistent dump of InnoDB tables. See Section 8.8 [mysqldump], page 477.
- 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 will work 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 will make 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 13.7 [Cast Functions], page 605.
- `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 5.5.5 [User resources], page 307.
- Added `mysql_secure_installation` to the '`scripts/`' directory.

C.3.19 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(string)` now returns the characters in `string` 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 will stop (and possibly roll 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.

C.3.20 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 will extend `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 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).

C.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 2.5.3 [Upgrading-from-3.23], page 131.

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.x version. Only bugfixes and minor enhancements to existing features will be added to 3.23.

The replication code and BerkeleyDB code is still not as tested and as the rest of the code, so we will probably need to do a couple of future releases of 3.23 with small fixes for this part of the code. As long as you don't use these features, you should be quite safe with MySQL 3.23!

Note that the preceding remarks don't mean that replication or Berkeley DB don't work. We have done a lot of testing of all code, including replication and `BDB` without finding any problems. It only means that not as many users use this code as the rest of the code and because of this we are not yet 100% confident in this code.

C.4.1 Changes in release 3.23.59 (not released yet)

- Fixed a symlink vulnerability in '`mysqlbug`' script - vulnerability id CAN-2004-0381. (Bug #3284)
- Fixed bug in privilege checking of `ALTER TABLE RENAME`. (Bug #3270)
- Fixed bugs in `ACOS()`, `ASIN()` (Bug #2338) and in `FLOOR()` (Bug #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)
- 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)

- `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`..) already behave. Note that one now has to use `mysqlbinlog -p<my_password>`; `mysqlbinlog -p <my_password>` will not work anymore (in other words, put no space after `-p`). (Bug #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, #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)
- 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)

C.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` will never send 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)
- Fixed table/column grant handling: The proper sort order (from most specific to less specific, see Section 5.4.6 [Request access], page 287) was not honored. (Bug #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)
- 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)

C.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)

- SHOW VARIABLES LIKE 'innodb_data_file_path' displayed only the name of the first data file. (Bug #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 16.5 [InnoDB start], page 764.
- 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)
- 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)
- 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)

C.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)
- 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 (e.g. 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.

C.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 C.4.6 [News-3.23.54], page 1105) 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.

C.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_column=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`.

C.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`.

C.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.

C.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 will 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.

C.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 (> 64K) 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`.

C.4.11 Changes in release 3.23.49

- 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.

C.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 will 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.

C.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.

C.4.14 Changes in release 3.23.46 (29 Nov 2001)

- Fixed problem with aliased temporary table replication.

- `InnoDB` and `BDB` tables will 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.

C.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.

C.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)`, will 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.

C.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.

C.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.

C.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 [Server options], page 228.
- 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 #...`

C.4.20 Changes in release 3.23.40

- 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` will now only flush the logs, 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.

C.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` will no longer repair 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 will be 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 4G) when using `ORDER BY`.

C.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 where 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 will use 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.

C.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 already used). 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 thread will now be started 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 will refuse to execute them if the error code indicates the update was terminated abnormally, and will have 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 uses the `tis620` character set must be fixed with `myisamchk -r` or `REPAIR TABLE` !
- Added `--skip-safemalloc` option to `mysqld`.

C.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.

C.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`.

C.4.26 Changes in release 3.23.34a

- Added extra files to the distribution to allow `INNOBASE` support to be compiled.

C.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.
- `REPLACE` will 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 multi-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.

C.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 will enable 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 UDF 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 will 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 multi-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 will be 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 will cause 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.

C.4.29 Changes in release 3.23.32 (22 Jan 2001: Production)

- 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()`.

C.4.30 Changes in release 3.23.31 (17 Jan 2001)

- 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.

C.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 will now automatically start 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.

C.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()` will shorten 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 multi-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 will 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 [mysqld_multi], page 224.
- 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().

C.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], page 456, 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.
- One 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 multi-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`.
- 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 will create the new table with the rows in a specific order.

C.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.

C.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` will be 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 will kill 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.

