13

Administering MySQL

In previous chapters, you learned how to install MySQL on Linux and Windows operating systems, maneuver through the MySQL installation, implement relational databases in the MySQL environment, and execute SQL statements that insert, update, delete, and retrieve data from those databases. In this chapter and the next several chapters, you learn how to perform administrative tasks that allow you to determine how MySQL runs, who can access the MySQL server, and how to replicate your system and prepare for database disaster.

This chapter introduces you to MySQL administration and describes how to modify default MySQL settings, set system variables, and implement logging. Chapter 14 explains how to manage security, and Chapter 15 describes how to optimize performance. In Chapter 16, you learn how to back up and restore your MySQL databases as well as set up replication. This chapter, then, provides the starting point from which you begin to perform basic administrative tasks. Specifically, the chapter covers the following topics:

- ❑ Using the mysqladmin client utility to verify system settings and perform server-related operations
- ❑ Using the `SHOW VARIABLES`, `SELECT`, and `SHOW STATUS` statements to retrieve system variable settings and using start-up options, option files, and the `SET` statement to set system variables
- ❑ Managing error, query, and binary logging

Performing Administrative Tasks

Once you install MySQL, you might find that you want to view information about the MySQL server, such as status or version information. You might also find that you want to perform administrative tasks such as stopping the server or flushing the host's cache. To allow you to perform these tasks, the MySQL installation includes the mysqladmin client utility, which provides an administrative interface to the MySQL server. Using this tool, you can perform a variety of administrative tasks, such as obtaining information about the MySQL configuration, setting passwords, starting the server, creating and dropping databases, and reloading privileges.

You use the mysqladmin client utility at a command prompt, as you use the mysql client utility. When launching mysqladmin, you can specify zero or more program options and one or more commands, as shown in the following syntax:

```
mysqladmin [<program option> [<program option>...]]
<command> [<command option>] [{<command> [<command option>]}...]
```

The program options define how to connect to the MySQL server and how to run the mysqladmin utility. For example, you can use program options to specify the user account to use when connecting to the MySQL server, the password for that account, and the port to use when establishing the connection.

Many of the program options provide two forms that you can use when specifying that option. For example, the `--force` option can also be specified as the `-f` option. The following table describes many of the program options that you can use with the mysqladmin utility and, when applicable, provides both the long and short form of the option. (When two options are provided, they're separated by a vertical pipe [|].) Note, however, that program options can vary from one release of MySQL to the next, so it's always a good idea to check the latest MySQL product documentation for current information about options as well as information about other options available to mysqladmin. (For additional information about specifying program options, see Chapter 3.)

Option syntax	**Description**
`--character-set-dir=<path>`	Specifies the directory where the character sets are stored.
`--compress` | `-C`	Compresses the data that is sent between the server and the client. Both the server and client must support compression.
`--connect_timeout=<seconds>`	Specifies the number of seconds to wait before timing out when trying to connect to the server.
`--count=<number>` | `-c <number>`	Specifies the number of iterations to make when the `--sleep` option is used to execute the mysqladmin command repeatedly.
`--force` | `-f`	Causes mysqladmin to execute the `drop <database>` command without asking for confirmation. Also forces the execution of multiple mysqladmin commands even if an error is caused by one of the commands. (Normally, mysqladmin would exit if one of the specified commands caused an error.)
`--help` | `-?`	Displays information about the options available to mysqladmin.
`--host=<host name>` | `-h <host name>`	Specifies the name of the host where the MySQL server is running.

Option syntax	Description
`--password[=<password>] | -p[<password>]`	Specifies the password to use when connecting to the MySQL server. When using the `-p` alternative to specify a password, the password must follow immediately after the option. (There should be no space.) For either option, if the password is not specified, MySQL prompts you for a password.
`--port=<port number> | -P <port number>`	Specifies the port number to use when connecting to the MySQL server.
`--protocol= {TCP | SOCKET | PIPE | MEMORY}`	Specifies the connection protocol to use when connecting to the MySQL server.
`--silent | -s`	Exits without providing output if mysqladmin cannot connect to the server.
`--sleep=<seconds> | -i <seconds>`	Specifies that the mysqladmin command (along with any defined options) should be executed repeatedly. The specified number of seconds represents the length of delay between executions. The `--sleep` command can be used in conjunction with the `--count` command.
`--user=<username> | -u <username>`	Specifies the user account to use when connecting to the MySQL server.
`--version | -V`	Displays version number and other information about mysqladmin as well as information about the MySQL server.
`--wait[=<number>] | -w[<number>]`	Retries to connect to the MySQL server if a connection cannot be established. The optional *`<number>`* placeholder indicates the number of times to retry. If no value is provided, a 1 is assumed. When using the `-w` alternative to specify this option, there should be no space between the `-w` and the number.

In addition to specifying program options when using the mysqladmin client utility, you must specify one or more commands. The commands define the actions that mysqladmin should take when the mysqladmin program is executed. For example, you can verify whether the MySQL server is running or reload the grant tables. In some cases, a command takes a command option. It is usually evident when an option is required. For instance, you can use mysqladmin to create a database. To do so, you must supply a name for the new database. The database name is the command option.

The following table describes many of the commands that you can use with the mysqladmin utility. As with program options, commands can vary from one release of MySQL to the next, so be sure to check the latest MySQL product documentation for current information about a command as well as information about other commands available to mysqladmin.

Command syntax	**Description**
`create <database >`	Creates a new database with the specified name.
`drop <database>`	Removes a database with the specified name.
`extended-status`	Displays the names and values of the server status variables.
`flush-hosts`	Flushes the host's cache.
`flush-logs`	Closes and then reopens the log files. For some log file types, MySQL creates a new file.
`flush-privileges`	Reloads the grant tables. This command is equivalent to the reload command.
`flush-status`	Clears the status variables and resets several counters to zero.
`flush-tables`	Flushes the table cache.
`flush-threads`	Flushes the thread cache.
`kill <thread ID> [<thread ID>...]`	Kills the specified threads. Each thread is associated with a single connection, so when you kill a thread, you're terminating that connection. You can use the `processlist` command to display a list of active threads.
`password <new password>`	Assigns a password to the MySQL user account currently being used to invoke mysqladmin. If a password already exists for the user account, the new password is applied *after* the execution of the mysqladmin command, which means that you must supply the original password for the `--password` program option.
`ping`	Verifies whether the MySQL server is running.
`processlist`	Displays a list of the active server threads. Each thread is assigned an ID, which you can use along with the `kill` command to terminate the thread.
`reload`	Reloads the grant tables. This command is equivalent to the `flush-privileges` command.
`refresh`	Flushes the table cache, closes and then reopens the log files, and reloads the grant tables.
`shutdown`	Stops the MySQL server.
`start-slave`	Starts replication on the slave server.

Command syntax	Description
`status`	Displays current status information about the MySQL server.
`stop-slave`	Stops replication on the slave server.
`variables`	Displays the names and values of the server variables.
`version`	Displays version information about the current MySQL server.

Now that you have an overview of the program options and commands that you can use with the mysqladmin client utility, take a look at a few examples that demonstrate the various tasks that you can perform by using this tool. The first example demonstrates how to use the `--help` program option to retrieve information about the options and commands that you can use with the mysqladmin utility:

```
mysqladmin --help
```

As you can see, you need to specify only the mysqladmin utility name and the `--help` program option. Even if a username and password are required to connect to the MySQL server, you do not have to specify them here, because the purpose of the `--help` option is to allow you to find the information that you need in order to use the tool for other tasks. When you execute the command, you receive information similar to the following:

```
mysqladmin  Ver 8.40 Distrib 4.1.5-gamma, for Win95/Win98 on i32
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

Administration program for the mysqld daemon.
Usage: mysqladmin [OPTIONS] command command....
  -c, --count=#       Number of iterations to make. This works with -i
                      (--sleep) only.
  -#, --debug[=name]  Output debug log. Often this is 'd:t:o,filename'.
  -f, --force         Don't ask for confirmation on drop database; with
                      multiple commands, continue even if an error occurs.
  -C, --compress      Use compression in server/client protocol.
  --character-sets-dir=name
                      Directory where character sets are.
  -?, --help          Display this help and exit.
  -h, --host=name     Connect to host.
  -p, --password[=name]
```

The results shown here represent only a part of what is returned when you specify the `--help` option. As you can see, each program option is listed here, along with a brief description of how it is used. The `--help` command also lists and describes the commands that are available to mysqladmin.

