CHAPTER – 9

SETTING UP A MYSQL DATABASE SERVER

Overview:

MySQL is a popular structured query language (SQL) database server. Like other SQL servers, MySQL provides the means of accessing and managing SQL databases. However, MySQL also provides tools for creating database structures, as well as for adding, modifying, and removing data from those structures. Because MySQL is a relational database, data can be stored and controlled in small, manageable tables. Those tables can be used in combination to create flexible yet complex data structures.

A Swedish company called MySQL AB is responsible for developing MySQL (www.mysql.com). MySQL AB has released MySQL as an open-source product, gaining revenue by offering a variety of MySQL support packages. The company also supports several application programming interfaces (APIs) to help application developers and Web content creators to access MySQL content.

Because MySQL is an open-source product, it has been ported to several different operating systems (primarily UNIX and Linux systems, though there are Windows versions and now even a Mac OS X version as well). As we may have guessed, these include binary versions of MySQL that run on Red Hat Linux.

Finding MySQL Packages:

We need at least the mysql and mysql-server packages installed to set up MySQL using the procedures. The following MySQL packages come with the Red Hat Linux distribution:

- **mysql**: This software package contains a lot of MySQL client programs (in /usr/bin), several client shared libraries, the default MySQL configuration file (/etc/my.cnf), a few sample configuration files, files to support different languages, and documentation.

- **mysql-server**: This software package contains the MySQL server daemon (mysqld) and the mysqld start-up script (/etc/init.d/mysqld). The package also creates various administrative files and directories needed to set up the MySQL databases.

- **mysql-devel**: This software package contains libraries and header files required for developing MySQL applications.

- **php-mysql**: This software package contains a shared library that allows PHP applications to access MySQL databases. This basically allows us to add PHP scripts to our Web pages that can access our MySQL database.

In the current version of Red Hat Linux, all mysql software packages are contained on CD #3. If they are not installed, we can install the mysql packages using the rpm command or the redhat-config-packages window.

Configuring the MySQL Server:

Like most server software in Red Hat Linux, the MySQL server relies on a start-up script and a configuration file to provide the service. Server activities are logged to the mysqld.log file in the /var/logdirectory. There are also mysql user and group accounts for managing MySQL activities. The following sections describe how these components all work together.

Tip: *For many of the steps described in this section, the MySQL server daemon must be running. Starting the server is described in detail later in this chapter. For the moment, we can start the server temporarily by typing the following (as root user):* ***/etc/init.d/mysqld start***

☑ Using Mysql User/Group Accounts:

When the MySQL software is installed, it automatically creates a mysql user account and a mysql group account. These user and group accounts are assigned to MySQL files and activities. In this way, someone can manage the MySQL server without needing to have root permission.

The mysql user entry appears in the /etc/password file as follows:

```
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash
```

The mysql entry just shown indicates that the UID and GID for the mysql user is 27. The text string identifying this user account is "MySQL Server." The home directory is /var/lib/mysql and the default shell is /bin/bash. The home directory identified will contain directories that hold each table of data we define for the MySQL server.

The group entry for mysql is even simpler. The following entry in the /etc/group file indicates that the mysql group has a group ID (GID) of 27.

```
mysql:x:27:
```

If we care to check the ownership of files associated with MySQL, we will see that most of these files have mysql assigned as the user account and group account that own each file. This allows daemon processes that are run by the mysql user to access the database files.

☑ Adding Administrative Users:

To administer MySQL, we need to create at least one administrative account. We can do this using the mysqladmincommand. To add the root user as a MySQL administrator, log in as the root user and type the following from a Terminal window (substituting our own password in place of my47gmc):

```
# mysqladmin -u root password my47gmc
```

After this command is run, the root user can run any MySQL administrative commands using the password.

If we happen to be logged in as another user when we want to use administrative privilege for a MySQL command, we can do that without re-logging in. Simply add the -u root argument to the command line of the MySQL command we are running. In other words, the Linux root user account has no connection to the MySQL root user account once the MySQL account is created. We would typically use different passwords for the two accounts.