C.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 multi-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 15.2 [MERGE], page 746.
- 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.

C.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).

C.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 will 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 will make 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 will now only update tables that have been changed since the last ANALYZE TABLE. Note that this is a new feature and tables will not be marked to be analysed 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 will 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.

C.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.

C.4.40 Changes in release 3.23.21

- 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 `AUTOCOMMIT` 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.

C.4.41 Changes in release 3.23.20

- Fixed bug in 3.23.19; `DELETE FROM tbl_name` removed the '`.frm`' file.
- Added `SHOW CREATE TABLE`.

C.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.

C.4.43 Changes in release 3.23.18

- 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 will 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.

C.4.44 Changes in release 3.23.17

- 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 will give 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`.

C.4.45 Changes in release 3.23.16

- 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 will also improve systems where you can't use persistent connections.
- Lots of fixes in the Berkeley DB interface.
- `UPDATE IGNORE` will 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.

C.4.46 Changes in release 3.23.15 (May 2000: Beta)

- 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 [MySQL indexes], page 427.
- `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)`.

C.4.47 Changes in release 3.23.14

- 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 command `REPAIR TABLE`.
- 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.

C.4.48 Changes in release 3.23.13

- 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` will wait 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.

C.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_field 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.

C.4.50 Changes in release 3.23.11

- 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.

C.4.51 Changes in release 3.23.10

- Fixed bug in 3.23.9 where memory wasn't properly freed when using LOCK TABLES.

C.4.52 Changes in release 3.23.9

- 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`.

C.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.

C.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 2.6.1 [Linux], page 141.
- Fixed that one will 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`.

C.4.55 Changes in release 3.23.6

- 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 will 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)`.

C.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 UDF functions.

C.4.57 Changes in release 3.23.4 (28 Sep 1999)

- Inserting a `DATETIME` into a `TIME` column no longer will 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.

C.4.58 Changes in release 3.23.3

- 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.

C.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.

C.4.60 Changes in release 3.23.1

- Fixed some compilation problems.

C.4.61 Changes in release 3.23.0 (05 Aug 1999: Alpha)

- A new storage engine library (MyISAM) with a lot of new features. See Section 15.1 [MyISAM], page 738.
- 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 will describe 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 will ensure 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 UDF 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` will now automatically make 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`.

C.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 2.5.5 [Upgrading-from-3.21], page 137.

C.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.

C.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 :(

C.5.3 Changes in release 3.22.33

- Fixed problems in Windows when locking tables with `LOCK TABLE`.
- Quicker kill of `SELECT DISTINCT` queries.

C.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.

C.5.5 Changes in release 3.22.31

- A few small fixes for the Windows version.

C.5.6 Changes in release 3.22.30

- 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.

C.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.

C.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.

C.5.9 Changes in release 3.22.27

- 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() will now work on Windows.

C.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.

C.5.11 Changes in release 3.22.25

- Fixed some small problems with the installation.

C.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

C.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`.

C.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.

C.5.15 Changes in release 3.22.21

- 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).

C.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.

C.5.17 Changes in release 3.22.19 (Mar 1999: Production)

- 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.

C.5.18 Changes in release 3.22.18

- 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.

C.5.19 Changes in release 3.22.17

- 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.

C.5.20 Changes in release 3.22.16 (Feb 1999: Gamma)

- `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`.

C.5.21 Changes in release 3.22.15

- `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.

C.5.22 Changes in release 3.22.14

- 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 will no lonher crash if your date is > 2039, but instead will return 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 UDF functions in future MySQL releases.

C.5.23 Changes in release 3.22.13

- 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!

C.5.24 Changes in release 3.22.12

- 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 will be 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.

C.5.25 Changes in release 3.22.11

- **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 will 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()`.

C.5.26 Changes in release 3.22.10

- `mysql_install_db` no longer starts the MySQL server! You should start `mysqld` with `safe_mysqld` after installing it! The MySQL RPM will, however, start the server as before.
- Added `--bootstrap` option to `mysqld` and recoded `mysql_install_db` to use it. This will make 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 will show the new status variables.