The next example that you look at combines program options and a command, as shown in the following:

```
mysqladmin -u root -p version
```

In this case, the `mysqladmin` command includes the `-u` program option followed by the name of the user account, which is root. Next comes the `-p` program option. The `-p` program option is different from other program options that require a value because it allows you to specify that value separately from the main `mysqladmin` command. When you specify `-p`, without a password value, you're prompted for a password when you execute the command. When you then enter your password, each letter of the password appears as an asterisk at the command line, rather than the actual character. MySQL recommends this method as a way to keep your password secure.

In addition to the two program options, the preceding example also includes the `version` command. The `version` command displays version information about the MySQL server, as shown in the following results:

```
mysqladmin  Ver 8.40 Distrib 4.1.6-gamma, for Win95/Win98 on i32
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          4.1.6-gamma-nt
Protocol version        10
Connection              localhost via TCP/IP
TCP port                3306
Uptime:                 2 hours 15 min 1 sec

Threads: 1  Questions: 24  Slow queries: 0  Opens: 26  Flush tables: 1  Open tab
les: 0  Queries per second avg: 0.003
```

As you can see, the results include version-related information, plus information about the protocol version, the connection, the port used to connect to MySQL, and the amount of time that the MySQL server has been running. The `version` command also returns status-related information, which is described in the following table.

Status variable	Description
Threads	The number of currently open connections.
Questions	The number of queries that have been sent to the server since the server has been running.
Slow queries	The number of queries that have taken more time to execute than the time specified in the `long_query_time` system variable.
Opens	The number of tables that have been opened since the server has been running.
Flush tables	The number of flush, refresh, and reload commands that have been executed since the server has been running.
Open tables	The number of current open tables. These are tables that are currently being accessed.
Queries per second avg	The average number of queries per second.

As you may recall from earlier in this chapter, you can use a long form or a short form to specify many of the program options. In the previous example, the short form of each program option is used. You can achieve the same effect, though, by using the long form, as shown in the following command:

```
mysqladmin --user=root --password version
```

As you can see, the command uses `--user=root` instead of `-u root` and `--password` instead of `-p`.

In addition to executing mysqladmin commands that return information, you can execute commands that take a specific action. For instance, the following mysqladmin example includes the reload command:

```
mysqladmin -u root -p reload
```

The `reload` command reloads the grant tables so that any changes made to the grant tables are immediately applied. The grant tables determine who can access the MySQL server and what type of access they have. (For more information about the grant tables and MySQL security, see Chapter 14.)

The mysqladmin client utility also allows you to specify multiple commands when running the utility. For example, the following command reloads the grant table and returns version-related information:

```
mysqladmin -u root -p reload version
```

When you execute this command, MySQL reloads the grant tables and returns the same information that you saw in an earlier example, when only the version command was specified.

Now that you have seen examples of how to use the mysqladmin client utility to perform administrative tasks, you can test the tool for yourself. In the following exercise, you try out the mysqladmin client utility by viewing the MySQL server's current status and process list. You then use the utility to create a database and then delete the database from your system

Try It Out Using mysqladmin to Administer MySQL

The following steps describe how to use the mysqladmin client utility to perform several administrative tasks:

1. The first command that you create returns current status information about the MySQL server. At your operating system's command prompt, execute the following command:

```
mysqladmin -u root -p status
```

When prompted, type in your password, and then press Enter. You should receive status information about the MySQL server, similar to the following:

```
Uptime: 8859  Threads: 1  Questions: 25  Slow queries: 0  Opens: 26  Flush table
s: 1  Open tables: 0  Queries per second avg: 0.003
```

2. The next command that you create returns status information and a process list. At your operating system's command prompt, execute the following command:

```
mysqladmin -u root -p status processlist
```

When prompted, type your password, and then press Enter. You should receive status information about the MySQL server, followed by a list of active server threads, similar to the following:

```
Uptime: 8870  Threads: 1  Questions: 26  Slow queries: 0  Opens: 26  Flush table
s: 1  Open tables: 0  Queries per second avg: 0.003
+----+------+----------------+----+---------+------+-------+------------------+
| Id | User | Host           | db | Command | Time | State | Info             |
+----+------+----------------+----+---------+------+-------+------------------+
| 18 | root | localhost:2695 |    | Query   | 0    |       | show processlist |
+----+------+----------------+----+---------+------+-------+------------------+
```

3. The next command is similar to the previous, except that the order of the status and processlist options is reversed. At your operating system's command prompt, execute the following command:

```
mysqladmin -u root -p processlist status
```

When prompted, type in your password, and then press Enter. You should receive a list of active server threads, followed by status information about the MySQL server, similar to the following:

```
+----+------+----------------+----+---------+------+-------+------------------+
| Id | User | Host           | db | Command | Time | State | Info             |
+----+------+----------------+----+---------+------+-------+------------------+
| 19 | root | localhost:2698 |    | Query   | 0    |       | show processlist |
+----+------+----------------+----+---------+------+-------+------------------+
Uptime: 8880  Threads: 1  Questions: 29  Slow queries: 0  Opens: 26  Flush table
s: 1  Open tables: 0  Queries per second avg: 0.003
```

4. The next command creates a database named db1. At your operating system's command prompt, execute the following command:

```
mysqladmin -u root -p create db1
```

When prompted, type your password, and then press Enter. You should be returned to your operating system's command prompt.

5. To view whether the database has been created, open the mysql client utility. At your operating system's command prompt, execute the following command:

```
mysql
```

The mysql client utility is launched.

6. Next, view a list of databases that currently exist on your system. At the mysql command prompt, execute the following command:

```
SHOW DATABASES;
```

At a minimum, your results should include the db1, mysql, and test tables, as shown in the following result set:

```
+----------+
| Database |
+----------+
| db1      |
| mysql    |
| test     |
+----------+
3 rows in set (0.00 sec)
```

7. Now close the mysql client utility by executing the following command:

```
exit
```

   You should be returned to your operating system's command prompt.

8. The final command that you create removes the db1 database from your system. At your operating system's command prompt, execute the following command:

```
mysqladmin -u root -p -f drop db1
```

   When prompted, type your password, and then press Enter. You should receive a message indicating that the db1 database has been dropped.

How It Works

In this exercise, you created a series of commands to perform various administrative tasks. The commands are based on the mysqladmin client utility. The first administrative task that you performed was to use the following command to return the current MySQL server status:

```
mysqladmin -u root -p status
```

The `mysqladmin` command includes several options. The first option (`-u`) specifies that the root user account should be used to connect to MySQL. The second option (`-p`) specifies that a password should be used. When you specify the `-p` option in this way, without supplying the password, you're prompted for the password when you execute the command. Note that you can use `--user=root` instead of `-u` root and `--password` instead of `-p`.

The third option included in the preceding example is the `status` command. The `status` command displays the current status information about the MySQL server. This includes such information as uptime, active threads, number of queries that have been processed, and other status-related data. The status-related information is, for the most part, similar to the status information returned by the `version` command. The main difference is that, for the `version` command, the uptime is displayed in hours, minutes, and seconds. For the `status` command, the uptime is displayed in seconds.