Tip: To save ourself the trouble of typing in the password each time we run a MySQL client command, we can add a password option under the [client] group in one of the option files. The most secure way to do that is to create a .my.cnffile in the root user's home directory that contains the following lines (substituting our password for the last argument shown).

```
[client]
password=my47gmc
```

☑ Setting MySQL Options:

We can set options that affect how the MySQL applications behave by using options files or command-line arguments. The MySQL server (as well as other administrative tools) reads the following options files when it starts up (if those files exist):

- **/etc/my.cnf:** Contains global options read by mysqld (server daemon) and mysql.server (script to start the server daemon).
- **/var/lib/mysql/my.cnf:** Contains options primarily for the mysqld daemon.
- **-defaults-extra-file:** We can identify a file on the command line that contains options to be used by the server. For example, the following command would cause the file /home/jim/my.cnf to be read for options after the global options and before the user-specific options:

```
# mysqld --defaults-extra-file=/home/jim/my.cnf
```

- **$HOME/.my.cnf:** Contains user-specific options. (The $HOME refers to the user's home directory, such as /home/susyq.)

Table below shows the MySQL commands that read the options files (in the order shown in the previous bullet list) and use those options in their processing. Options are contained within groups that are identified by single words within brackets. Group names that are read by each command are also shown in the table.

Command	Description	Group names
mysqld (in /usr/libexec)	The MySQL server daemon.	[mysqld] [server]
safe_mysqld	Run by the mysql start-up script to start the MySQL server.	[mysql] [server] [mysql.server]
mysql	Offers a text-based interface for displaying and working with MySQL databases.	[mysql] [client]
mysqladmin	Used to create and maintain MySQL databases.	[mysqladmin] [client]
isamchk	Used to check, fix, and optimize ISAM databases (.ism suffix).	[isamchk]
myisamchk	Used to check, fix, and optimize MyISAM databases (.myi suffix).	[myisamchk]
pack_isam	Used to pack ISAM databases (.ism suffix).	[pack_isam]

myisampack	Used to compress MyISAM database tables.	[myisampack]
mysqldump	Offers a text-based interface for backing up MySQL databases.	[mysqldump] [client]
mysqlimport	Loads plain-text data files into MySQL databases.	[mysqlimport] [client]
mysqlshow	Shows MySQL databases and tables we select.	[mysqlshow] [client]

Though we can use any of the options files to set our MySQL options, begin by configuring the /etc/my.cnf file. Later, if we want to override any of the values set in that file we can do so using the other options files or command-line arguments.

☑ Creating the My.Cnf Configuration File:

Global options that affect how the MySQL server and related client programs run are defined in the /etc/my.cnf file. The default my.cnf file contains only a few settings needed to get a small MySQL configuration going. The following is an example of the /etc/my.cnf file that comes with MySQL:

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock

[mysql.server]
user=mysql
basedir=/var/lib

[safe_mysqld]
err-log=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid
```

Most of the settings in the default my.cnf file define the locations of files and directories needed by the mysqld server. Each option is associated with a particular group, with each group identified by a name in square brackets. The above options are associated with the mysqld daemon ([mysqld]), the MySQL server ([mysql.server]), and the safe_mysqld script that starts the mysqld daemon ([safe_mysqld]). (See Table above for a list of these clients.)

The default datadir value indicates that /var/lib/mysql is the directory that stores the mysql databases we create. The socketoption identifies /var/lib/mysql/mysql.sock as the socket that is used to create the MySQL communications end-point associated with the mysqld server. The basedir option identifies /var/lib as the base directory in which the mysql software is installed. The user option identifies mysql as the user account that has permission to do administration of the MySQL service.

The err-log and pid-file options tell the safe_mysqld script the locations of the error log (/var/log/mysqld.log) and the file that stores the process ID of the mysqld daemon when it is running (/var/run/mysqld/mysqld.pid). The safe_mysqld script actually starts the mysqld daemon from the mysqld start-up script.

☑ Choosing Options:

There are many values that are used by the MySQL server that are not explicitly defined in the my.cnf file. The easiest way to see which options are available for MySQL server and clients is to run each command with the --help option. For example, to see available mysqld options (as well as other information) type the following from a Terminal window:

```
# /usr/libexec/mysqld --help | less
```

Then press the Spacebar to step through the information one screen at a time.

Another way to find which options are available is with the man command. For example, to see which options are available to set for the mysqld daemon, type the following:

```
man mysqld
```

It's quite likely that we can try out our MySQL database server without changing any options at all. However, after we set up our MySQL database server in a production environment, we will almost surely want to tune the server to match the way the server is used. For example, if it is a dedicated MySQL server, we will want to allow MySQL to consume more of the system resources than it would by default.

Below are a few examples of additional options that we may want to set for MySQL:

- **password = *ourpwd***: Adding this option to a [client] group in a user's $HOME/.my.cnf file allows the user to run MySQL client commands without having to enter a password each time. (Replace *ourpwd* with the user's password.)
- **port = #** : Defines the port number to which the MySQL service listens for MySQL requests. (Replace # with the port number we want to use.) By default, MySQL listens to port number 3306 on TCP and UDP protocols.
- **safe-mode**: Tells the server to skip some optimization steps when the server starts.
- **tmpdir = *path***: Identifies a directory, other than the default /tmp, for MySQL to use for writing temporary files. (Substitute a full path name for *path*.)

In addition to the options we can set, MySQL clients also have a lot of variables that we can set. Variables set such things as buffer sizes, timeout values, and acceptable packet lengths. These variables are also listed on the --help output. To change a variable value, we can use the --set-variable option, followed by the variable name and value. For example, to set the sort_buffer variable to 10MB, we could add the following option under our [mysqld] group:

```
[mysqld]
set-variable = sort_buffer=10M
```

The following is a list of other variables we could set for our server. In general, raising the values of these variables improves performance, but also consumes more system resources. So we need to be careful raising these values on machines that are not dedicated to MySQL or that have limited memory resources.

Note: *For variables that require us to enter a size, indicate Megabytes using an M (e.g. 10M) or Kilobytes using a K (e.g. 256K).*

- **key_buffer_size = *size***: Sets the buffer size that is used for holding index blocks that are used by all threads. This is a key value to raise to improve MySQL performance.
- **max_allowed_packet = *size***: Limits the maximum size of a single packet. Raise this limit if we require processing of very large columns.
- **myisam_sort_buffer_size = *size***: Sets the buffer size used for sorting while repairing an index, creating an index, or altering a table.
- **record_buffer = *size***: Sets the buffer size used for threads doing sequential scans. Each process doing a sequential scan allocates a buffer of the size set here.
- **sort_buffer = size**: Defines how much buffer size is allocated for each thread that needs to do a sort. Raising this value makes sorting threads go faster.
- **table_cache = #**: Limits the total number of tables that can be open at the same time for all threads. The number of this variable represents the total number of file descriptors that MySQL can have open at the same time.
- **thread_cache = *size***: Sets the number of threads that are kept in cache, awaiting use by MySQL. When a thread is done being used, it is placed back in the cache. If all the threads are used, new threads must be created to service requests.

☑ **Checking Options:**

In addition to seeing how options and variables are set in the options files, we can also view how all variables are set on our current system. We can view both the defaults and the current values being used by the MySQL server.

The --help command-line argument lets us see the options and variables as they are set for the server and for each MySQL client. Here is an example of the output showing this information for the mysqld server daemon:

```
# /usr/libexec/mysqld --help | less
      .
      .
      .
The default values (after parsing the command line arguments) are:


basedir:     /usr/
datadir:     /var/lib/mysql/
tmpdir:      /tmp/
```

```
language:    /usr/share/mysql/english/
pid file:    /var/lib/mysql/maple.pid
TCP port:    3306
Unix socket: /var/lib/mysql/mysql.sock

system locking is not in use

Possible variables for option --set-variable (-O) are:
back_log              current value: 50
bdb_cache_size        current value: 8388600
bdb_log_buffer_size   current value: 0
bdb_max_lock          current value: 10000
bdb_lock_max          current value: 10000
binlog_cache_size     current value: 32768
connect_timeout       current value: 5
      .
      .
      .
table_cache           current value: 64
thread_concurrency    current value: 10
thread_cache_size     current value: 0
tmp_table_size        current value: 33554432
thread_stack          current value: 65536
wait_timeout          current value: 28800
```

After the server is started, we can see the values that are actually in use by running the mysqladmin command with the variables option. (Pipe the output to the less command so we can page through the information.) Here is an example (if we haven't stored our password, we will be prompted to enter it here):

```
# mysqladmin variables | less
+-------------------------+-----------------------------------------|
| Variable_name           | Value                                   |
+-------------------------+-----------------------------------------
| back_log                | 50                                      |
| basedir                 | /usr/                                   |
| bdb_cache_size          | 8388600                                 |
| bdb_log_buffer_size     | 32768                                   |
```