C.5.27 Changes in release 3.22.9

- SET SQL_LOG_UPDATE=0 caused a lockup of the server.
- New SQL command: FLUSH [TABLES | HOSTS | LOGS | PRIVILEGES] [, ...]
- New SQL command: 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 will get 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 will flush all tables to disk after each update. This makes things much safer on the Windows platforms but also **much** slower.

C.5.28 Changes in release 3.22.8

- Czech character sets should now work much better.

- `DATE_ADD()` and `DATE_SUB()` didn't work with group functions.
- `mysql` will now also try 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 will 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.

C.5.29 Changes in release 3.22.7 (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 will parse 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 get the configure 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 `DATETIME` 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 will make 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.

C.5.30 Changes in release 3.22.6

- Faster `mysqldump`.
- For the `LOAD DATA INFILE` statement, you can now use the new `LOCAL` keyword to read the file from the client. `mysqlimport` will automatically use `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.

C.5.31 Changes in release 3.22.5

- 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`' file names.
- 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 will now ignore 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 [MySQL indexes], page 427.
- Lots of new benchmarks.
- A new C API chapter and lots of other improvements in the manual.

C.5.32 Changes in release 3.22.4

- 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 id.

- 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.

C.5.33 Changes in release 3.22.3

- 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'.

C.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.

C.5.35 Changes in release 3.22.1 (Jun 1998: Alpha)

- 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.

C.5.36 Changes in release 3.22.0

- 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 will be 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 will speed 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 will enable assembler functions for the most important string functions for more speed!

C.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.

C.6.1 Changes in release 3.21.33

- 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.

C.6.2 Changes in release 3.21.32

- 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 ont the same table before the first thread had finished. A refresh (or `--flush-tables`) will now 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.

C.6.3 Changes in release 3.21.31

- 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.

C.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 'hostname'.err instead of 'hostname'.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.

C.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").

C.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.

C.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.)

C.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.

C.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 UDF 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.

C.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.

C.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.

C.6.12 Changes in release 3.21.22

- Starting with this version, all MySQL distributions will be 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.

C.6.13 Changes in release 3.21.21a

- Configure changes for some operating systems.

C.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.

C.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.

C.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).

C.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.

C.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 will be quicker than the old algorithm when using bigger key caches.

C.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`.

C.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.

C.6.21 Changes in release 3.21.14b

- Fixed a couple of small `configure` problems on some platforms.

C.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_column=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.

C.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.

C.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 standard SQL standard. This will make `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 will give better performance on `SELECT` and a `WHERE` that matches many rows.
- Added Russian error messages.

C.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`.

C.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.

C.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 2.6.5.8 [SCO], page 164.

C.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.

C.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.

C.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()`.

C.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://www.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.

C.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.

C.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 will 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 will make 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 already existed.

- `mysql` now shows the thread ID when starting or doing a reconnect.
- Changed the default sort buffer size from 2MB to 1MB.

C.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.