The next `mysqladmin` command that you created was similar to the last command, except that it also included the `processlist` option, as shown in the following command:

```
mysqladmin -u root -p status processlist
```

The `processlist` option displays a list of the active server threads. Because the command includes the `status` and `processlist` options, information returned by both options is displayed in the results, in the order that the options are specified in the commands. If you reverse the options, as you did in the following command, the results are reversed:

```
mysqladmin -u root -p processlist status
```

Now the results first display the process list and then display the status information.

The next command that you created adds the db1 database to your system, as shown in the following `mysqladmin` command:

```
mysqladmin -u root -p create db1
```

The command includes the `create` option, followed by the name of the new database (db1). After you executed this command, you opened the mysql client utility and viewed a list of databases to verify that the database has been added. You then exited from the mysql utility and used the following command to remove the database:

```
mysqladmin -u root -p -f drop db1
```

The command includes the `drop` option and the name of the database. As a result, the db1 database is removed from your system. Notice that the command also includes the `-f` option. This is also referred to as the `--force` option, which causes mysqladmin to execute the `drop` command without asking for confirmation. As a result, the db1 database is removed from the system as soon as you execute the command, and you're not prompted to confirm the command.

Managing System Variables

When MySQL starts up, it uses default and user-defined settings that determine how the MySQL server runs. To a great degree, these settings determine the environment in which MySQL operates and how users can connect to the system. The settings are stored in system variables that specify such details as connection timeouts, where the data directory is located, whether logging is enabled, the maximum number of connections allowed, cache sizes, and numerous other settings.

All system variables are configured with default settings. You can override default variable settings by specifying new settings at server startup in command-line options or in option files. You can also specify a number of variable settings at runtime by using `SET` statements. (You learn more about these three options later in the chapter.)

MySQL system variables are either global variables or session variables. Global system variables apply to the entire server operation. Session system variables apply to specific client connections. When the server starts, it initializes all global system variables to their default and, when applicable, user-defined settings (those settings specified at startup). When a connection is established, MySQL initializes a set of session variables for that connection.

Most session variables correspond to global variables. The values initially assigned to those session variables are based on the current values of their global counterparts at the time the connection is established. For example, if the `wait_timeout` global variable is set to 600 when a connection is established, the `wait_timeout` session variable associated with that connection is set to 600. (The `wait_timeout` system variable specifies the number of seconds to wait for activity on a connection before closing that connection.) Some session variables do not correspond to global variables. In this case, MySQL assigns a default value to these session variables.

Not all session variables are the same in terms of the function they perform or whether they can be updated. MySQL supports the following three types of system variables:

- **Server system variables:** A set of system variables that determines how MySQL should run and how clients should interact with the server. Essentially, server system variables refer to all system variables that are not explicitly used to return status information.

- ❑ **Dynamic system variables:** A subset of server system variables whose settings can be modified at runtime.
- ❑ **Server status variables:** A set of system variables whose primary purpose is to provide status-related information about the server's operation. All status variables are considered global variables.

MySQL provides different methods for viewing the three types of system variables. In the next section, you learn how to view each type of variable and its current settings.

Retrieving System Variable Settings

The method that you use to view system variable settings depends on the type of system variable. For server system variables, you use a `SHOW VARIABLES` statement. For dynamic system variables, you use a `SELECT` statement. And for server status variables, you use a `SHOW STATUS` statement.

MySQL supports hundreds of system variables, many more than can be described here. As a result, whenever you need information about a specific system variable or you want to determine what variables are supported, you're encouraged to use one of the statements described in the following sections and to refer to the MySQL product documentation.

Using the SHOW VARIABLES Statement to Retrieve Server System Variable Settings

The `SHOW VARIABLES` statement allows you to view most of the server system variables and their current settings from within the mysql client utility. (The statement does not display some dynamic system variables. For those, you must use `SELECT` statements, which are described later in the chapter.) You can view global or session system variables, and you can qualify your `SHOW VARIABLES` statement by adding a `LIKE` clause, as shown in the following syntax:

```
SHOW [GLOBAL | SESSION] VARIABLES [LIKE '<value>']
```

As you can see, the only required elements in this statement are the `SHOW VARIABLES` keywords. You can also specify either the `GLOBAL` keyword or the `SESSION` keyword. If you specify `GLOBAL`, MySQL displays global system variables. If you specify `SESSION`, MySQL displays session system variables. If you don't specify either option, session variables are displayed if they exist, and global variables are displayed if session variables don't exist.

The `SHOW VARIABLES` syntax also supports the optional `LIKE` clause. The `LIKE` clause allows you to specify an exact value in order to return a specific system variable or to specify an approximate value (by using wildcards) in order to return system variables with names that contain a certain pattern.

Later in this section, you see an example of how to use the `LIKE` clause, but first take a look at a simple example of a `SHOW VARIABLES` statement. The following statement returns all global variables and their settings:

```
SHOW GLOBAL VARIABLES;
```

As you can see, the statement includes the required `SHOW VARIABLES` keywords and the optional `GLOBAL` keyword. When you execute this statement, you should receive results similar to the following:

```
+---------------------------------+-----------------------------------------------+
| Variable_name                   | Value                                         |
+---------------------------------+-----------------------------------------------+
| back_log                        | 50                                            |
| basedir                         | C:\Program Files\MySQL\MySQL Server 4.1\      |
| binlog_cache_size               | 32768                                         |
| bulk_insert_buffer_size         | 8388608                                       |
| character_set_client            | latin1                                        |
| character_set_connection        | latin1                                        |
| character_set_database          | latin1                                        |
| character_set_results           | latin1                                        |
| character_set_server            | latin1                                        |
| character_set_system            | utf8                                          |
```

The results shown here represent only a partial list of the server system variables. Normally, the statement returns nearly 180 system variables. If you want to return the session system variables, you would use the following statement:

```
SHOW SESSION VARIABLES;
```

The results returned by this statement are normally similar to the results returned by the statement when the `GLOBAL` keyword is used. As you can see, you most likely want to trim down the number of rows that are returned by these statements, which is where the `LIKE` clause comes in. The `LIKE` clause allows you to specify a value that is compared to the variable names. For example, suppose that you want to view all variables that are related to database queries. You might want to specify a value in the `LIKE` clause that includes the string *query,* as shown in the following statement:

```
SHOW VARIABLES LIKE '%query%';
```

As you can see, the `LIKE` clause specifies the string, surrounded by the percentage (%) wildcards. As a result, the variable name can begin or end with any character, as long as query is included somewhere in the name. When you execute this statement, you should receive results similar to the following:

```
+------------------------------+---------+
| Variable_name                | Value   |
+------------------------------+---------+
| ft_query_expansion_limit     | 20      |
| have_query_cache             | YES     |
| long_query_time              | 10      |
| query_alloc_block_size       | 8192    |
| query_cache_limit            | 1048576 |
| query_cache_min_res_unit     | 4096    |
| query_cache_size             | 0       |
| query_cache_type             | ON      |
| query_cache_wlock_invalidate | OFF     |
| query_prealloc_size          | 8192    |
+------------------------------+---------+
10 rows in set (0.00 sec)
```

The table displays the variables and their settings only for those variables whose name contains *query*. For information about the meaning of each variable, check out the MySQL product documentation. Now take a look at how a `SELECT` statement is used to retrieve the settings of a dynamic system variable.

Using the SELECT Statement to Retrieve Dynamic System Variable Settings

Like all system variables, MySQL supports dynamic system variables that are either global or specific to a session. Many dynamic system variables exist as both, while others are either global or session, but not both. For example, the `autocommit` variable (which specifies the autocommit mode) is session specific, but the `binlog_cache_size` variable (which specifies the size of the cache to use for the binary logging process) is global and does not apply to the individual session. The `wait_timeout` variable applies at a global level as well as at a session level, however, and the settings for each can be different.

You can use a `SELECT` statement to retrieve the current setting for a dynamic session variable, whether it applies globally or to a session. For individual variables, this is an easier method to use than the `SHOW VARIABLES` statement. For session variables that are not returned by `SHOW VARIABLES` statement, however, this is the only method available for viewing dynamic variable settings.

The following syntax shows how to use a `SELECT` statement to retrieve a dynamic variable setting:

```
SELECT @@[global.]<variable> [{, @@[global.]<variable>}...]
```

As you can see, you must specify the `SELECT` keyword, the double at symbols (@@), and the variable name. If you're retrieving the value of a global variable, you must also precede the variable name with the global keyword, followed by a period, as shown in the following example:

```
SELECT @@global.max_binlog_size;
```

In this `SELECT` statement, a value is retrieved from the `global max_binlog_size` variable, a variable that determines the maximize size in bytes to which a binary log can grow. When you execute this statement, you should receive results similar to the following:

```
+--------------------------+
| @@global.max_binlog_size |
+--------------------------+
|               1073741824 |
+--------------------------+
1 row in set (0.00 sec)
```

Because this variable is one of the system variables returned by the `SHOW VARIABLES` statement, you can also use that statement to retrieve the same results, as shown in the following statement:

```
SHOW GLOBAL VARIABLES LIKE 'max_binlog_size';
```

As you can see, the `LIKE` clause includes a specific value and not wildcards. The statement produces results similar to the following:

```
+-----------------+------------+
| Variable_name   | Value      |
+-----------------+------------+
| max_binlog_size | 1073741824 |
+-----------------+------------+
1 row in set (0.00 sec)
```

Although the results appear in a different form from those returned by the SELECT statement, the information is essentially the same. Remember, however, that this method works only for those system variables that are returned by the SHOW VARAIBLES statement.

If you want to retrieve a value for a session variable, rather than a global variable, you simply omit the global keyword and the period, as shown in the following example:

```
SELECT @@tx_isolation;
```

In this case, the SELECT statement retrieves the value of the tx_isolation session variable, which determines the current connection's isolation level, as shown in the following results:

```
+-----------------+
| @@tx_isolation  |
+-----------------+
| REPEATABLE-READ |
+-----------------+
1 row in set (0.01 sec)
```

As you can see, the isolation level is REPEATABLE READ. (For more information about isolation levels, see Chapter 12.)

If you refer back to the preceding syntax, notice that you can specify more than one dynamic system variable in your SELECT statement. For example, the following SELECT statement returns the isolation level for the current session and the maximum permitted size of the binary log file:

```
SELECT @@tx_isolation, @@global.max_binlog_size;
```

When you execute this statement, the settings from both variables are displayed, as shown in the following results:

```
+-----------------+--------------------------+
| @@tx_isolation  | @@global.max_binlog_size |
+-----------------+--------------------------+
| REPEATABLE-READ |               1073741824 |
+-----------------+--------------------------+
1 row in set (0.00 sec)
```

The SELECT statement, like the SHOW VARIABLES statement, is a handy method for viewing system variables; however, neither method displays the settings of the server status variables. For that, you need the SHOW STATUS statement.

Using the SHOW STATUS Statement to Retrieve Server Status Variable Settings

Most server status variables are essentially counters set to zero when the MySQL server is started. From there, they count the number of times that a particular event has occurred. For example, the Connections status variable provides a total of the number of connections that have been attempted since the server was started, and the Bytes_sent variable tracks the total number of bytes sent to all clients. In addition to the status variables that count events, some status variables provide other types of information, such as the amount of memory available for the query cache or the number of open tables. To view the current status variable settings, you can use the SHOW STATUS statement, which is shown in the following syntax:

```
SHOW STATUS [LIKE '<value>']
```

As the syntax shows, you must include the SHOW STATUS keywords. In addition, you can include a LIKE clause. The LIKE clause in this statement works the same way as the LIKE clause in the SHOW VARIABLES statement, as you see later in this section.

Returning now to the syntax, you can see that a basic SHOW STATUS statement is very straightforward, as shown in the following example:

```
SHOW STATUS;
```

When you execute this statement, all status variables and their current settings are returned. The following result set shows part of the results that you can expect from the SHOW STATUS statement:

```
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| Aborted_clients          | 0     |
| Aborted_connects         | 0     |
| Binlog_cache_disk_use    | 0     |
| Binlog_cache_use         | 0     |
| Bytes_received           | 993   |
| Bytes_sent               | 879   |
| Com_admin_commands       | 0     |
| Com_alter_db             | 0     |
| Com_alter_table          | 0     |
| Com_analyze              | 0     |
| Com_backup_table         | 0     |
| Com_begin                | 0     |
| Com_change_db            | 0     |
| Com_change_master        | 0     |
| Com_check                | 0     |
| Com_checksum             | 0     |
| Com_commit               | 0     |
| Com_create_db            | 0     |
| Com_create_function      | 0     |
| Com_create_index         | 0     |
| Com_create_table         | 4     |
| Com_dealloc_sql          | 0     |
```

In all, there are more than 150 status variables in MySQL. For this reason, the LIKE clause is a handy tool to use to trim down your results. For example, the following SHOW STATUS statement returns all those status variables whose name contains the string *select*:

```
SHOW STATUS LIKE '%select%';
```

When you execute this statement, you should receive results similar to the following:

```
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Com_insert_select     | 0     |
| Com_replace_select    | 0     |
| Com_select            | 44    |
| Select_full_join      | 0     |
| Select_full_range_join | 0    |
| Select_range          | 0     |
| Select_range_check    | 0     |
| Select_scan           | 42    |
+-----------------------+-------+
8 rows in set (0.00 sec)
```

As you can see, the statement returns only status variables that contain the string *select* somewhere in the variable name.

Now that you have an overview of the various SQL statements that you can use to view system variables and their settings, you can begin to use these statements. In the following exercise, you try out the `SHOW VARIABLES`, `SELECT`, and `SHOW STATUS` statements in order to view the settings assigned to specified system variables.

Try It Out Viewing System Settings

The following steps describe how to use various SQL statements to view the settings assigned to system variables:

1. Open the mysql client utility.
2. Create a `SHOW VARIABLES` statement that returns global variables with the string *log* contained anywhere in the variable name. Execute the following SQL statement at the mysql command prompt:

```
SHOW GLOBAL VARIABLES LIKE '%log%';
```

You should receive results similar to the following:

```
+-------------------------------+------------+
| Variable_name                 | Value      |
+-------------------------------+------------+
| back_log                      | 50         |
| binlog_cache_size             | 32768      |
| expire_logs_days              | 0          |
| innodb_locks_unsafe_for_binlog | OFF       |
| innodb_flush_log_at_trx_commit | 1         |
| innodb_log_arch_dir           |            |
| innodb_log_archive            | OFF        |
| innodb_log_buffer_size        | 1048576    |
| innodb_log_file_size          | 10485760   |
| innodb_log_files_in_group     | 2          |
| innodb_log_group_home_dir     | .\         |
| innodb_mirrored_log_groups    | 1          |
| log                           | ON         |
| log_bin                       | ON         |
| log_error                      | .\ws01.err |
| log_slave_updates              | OFF        |
| log_slow_queries               | OFF        |
| log_update                     | OFF        |
| log_warnings                   | 1          |
| max_binlog_cache_size          | 4294967295 |
| max_binlog_size                | 1073741824 |
| max_relay_log_size             | 0          |
| relay_log_purge                | ON         |
| sync_binlog                    | 0          |
+--------------------------------+------------+
24 rows in set (0.00 sec)
```