C.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 will do 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 will provide the full 64-bit range with bit functions and fix 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` will now put `NULL` field values first. `GROUP BY` will also work 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;
```

C.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.

C.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 will make 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.

C.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)`.

C.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.

C.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.

C.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.

C.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`).

C.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.)

C.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.

C.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.

C.7.10 Changes in release 3.20.8

- Fixed bug with `ALTER TABLE` and multi-part keys.

C.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`.

C.7.12 Changes in release 3.20.6

- Portable to more systems because of MIT-pthreads, which will be 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().

C.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 INDEX 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.

C.7.14 Changes in release 3.20.0

- The source distribution is done with configure and Automake. It will make 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 will follow 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.

C.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.

C.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.

C.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.

C.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 will not yet work as expected (probably it will group/sort 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.

C.9 `InnoDB` Change History

C.9.1 MySQL/InnoDB-5.0.0, December 24, 2003

- **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.
- `InnoDB` in 5.0.0 is essentially the same as InnoDB-4.1.1 with the bugfixes of InnoDB-4.0.17 included.

C.9.2 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** carefully 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)
- 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)
- 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) and (Bug #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)
- 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 will 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)

C.9.3 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, e.g., '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.

C.9.4 MySQL/InnoDB-4.1.1, December 4, 2003

- **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.
- 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.

C.9.5 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.

C.9.6 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.

C.9.7 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.

C.9.8 MySQL/InnoDB-4.0.14, July 22, 2003

bullet InnoDB now supports the SAVEPOINT and ROLLBACK TO SAVEPOINT SQL statements. See http://www.innodb.com/ibman.php#Savepoints for the syntax.

bullet You can now create column prefix keys like in CREATE TABLE t (a BLOB, INDEX (a(10))).

bullet 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 will print 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 error, but storing that info 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.

- bullet Fixed a bug: os_event_wait() did not work properly in Unix, which might have caused starvation in various log operations.
- bullet 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.
- bullet 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.

C.9.9 MySQL/InnoDB-3.23.57, June 20, 2003

- bullet 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.
- bullet 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.
- bullet 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.
- bullet Fixed a bug: `GROUP BY` and `DISTINCT` could treat NULL values inequal.

C.9.10 MySQL/InnoDB-4.0.13, May 20, 2003

- bullet `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.
- bullet `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.
- bullet `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 since it is only an estimate, 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`.
- bullet `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.
- bullet 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.
- bullet 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.

bullet If `innodb_force_recovery=6`, do not let `InnoDB` do repair of corrupt pages based on the doublewrite buffer.

bullet `InnoDB` startup now happens faster because it does not set the memory in the buffer pool to zero.

bullet Fixed a bug: The `InnoDB` parser for `FOREIGN KEY` definitions was confused by the keywords 'foreign key' inside MySQL comments.

bullet 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()`.

bullet 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.

bullet 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.

bullet 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.

bullet Fixed a bug: Allow `HANDLER PREV` and `NEXT` also after positioning the cursor with a unique search on the primary key.

bullet 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.

bullet 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.

C.9.11 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.

C.9.12 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.

C.9.13 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.

C.9.14 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.

C.9.15 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.

C.9.16 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 will appear in 4.0.11 and will probably 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.

C.9.17 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.

C.9.18 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 will 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 will become 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.

C.9.19 MySQL/InnoDB-4.0.7, December 26, 2002

- `InnoDB` in 4.0.7 is essentially the same as in 4.0.6.

C.9.20 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.

C.9.21 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 already 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 will become corrupt. From now on `InnoDB` prevents creation of such indexes.

C.9.22 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 inserted 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 already in the root. This could cause unnecessary table scans in SQL queries. The fix will also be 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 will be fixed in 4.0.6.

C.9.23 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.

C.9.24 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.

C.9.25 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.

C.9.26 MySQL/InnoDB-3.23.52, August 16, 2002

- The feature set of 3.23 will be frozen from this version on. New features will go the 4.0 branch, and only bugfixes will be 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 will no longer see 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.

C.9.27 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.

C.9.28 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.

C.9.29 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 will now run 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 already 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.

C.9.30 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.

C.9.31 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 multiprocessor Linux computer. Large CPU-bound SELECT queries will 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.

C.9.32 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 will now run much faster.
- MySQL now knows that the table scan of an `InnoDB` table is done through the primary key. This will save 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.

C.9.33 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.

C.9.34 MySQL/InnoDB-3.23.46, November 30, 2001

- This is the same as 3.23.45.

C.9.35 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.

C.9.36 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 will now work also for `InnoDB` tables.
- Accent characters in the default character set latin1 will be 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` will calculate better table cardinality estimates.