3. Create a `SELECT` statement that returns a value for the `max_connections` global system variable. The `max_connections` variable determines the maximum number of simultaneous connections that are permitted. Execute the following SQL statement at the mysql command prompt:

```
SELECT @@global.max_connections;
```

You should receive results similar to the following:

```
+--------------------------+
| @@global.max_connections |
+--------------------------+
|                      100 |
+--------------------------+
1 row in set (0.00 sec)
```

4. Create a `SHOW STATUS` statement that returns status variables with the string *thread* contained anywhere in the variable name. Execute the following SQL statement at the mysql command prompt:

```
SHOW STATUS LIKE '%thread%';
```

You should receive results similar to the following:

```
+------------------------+-------+
| Variable_name          | Value |
+------------------------+-------+
| Delayed_insert_threads | 0     |
| Slow_launch_threads    | 0     |
| Threads_cached         | 0     |
| Threads_connected      | 1     |
| Threads_created        | 1     |
| Threads_running        | 1     |
+------------------------+-------+
6 rows in set (0.00 sec)
```

How It Works

The first SQL statement that you created in this exercise was the following `SHOW VARIABLES` statement:

```
SHOW GLOBAL VARIABLES LIKE '%log%';
```

The statement includes the mandatory `SHOW VARIABLES` keyword along with the `GLOBAL` option. As a result, this statement returns only global variables and their settings. In addition, the statement uses a

LIKE clause to specify that any variables returned must include the string *log* somewhere in the name of the variable. (The % wildcards indicates that zero or more of any characters can precede or follow the string.)

The next SQL statement that you created in this exercise was the following SELECT statement:

```
    SELECT @@global.max_connections;
```

The SELECT statement retrieves the value in the max_connections variable. Because the variable name is prefixed with global, the statement applies to the global variable, rather than the session variable. To view the value for the session variable, you must remove the global prefix.

The next statement that you created in this exercise was the following SHOW STATUS statement:

```
    SHOW STATUS LIKE '%thread%';
```

In addition to including the mandatory SHOW STATUS keywords, the statement includes a LIKE clause that specifies that the status variables returned by the statement must contain the string *thread* somewhere in the name of the variable.

Modifying the Server Configuration

In Chapter 3, you learned how you can specify options when you start up a MySQL program, including the MySQL server. Of these methods, the two most commonly used are specifying options at a command prompt when you start up the program or specifying the options in an option file. Most of the options that you can specify at a command prompt or in an option file are the system variables that affect how the server runs. Once the server is running, you can modify the settings of dynamic system variables by using a SET statement. In this section, you review how to specify options at server startup; then you learn how to use a SET statement to set system variables at runtime.

This section touches only briefly on the subjects of specifying options at a command prompt or in an option file. Chapter 3 covers each topic in far greater detail. The information is included here in order to provide a cohesive overview of setting variable values.

Specifying System Settings at the Command Line

As discussed in Chapter 3, you can specify options when you start the MySQL server. When assigning a value to a variable, you precede the variable name by double dashes (--) and then use an equal sign after the variable name to assign the value. For example, the following command assigns a value to the query_cache_limit system variable:

```
    mysqld --query_cache_limit=1000000
```

The query_cache_limit variable limits the size (in bytes) of a result set that can be cached. Any result set larger than the specified size is not cached. By setting a value for the query_cache_limit at the command line, you're overriding the default limit, which is 1048576 bytes.

You can also specify multiple options at the command line, as shown in the following example:

```
mysqld --query_cache_limit=1000000 --wait_timeout=600
```

When the server starts, the query cache limit is set to 1000000 bytes, and the connection timeout is set to 600. Despite your ability to set these variables and other variables at the command line, the preferable way to specify startup options is to use an option file.

Specifying System Settings in an Option File

The advantage of using an option file over using command-line options is that, with an option file, you have to enter the optional settings only once. With specifying options at a command line, you must specify the options each time you start the server. The only time that specifying options at the command line is useful is when you want your setting to apply only to a single server startup, and not to each server startup. Otherwise, the option file is the better choice.

To specify settings for system variables in an option file, you must add the settings to the `[mysqld]` section of your option file. For example, the following entries demonstrate how to set values for two system variables:

```
[mysqld]
query_cache_limit=1000000
wait_timeout=600
```

When you specify the system variables in an option file, you do not need to precede the variable name with double dashes, as you do when specifying variables in a command line. You do, however, have to use an equal sign and then specify the appropriate value.

When specifying server-related variable settings in a Windows option file, you have to be aware of where the MySQL service looks for an option file. By default, the service looks for the `my.ini` *file in the* `C:\Program Files\MySQL\MySQL Server <version>` *directory for the server-related options when starting the server. The MySQL client utilities look for the* `my.ini` *file in the* `C:\WINDOWS` *directory for client-related settings. One option is to remove the current MySQL service and then re-create the service so that it points to the same option file used by the MySQL client utilities. This way, you need to maintain only one option file. (This process is described in a Try It Out section later in the chapter.) Otherwise, you can maintain two option files.*

Specifying System Settings at Runtime

Once your server is running, you might find that you want to modify the system variable settings, either at a global level or at the session level. If you make the change at the global level, the change affects all clients. However, the change does not affect corresponding session variables for current connections, only for new connections initiated. If you make a change at the session level, the change affects only the connection in which the change has been made. No other connections are affected.

To set a dynamic system variable at runtime, you can use the `SET` statement, which is shown in the following syntax:

```
SET [GLOBAL | SESSION] <variable setting>
```

The statement must include the SET keyword and the variable setting, which usually includes a variable name followed by the equal sign and the new value. If you specify the GLOBAL keyword, MySQL assigns the new value to the specified global variable. If you specify the SESSION or neither of the optional keywords, MySQL assigns the new value to the specified session variable.

For a list of dynamic system variables and whether those variables apply at the global level, at the session level, or both, see the MySQL product documentation.

For example, if you want to set the query_cache_limit variable to 1000000 at the global level, use the following SET statement:

```
SET GLOBAL QUERY_CACHE_LIMIT=1000000;
```

As you can see, the statement includes the SET GLOBAL keywords, the name of the variable, an equal sign, and the new value. When you execute this statement, the new setting is applied at the global level, and all connections are affected. As a result, the query cache limit is now smaller than the default.

To set a session-level variable, you can either specify the SESSION keyword or omit it, as shown in the following example:

```
SET WAIT_TIMEOUT=600;
```

As you can see, the statement sets the wait_timeout variable to 600 seconds, much less than the default of 28800 seconds.

Now that you have an understanding of how to use a SET statement to set the values of dynamic system variables, you can try out one of these statements. In the following exercise, you use a SET statement to set the value of the foreign_key_checks system variable. The variable determines whether foreign key references on InnoDB tables are checked when data is modified. If the value is set to 1, foreign key references are checked. If set to 0, they are not checked. By default, the variable is set to 1.

Try It Out Managing System Settings at Runtime

The following steps describe how to set the value of the foreign_key_checks system variable:

1. Open the mysql client utility.
2. First, use a SELECT statement to determine the current setting of the foreign_key_checks system variable. Execute the following SQL statement at the mysql command prompt:

```
SELECT @@foreign_key_checks;
```

You should receive results similar to the following:

```
+----------------------+
| @@foreign_key_checks |
+----------------------+
|                    1 |
+----------------------+
1 row in set (0.00 sec)
```

3. To turn off foreign key checking, set the `foreign_key_checks` variable to 0. Execute the following SQL statement at the mysql command prompt:

```
SET FOREIGN_KEY_CHECKS=0;
```

   You should receive a message indicating the successful execution of the statement.

4. Once again use a `SELECT` statement to determine the current setting of the `foreign_key_checks` system variable. Execute the same SQL statement that you executed in Step 2. You should receive results similar to the following:

```
+----------------------+
| @@foreign_key_checks |
+----------------------+
|                    0 |
+----------------------+
1 row in set (0.00 sec)
```

5. Next, end your MySQL session by closing the mysql client utility, then start a new session by relaunching the utility.

6. Finally, verify the current `foreign_key_checks` setting one last time. Execute the following SQL statement at the mysql command prompt:

```
SELECT @@foreign_key_checks;
```

   You should receive results similar to the following:

```
+----------------------+
| @@foreign_key_checks |
+----------------------+
|                    1 |
+----------------------+
1 row in set (0.00 sec)
```

How It Works

At several different points in this exercise, you used the following `SELECT` statement to verify the current setting for the `foreign_key_checks` system variable:

```
SELECT @@foreign_key_checks;
```

You can use this method to return a value of a dynamic system variable. As the statement shows, you must include the `SELECT` keyword followed by the variable name. The variable name must be preceded by the double at symbols (@@). In addition, because the variable name is not preceded by the `global` prefix, the value for the session variable is returned.

When you checked the `foreign_key_checks` variable the first time, the value was set to 1. You then used the following `SET` statement to change the value to 0:

```
SET FOREIGN_KEY_CHECKS=0;
```

You then checked the foreign_key_checks setting again and verified that it was now set to 0. From there, you ended your session and opened a new session. When you checked the foreign_key_checks setting this time, you verified that it was now set to 1. When you set the foreign_key_checks value, you did not include the GLOBAL keyword, so the session variable was set. As a result, when you ended your session and then started a new session, the global setting was applied.

Managing Log Files

When you install MySQL, it automatically sets up error log files, whether you install MySQL on Linux or on Windows. You can also implement other types of logging, such as query logging and binary logging. In this section, you learn how to work with error logs, query logs, and binary logs. For information about other types of logging available in MySQL, see the MySQL product documentation.

Note that, by default, all log files are stored in the MySQL data directory for your specific installation.

Working with Error Log Files

The error log is a text file that records information about when the MySQL server was started and stopped, as well as error-related information. You can view the error log through a text editor such as Notepad or Vim. By default, the error log is saved to the data directory. In Linux, the file is saved as *<host>*.err. You can specify a different path and filename by adding the following command to the [mysqld] section of the option file:

```
log-error=<path/filename>
```

In Windows, the error file is saved as mysql.err. You cannot change the path or filename. MySQL also creates a file named *<host>*.err. This file, though, contains only a subset of the information in the mysql.err file and currently is not of much use. Based on MySQL documentation, it appears that the *<host>*.err file might eventually replace the mysql.err file and that you may be able to rename the file and specify a new path, but this is not how the error logging in Windows currently works.

When you view the error file, you see entries similar to the following:

```
c:\program files\mysql\mysql server 4.1\bin\mysqld-nt: ready for connections.
Version: '4.1.5-gamma-nt'  socket: ''  port: 3306  Source distribution
041002 10:33:29  [ERROR] DROP USER: Can't drop user: 'user1'@'%'; No such user
041006  9:04:10  [NOTE] c:\program files\mysql\mysql server 4.1\bin\mysqld-nt:
Normal shutdown

041006  9:04:10  InnoDB: Starting shutdown...
041006  9:04:13  InnoDB: Shutdown completed; log sequence number 0 84629
041006  9:04:13  [NOTE] c:\program files\mysql\mysql server 4.1\bin\mysqld-nt:
Shutdown complete
```

The first two lines shown here are what you would typically see when a server startup is recorded. The information is related specifically to the server and includes such details as the path and server filename, the version, and the port number. The third line shows an error that occurred on October 2, 2004 (041002). In this case, the error results from trying to drop a user that did not exist. Whenever an error occurs, a line is added to the log. The remaining lines are all related to shutting down the server. Whenever you start or stop the server, you see information in the error log similar to what you see here.

Enabling Query and Binary Logging

Unlike error logging, you must enable query and binary logging in order to implement it on your system. This section explains how to implement each of these types of logging and how to view those log files once logging is implemented.

Setting Up Query Logging

The query log, also referred to as the general query log, records all connections to the server, SQL statements executed against the server, and other events such as server startup and shutdown. Query logging is not enabled when you install MySQL. To enable query logging, you should add the following command to the `[mysqld]` section of your option file:

```
log[=<path/filename>]
```

If you do not specify a path and filename, the query log is stored in the data directory and is assigned the name `<host>.log`. Because the query log is a text file, you can use a text editor such as Notepad or Vim to view its contents. MySQL writes queries to the log in the order in which the server receives them. (This might be different from the order in which the statements are executed.) Each query is entered as a line in the log. For example, the following two entries in a query log are for a `SELECT` statement and a `SET` statement:

```
041006 14:59:58      16 Query       SELECT @@foreign_key_checks
041006 15:00:08      16 Query       SET FOREIGN_KEY_CHECKS=0
```

The query log can be especially useful when you want to track who is connecting to the server, where the user is connecting from, and what statements that user is executing. This can be helpful in troubleshooting or debugging a system because it allows you to pinpoint exactly where the statement is originating and how the statement is created. Because the query log records every statement that is executed, in addition to connection and other information, the logs can grow quite large and can affect performance. As a result, you might prefer to use binary logging, rather than query logging.

Setting Up Binary Logging

The binary log stores logged data in a more efficient binary format than the query log and records only statements that update or potentially update data. For example, a `DELETE` statement that affects no rows would be recorded in the log because it could potentially update the data. Like query logging, binary logging is not enabled when you install mysql. To enable binary logging, you should add the following command to the `[mysqld]` section of your option file:

```
log-bin[=<path/filename>]
```

If you do not specify a path and filename, the file is stored in the data directory and is assigned the name `<host>-bin.000001`. If a binary log file already exists, the file extension is incremented by 1, based on the highest file number. In addition, when you set up binary logging, a file named `<host>-bin.index` is created. The index file tracks the binary log files. You can change the name of the index file and its location by adding the following command to the `[mysqld]` section of your option file:

```
log-bin-index[=<path/filename>]
```

If you specify a filename for either the log file or the index file and that filename includes an extension, MySQL automatically drops the extension and adds a numerical extension to the log file and the `.index` extension to the index file.

Because the log files and index files are created as binary files, you should use the mysqlbinlog client utility to view the contents of the file. To use mysqlbinlog, you must specify the utility name, followed by the name of the file, as shown in the following syntax:

```
mysqlbinlog <host>-bin.<numeric suffix>
```

For example, suppose that you want to view the contents of a binary log file named Server21-bin.000001. To do so, you would use the following command:

```
mysqlbinlog Server21-bin.000001
```

MySQL adds information for each event that is tracked by binary logging. For example, the following binary log entry shows that data was inserted in the t1 table:

```
# at 138
#041006  9:09:57 server id 1  log_pos 138  Query  thread_id=1  exec_time=0  error_
SET TIMESTAMP=1097078997;
INSERT INTO t1 VALUES (12);
```

As you can see, the information includes identifying information about the log entry and the statement execution, a timestamp, and the actual INSERT statement. (Note that, in the results shown here, the second line is cut off after error_, but you might see the line wrapped around or cut off at a different point.) Each time an event is recorded, an entry similar to this is added to the log file. If you restart your server or flush the logs, a new log file is created, with the numeric suffix incremented by one. (When you flush logs, MySQL closes and reopens the log files. In the case of the binary log files, a new log file is also created. To flush the logs, you can execute a FLUSH LOG statement or use the flush-logs command of the mysqladmin utility.)

Flushing logs is part of a larger process of log file maintenance. When any type of logging is enabled, the log files can grow quite large. As a result, administrators must often devise a comprehensive maintenance strategy to manage the log files. Planning this sort of maintenance is beyond the scope of this book; however, if you ever become responsible for this task, you should refer to the MySQL product documentation as well as other MySQL-related documentation for information on log flushing, log rotation, age-based expiration, and other aspects of log file maintenance.