- Change in deadlock resolution: in .43 a deadlock rolls back only the SQL statement, in .44 it will roll 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 will help you in dumping tables from a corrupted database.
- A new my.cnf option innodb_fast_shutdown will speed 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.

C.9.37 MySQL/InnoDB-3.23.43, October 4, 2001

- This is essentially the same as InnoDB-3.23.42.

C.9.38 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.

C.9.39 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.

C.9.40 MySQL/InnoDB-3.23.40, July 16, 2001

- Only a few rare bugs fixed.

C.9.41 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.

C.9.42 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 will reveal disk defects.
- Several bugfixes.

Appendix D Porting to Other Systems

This appendix will help you port MySQL to other operating systems. Do check the list of currently supported operating systems first. See Section 2.1.1 [Which OS], page 60. 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 2.3.5 [MIT-pthreads], page 110.

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 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 likly to be added unless someone has a good reason for it.

If you want/need to change any 'Makefile' or the configure script you will also need GNU Automake and Autoconf. See Section 2.3.3 [Installing source tree], page 104.

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 Section D.1 [Debugging server], page 1199.

Note: Before you start debugging `mysqld`, first get the test programs `mysys/thr_alarm` and `mysys/thr_lock` to work. This will ensure that your thread installation has even a remote chance to work!

D.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 will disable all new, potentially unsafe functionality) or with `--safe-mode` which disables a lot of optimization that may cause problems. See Section A.4.1 [Crashing], page 1013.

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` will dump 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 [MySQL Database Administration], page 218. 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 2.6 [Operating System Specific Notes], page 141.

D.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 will avoid 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 will install 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 will disable the memory overrun checks for each call to `malloc()` and `free()`.

If `mysqld` stops crashing when you compile it with `--with-debug`, you have probably 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` now 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 will be 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 1.7.1.1 [Mailing-list], page 32. 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.

D.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 will 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://support.mysql.com/pub/mysql/secret/` 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 D.3 [The DBUG package], page 1205.

D.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!

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 will install 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'` will help 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.4.1 [Crashing], page 1013.

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 1.7.1.1 [Mailing-list], page 32.

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.

D.1.4 Using a Stack Trace

On some operating systems, the error log will contain a stack trace if `mysqld` dies unexpectedly. You can use this to find out where (and maybe why) `mysqld` died. See Section 5.8.1 [Error log], page 345. To get a stack trace, you must not compile `mysqld` with the `-fomit-frame-pointer` option to gcc. See Section D.1.1 [Compiling for debugging], page 1199.

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) already 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 will print 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 will 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 1.7.1.3 [Bug reports], page 34.

D.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 [MySQL Database Administration], page 218.

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 'hostname'.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], page 402.

You can find the queries that take a long time to execute by starting `mysqld` with `--log-slow-queries`. See Section 5.8.5 [Slow query log], page 350.

If you find the text `mysqld restarted` in the error log file (normally named '`hostname.err`') you have probably found a query that causes `mysqld` to fail. If this happens, you should check all your tables with `myisamchk` (see Chapter 5 [MySQL Database Administration], page 218), 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 will automatically check and try to repair `MyISAM` tables if they are marked as 'not closed properly' or 'crashed'. If this happens, MySQL will write 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 [Server options], page 228.

It's not a good sign if `mysqld` did died unexpectedly, but in this case one shouldn't investigate the `Checking table...` messages but instead try to find out why `mysqld` died.

D.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 will do 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 5.8.4 [Binary log], page 347. 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://support.mysql.com/pub/mysql/secret/` and enter it into our bugs system at `http://bugs.mysql.com/`. If you are a support customer), you can also `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.

D.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 2.3.2 [`configure` options], page 101.

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 will provide useful information in case you mail a bug report. See Section 1.7.1.3 [Bug reports], page 34.

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.

D.3 The DBUG Package

The MySQL server and most MySQL clients are compiled with the DBUG package originally made 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 D.1.2 [Making trace files], page 1200.

One uses the debug package by invoking the program with the `--debug="..."` or the `-#...` option.

Most MySQL programs has a default debug string that will be used if you don't specify an option to `--debug`. The default trace file is usually `/tmp/programname.trace` on Unix and `\programname.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. That is, `-#D,20` is delay two seconds.
f	Limit debugging and/or tracing, and profiling to the list of named functions. Note that a null list will disable 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 dbug output.
o	Redirect the debugger output stream to the specified file. The default output is stderr.
O	As `o` but the file is really flushed between each write. When needed the file is closed and reopened 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 will occur for either debugging or tracing macros. The default is a compile time option.

Some examples of debug control strings which 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`.

D.4 Locking methods

Currently MySQL only supports table locking for `ISAM`/`MyISAM` and `HEAP` tables, page-level locking for `BDB` tables and row-level locking for `InnoDB` tables. See Section 7.3.1 [Internal locking], page 422. With `MyISAM` tables one can freely mix `INSERT` and `SELECT` without locks, if the `INSERT` statements are non-conflicting (that is, whenever they append to the end of the table file rather than filling freespace from deleted rows/data).

Starting in version 3.23.33, you can analyse the table lock contention on your system by checking `Table_locks_waited` and `Table_locks_immediate` environment variables.

To decide if you want to use a table type with row-level locking, you will want to look at what the application does and what the select/update pattern of the data is.

Pros for row 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.

Cons:

- Takes more memory than page level or table locks.
- Is slower than page level or table locks when used on a big part of the table, because you have to do many more locks.
- Is definitely much worse than other locks if you often do `GROUP BY` on a large part of the data or if you often have to scan the whole table.
- With higher level locks one can also more easily support locks of different types to tune the application as the lock overhead is less notable as for row level locks.

Table locks are superior to page level / row level locks in the following cases:

- Mostly reads
- Read and updates on strict keys; this is where one updates or deletes a row that can be fetched with one key read:

```
UPDATE tbl_name SET column=value WHERE unique_key#
DELETE FROM tbl_name WHERE unique_key=#
```

- `SELECT` combined with `INSERT` (and very few `UPDATE` and `DELETE` statements).
- Many scans / `GROUP BY` on the whole table without any writers.

Other options than row / page level locking:

Versioning (like 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 one started to access it. Other names for this are time travel, copy on write or copy on demand.

Copy on demand is in many case much better than page or row level locking; the worst case does, however, use much more memory than when using normal locks.

Instead of using row level locks one can use application level locks (like get_lock/release_lock in MySQL). This works only in well-behaved applications.

In many cases one can do an educated guess which locking type is best for the 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 part of the application may require different lock types.

Here are some tips about locking in MySQL:

Most Web applications do lots of selects, very few deletes, updates mainly on keys, and inserts in some specific tables. The base MySQL setup is very well tuned for this.

Concurrent users are not a problem if you don't mix updates with selects that need to examine many rows in the same table.

If you mix inserts and deletes on the same table, then `INSERT DELAYED` may be of great help.

You can also use `LOCK TABLES` to speed up things (many updates within a single lock is much faster than updates without locks). Splitting thing to different tables will also help.

If you encounter speed problems with the table locks in MySQL, you may be able to solve these by converting some of your tables to `InnoDB` or `BDB` tables. See Chapter 16 [InnoDB], page 758. See Section 15.4 [BDB], page 751.

The optimization section in the manual covers many different aspects of how to tune applications. See Section 7.2.14 [Tips], page 419.

D.5 Comments about RTS threads

I have tried to use the RTS thread packages with MySQL but stumbled on the following problems:

They use an old version of a lot of 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 already 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
```

D.6 Differences between different 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 E Environment Variables

Here is a list of 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 [Option files], page 212.

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 5.5.7 [Password security], page 310.
`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 [Timezone problems], page 1019.
`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 F 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], page 1029. 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], page 188.

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 meaning inside a `[]` pair has no special meaning and 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 ‘`regexp/cname.h`’.

```
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 G GNU General Public License

Version 2, June 1991

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.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. 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.

1. 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.

2. 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.

3. 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.

4. 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.

5. 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.

6. 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.
7. 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.
8. 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.
9. 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.

10. 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

11. 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.
12. 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.

SQL command, type and function index

A

B

C

D

E

N

O

P

Q

R

U

V

W

X

Y

Concept Index

-

A

B

C

D

E

F

G

H

I

J

K

L

M

Q

R

S

T

U

V

W

X

Y