Now that you have an overview of how to implement query and binary logging, you can implement logging on your system. The next two Try It Out sections have you try out how to enable query and binary logging on Linux and Windows. The first Try It Out section focuses on Linux logging, and the second one focuses on Windows logging. In both cases, you stop and restart the MySQL server and modify the option file. Because MySQL is installed as a service on Windows, you also take the extra step of removing the service and then re-creating it so that it points to the correct option file.

Try It Out Enabling Query and Binary Logging on Linux

The following steps describe how to set up query and binary logging on Linux:

1. Open a Terminal window (if you're working in a GUI environment) or use the shell's command prompt. In either case, you should be at the root directory.
2. First, you must add the necessary options to the option file. To do this, use the Vim text editor to edit the .my.cnf configuration file. To edit the file, execute the following command:

```
vi .my.cnf
```

The `vi` command opens the Vim text editor in Linux, which allows you to edit text files.

3. In order to edit the files, you must change the Vim utility to insert mode by typing the following letter:

```
i
```

As soon as you type this command, you are placed in Insert mode.

4. Scroll down to the section that begins with `[mysqld]`, and add a blank line beneath the `[mysqld]` heading.
5. To set up query and binary logging, add the following code beneath the `[mysqld]` heading:

```
log
log-bin
```

6. Press the Escape button to get out of Insert mode.
7. To indicate to the Vim utility that you have completed your edit, you must type the following command:

```
:
```

When you type this command, a colon is inserted at the bottom of the screen and your cursor is moved to the position after the colon.

8. At the colon, type the following command, and press Enter:

```
wq
```

The `w` option tells the Vim utility to save the edits on exiting, and the `q` option tells Vim to exit the program. When you execute this command, you're returned to the command prompt (at the root).

9. In order for the changes to the option file to take effect, you must shut down the MySQL server and then start it up again. To shut down the server, execute the following command:

```
mysqladmin -u root -p shutdown
```

Enter a password when prompted, and then press Enter. Depending on your Linux installation, you might receive a message indicating that the server has been shut down.

10. Now you must restart the MySQL server. Execute the following command:

```
mysqld_safe -u mysql &
```

You should receive a message indicating that the server has started. If, after you execute the `mysqld_safe` command, you're not returned to the command prompt right away, press Enter to display the prompt.

11. At the command prompt, change to the data directory for your installation, and verify that the following three files have been added to the data directory:
    - <*host*>.log
    - <*host*>-bin.000001
    - <*host*>-bin.index

12. The next part of this process is to verify that changes are being logged to the log files. Use the following command to open the mysql utility with the test database active:

```
mysql test
```

You should be connected to MySQL, and the mysql prompt should be displayed.

13. Next, create a table, add a row to the table, drop the table, and exit the mysql client utility. Execute the following statements:

```
CREATE TABLE t1 (c1 INT);
INSERT INTO t1 VALUES (12);
SELECT * FROM t1;
DROP TABLE t1;
exit
```

You're returned to your shell's command prompt (at the data directory).

14. Use the Vim text editor to view the `<host>.log` file. Execute the following command:

```
vi <host>.log
```

In addition to version, port, and socket information, the file should include the following details:

```
Time                 Id Command     Argument
041001     4:33:09    1 Connect     root@localhost on test
041001     4:33:49    1 Query       CREATE TABLE t1 (c1 INT)
041001     4:34:01    1 Query       INSERT INTO t1 VALUES (12)
041001     4:34:10    1 Query       SELECT * FROM t1
041001     4:34:17    1 Query       DROP TABLE t1
041001     4:34:22    1 Quit
```

15. Once you finish viewing the contents of the log file, type the following command:

```
:
```

When you type this command, a colon is inserted at the bottom of the screen, and your cursor is moved to the position after the colon.

16. At the colon, type the following command, and press Enter:

```
q
```

You're returned to the command prompt (at the data directory).

17. Use the `mysqlbinlog` utility to view the `<host>-bin.000001` file. Execute the following command:

```
mysqlbinlog <host>-bin.000001
```

In addition to version and connection information, the file should include the following details:

```
# at 79
#041005 18:40:08 server id 1  log_pos 79  Query  thread_id=1  exec_time=0  error_
use test;
SET TIMESTAMP=1097026808;
```

```
CREATE TABLE t1 (c1 INT);
# at 138
#041005 18:40:08 server id 1  log_pos 138  Query  thread_id=1  exec_time=0  error_
SET TIMESTAMP=1097026808;
INSERT INTO t1 VALUES (12);
# at 199
#041005 18:40:08 server id 1  log_pos 199  Query  thread_id=1  exec_time=0  error_
SET TIMESTAMP=1097026808;
DROP TABLE t1;
```

Note that, depending on the viewing capacity of your command window, longer lines might be cut off or wrapped around at different places.

How It Works

To set up logging on Linux, you had to first edit the option file, which is stored in the root directory. (You created the .my.cnf option file in Chapter 3. For details about the file, see that chapter.) You edited the option file by adding the following commands to the `[mysqld]` section:

```
log
log-bin
```

The log command implements query logging and creates the `<host>.log`. The `log-bin` command implements binary logging and creates the `<host>-bin.000001` file and the `<host>-bin.index` file. If a binary log file already exists, the file extension is incremented by one, based on the highest file number.

After you modified and saved the .my.cnf option file, you used the following command to shut down the MySQL server:

```
mysqladmin -u root -p shutdown
```

The command uses the mysqladmin utility and the shutdown option to shut down MySQL. The command also specifies the root user account and prompts for a password. Once the server was shut down, you immediately started it up again by using the following command:

```
mysqld_safe -u mysql &
```

The command uses the `mysqld_safe` shell script to start the server and monitor it. The user specified is the mysql account that was created when you installed MySQL. (See Chapter 2 for information about that account.) The ampersand (&) at the end of the command is specific to Linux and indicates that the script should run as a process in the background to support the script's monitoring capabilities.

After restarting the MySQL server, you viewed the contents of the data directory to verify that the three new log files had been added. From there, you used the mysql client utility to execute several SQL statements; then you used the Vim text editor to view the contents of the `<host>.log` file. The log file should have contained the four SQL statements that you executed, as well as the time and date of when they were executed.

Next, you used the following command to view the contents of the `<host>-bin.000001` file:

```
mysqlbinlog <host>-bin.000001
```

The mysqlbinlog command allows you to view the contents of the binary log and index files. The log file should have contained the SQL statements that affect data (CREATE TABLE, INSERT, and DROP TABLE), but not the SELECT statement.

Try It Out Enabling Query and Binary Logging on Windows

The following steps describe how to set up query and binary logging on Windows:

1. Because the default MySQL service points to a different option file than the client utilities use, you can delete the current service and re-create it to point to the my.ini file. To remove the MySQL service, you must first stop it. Open a Command Prompt window, and at the command prompt, execute the following command:

```
net stop mysql
```

   You should receive a message verifying that the MySQL service has been stopped.

2. Next, remove the MySQL service by executing the following command:

```
mysqld-nt --remove
```

   You should receive a message verifying that the service has been removed.

3. You must now add the MySQL service back to Windows, only now the service should point to the my.ini file. Execute the following command:

```
"c:\program files\mysql\mysql server 4.1\bin\mysqld-nt" --install MySQL --defaults-
file="c:\windows\my.ini"
```

   You should receive a message that verifies that the service has been successfully installed.

4. Before you can use MySQL, you must start the service. Execute the following command:

```
net start mysql
```

   You should receive a message that indicates that the MySQL service has been started.

5. The next step is to set up logging in the my.ini option file. Open the C:\WINDOWS\my.ini file in a text editor such as Notepad. Scroll down to the section that begins with [mysqld], and add a blank line beneath the [mysqld] heading.

6. Add the following code beneath the [mysqld] heading:

```
log
log-bin
```

7. Save and close the my.ini file.

8. To implement the changes added to the my.ini file, you should first stop the MySQL service and then restart it. To stop the service, execute the following command:

```
net stop mysql
```

   You should receive a message that indicates that the service has been stopped.

You could have waited to start up the service until after you modified the option file, thus skipping steps 4 and 8. By restarting the service before modifying the option file, though, you're ensuring that the service has been properly added to the Windows environment and that it starts up properly. If you wait to start up the service until after you modify the option file and you run into a problem when starting the service, you do not know if the problem is because of the service configuration or because of the change you made to the option file.

9. To start the MySQL service, execute the following command:

```
net start mysql
```

   You should receive a message that verifies that the service has been started.

10. At the command prompt, change to the `C:\Program Files\MySQL\MySQL Server 4.1\data` directory, and then verify that the following three files have been added to the data directory:

    ❑ <*host*>.log

    ❑ <*host*>-bin.000001

    ❑ <*host*>-bin.index

11. You use the mysql client utility to verify that the log files have been properly initiated. Execute the following command:

```
mysql test
```

   You should be connected to MySQL, and the mysql prompt should be displayed.

12. Next, create a table, add a row to the table, drop the table, and exit the mysql client utility. Execute the following SQL statements:

```
CREATE TABLE t1 (c1 INT);
INSERT INTO t1 VALUES (12);
SELECT * FROM t1;
DROP TABLE t1;
exit
```

   Each statement should be executed and should return the appropriate message. You're then returned to the command prompt (at the data directory).

13. Use a text editor such as Notepad to view the contents of the `<host>.log`. In addition to version, port, and socket information, the file should include the following details:

```
Time                 Id Command     Argument
041001     4:33:09    1 Connect     root@localhost on test
041001     4:33:49    1 Query       CREATE TABLE t1 (c1 INT)
                      1 Query       INSERT INTO t1 VALUES (12)
                      1 Query       SELECT * FROM t1
                      1 Query       DROP TABLE t1
                      1 Quit
```

14. Once you finish viewing the contents of the log file, close the file.

15. Use the `mysqlbinlog` utility to view the *`<host>`*`-bin.000001` file. At the command prompt, execute the following command:

```
mysqlbinlog <host>-bin.000001
```

In addition to version and connection information, the file should include the following details:

```
# at 79
#041005 18:40:08 server id 1  log_pos 79  Query  thread_id=1  exec_time=0  error_
use test;
SET TIMESTAMP=1097026808;
CREATE TABLE t1 (c1 INT);
# at 138
#041005 18:40:08 server id 1  log_pos 138  Query  thread_id=1  exec_time=0  error_
SET TIMESTAMP=1097026808;
INSERT INTO t1 VALUES (12);
# at 199
#041005 18:40:08 server id 1  log_pos 199  Query  thread_id=1  exec_time=0  error_
SET TIMESTAMP=1097026808;
DROP TABLE t1;
```

How It Works

To consolidate your commands into one option file, you had to first delete the MySQL service and then re-create the service. The first step, then, was to use the following command to stop the MySQL service:

```
net stop mysql
```

The command stops the service, which means that you can no longer access the MySQL server. You had to stop the service before you could remove it. Once it was stopped, you executed the following command:

```
mysqld-nt --remove
```

The `mysqld-nt` command (the MySQL server normally implemented in a Windows environment) uses the `--remove` option to remove the service from Windows. Once the service was removed, you added a new service by using the following command:

```
"c:\program files\mysql\mysql server 4.1\bin\mysqld-nt" --install MySQL --defaults-
file="c:\windows\my.ini"
```

The command first specifies the path and filename of the `mysqld-nt` server file, the `--install` option, and the `--defaults-file` option. The `--install` option specifies that the new service be named MySQL, and the `--defaults-file` option specifies that the `my.ini` option file, which is located in the `C:\WINDOWS` directory, should be referenced when starting the service.

After you created the MySQL service, you then used the following command to start the service:

```
net start mysql
```

Starting the service now allows you to verify that it has been successfully installed. To test that installation, you can try to open the mysql client utility, which works only if a connection to the server can be established (which means that the service is installed and running).

Once the new service had been installed, your next step was to set up logging. In order to do so, you had to edit first the option file, which is stored in the `C:\WINDOWS` directory. (You created the `my.ini` option file in Chapter 3. For details about the file, see that chapter.) You edited the option file by adding the following commands to the `[mysqld]` section:

```
log
log-bin
```

The log command implements query logging and creates the `<host>.log`. The `log-bin` command implements binary logging and creates the `<host>-bin.000001` file and the `<host>-bin.index` file. If a binary log file already exists, the file extension is incremented by one, based on the highest file number.

After the option file had been modified, you again stopped and then restarted the MySQL service. From there, you viewed the contents of the data directory to verify that the three new log files had been added. Then you used the mysql client utility to execute several SQL statements. After you exited the mysql utility, you used a text editor to view the contents of the `<host>.log` file. The log file should have contained the four SQL statements that you executed, as well as the time and date of when they were executed.

Next, you used the following command to view the contents of the `<host>-bin.000001` file:

```
mysqlbinlog <host>-bin.000001
```

The `mysqlbinlog` command allows you to view the contents of the binary log and index files. The log file should have contained the SQL statements that affect data (`CREATE TABLE`, `INSERT`, and `DROP TABLE`), but not the `SELECT` statement.

Summary

This chapter covered a number of topics related to administering the MySQL server. You learned how to view and modify system settings, take specific actions that affect how MySQL runs, and implement logging. Specifically, you learned how to perform the following administrative tasks:

- ❑ Use the mysqladmin client utility to view system information and modify server-related processes
- ❑ Use the `SHOW VARIABLES` statement to view server system variables
- ❑ Use the `SELECT` statement to view dynamic system variable settings
- ❑ Use the `SHOW STATUS` statement to retrieve server status variable settings
- ❑ Specify system settings at server startup
- ❑ Specify system settings in an option file
- ❑ Specify system settings at runtime
- ❑ View the error log file
- ❑ Implement query logging and view query log files
- ❑ Implement binary logging and view binary log files

Now that you have an overview of how to perform basic administrative tasks in MySQL, you're ready to move on to the next topic — security. By understanding how to set up your system's security, you can protect the data in your databases and permit access only to specified users, preventing all other users from viewing or modifying that data. From there, you move on to optimizing performance, implementing replications, backing up your databases, and restoring those databases should it become necessary. Fortunately, much of what you've learned in this chapter provides you with the foundation you need to perform the administrative tasks described in subsequent chapters.

Exercises

In this chapter, you learned how to perform a number of administrative tasks. To help you build on your ability to carry out these tasks, the following exercises are provided. To view solutions to these exercises, see Appendix A.

1. Create a command that flushes the table cache, closes and then reopens the log files, and reloads the grant tables. In addition, the command must return status information about the MySQL server. You must use the command as the user myadmin, and the command must prompt you for your password.
2. Create an SQL statement that displays all global server system variables and their values for those variables whose name contains the string *max* anywhere in the name.
3. Create an SQL statement that displays the current session setting for the dynamic system variable named `query_cache_limit`.
4. Create an SQL statement that displays all server status variables and their values for those variables whose name contains the string *cache* anywhere in the name.
5. Create an SQL statement that sets the session value of the `max_tmp_tables` dynamic system variable to 24.
6. Add the necessary commands to an option file in order to enable binary logging and create the necessary binary log and index files. The files should be stored in the data directory.
7. Create a command that displays the contents of the `Server21-bin.000327` binary log file.