```
| bdb_home                | /var/lib/mysql/                          |
| bdb_max_lock            | 10000                                    |
| bdb_logdir              |                                          |
| bdb_shared_data         | OFF                                      |
| bdb_tmpdir              | /tmp/                                    |
                      .
                      .
                      .
| tmp_table_size          | 33554432                                 |
| tmpdir                  | /tmp/                                    |
| version                 | 3.23.57                                  |
| wait_timeout            | 28800                                    |
+-------------------------+------------------------------------------+
```

If we decide that the option and variable settings that come with the default MySQL system don't exactly suit us, we don't have to start from scratch. Sample my.cnf files that come with the mysql package can let us begin with a set of options and variables that are closer to the ones we need.

☑ Using Sample My.Cnf Files:

Sample my.cnf files are available in the /usr/share/doc/mysql-server* directory. To use one of these files, do the following:

1. Keep a copy of the old my.cnf file:

```
# mv /etc/my.cnf /etc/my.cnf.old
```

2. Copy the sample my.cnf file we want to the /etc/my.cnf file. For example, to use the my-medium.cnf file, type the following:

```
# cp /usr/share/doc/mysql-server*/my-medium.cnf /etc/my.cnf
```

3. Edit the new /etc/my.cnf file (as root user) using any text editor to further tune our MySQL variables and options.

The following paragraphs describe each of the sample my.cnf files:

- **my-small.cnf:**

This options file is recommended for computer systems that have less than 64MB of memory and are only used occasionally for MySQL. With this options file, MySQL won't be able to handle a lot of usage but it won't be a drag on the performance of our computer.

For the mysqld server, buffer sizes are set low, only 64K for the sort_buffer and 16K for the key_buffer. The thread_stack is only set to 64K and net_buffer_length is only 2K. The table_cache is set to 4.

- **my-medium.cnf:**

As with the small options file, the my-medium.cnf file is intended for systems where MySQL is not the only important application running. This system also has a small amount of total memory available between 32MB and 64MB, however more consistent MySQL use is expected.

The key_buffer size is set to 16M in this file, while the sort_buffer value is raised to 512K for the mysqld server. The table_cache is set to 64 (allowing more simultaneous threads to be active). The net_buffer_length is raised to 8K.

- **my-large.cnf:**

The my-large.cnf sample file is intended for computers that are dedicated primarily to MySQL service. It assumes about 512M of available memory.

Server buffers allow more active threads and better sorting performance. Half of the system's assumed 512M of memory is assigned to the key_buffer variable (256M). The sort_buffer size is raised to 1M. The table_cache allows more simultaneous users (up to 256 active threads).

- **my-huge.cnf:**

As with the my-large.cnf file, the my-huge.cnf file expects the computer to be used primarily for MySQL. However, the system for which it is intended offers much more total memory (between 1G and 2G of memory).

Sort buffer size (sort_buffer) is raised to 2M while the key_buffer is set to consume 384M of memory. The table_cache size is doubled to allow up to 512 active threads.

Starting the MySQL Server:

For Red Hat Linux, the MySQL server is off by default. To turn it on, however, is fairly simple. The /etc/init.d/mysqld start-up script is delivered with the mysql-server package. To start the server, we can either run the mysqld start-up script manually or set it to start each time our system boots.

To start the MySQL server manually, type the following from a Terminal window as root user:

```
# service mysqld start
```

To have the MySQL server start each time the computer reboots, type the following (as root):

```
# chkconfig mysqld on
```

This sets mysqld to start during most multiuser run states (levels 3, 4, and 5). To check that the service is turned on for those levels, type **chkconfig --list mysqld** from a Terminal window.

Checking That MySQL Server Is Working:

We can use the mysqladmin or mysqlshow commands to check that the MySQL server is up and running. Here's an example of how to check information about the MySQL server using the mysqladmin command.

```
# mysqladmin -u root -p version proc
Enter password: ********
mysqladmin  Ver 8.23 Distrib 3.23.57, for pc-linux-gnu on i686
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       3.23.57
Protocol version     10
Connection           Localhost via UNIX socket
UNIX socket          /var/lib/mysql/mysql.sock
Uptime:              2 days 10 hours 47 min 35 sec

Threads: 1  Questions: 184  Slow queries: 0  Opens: 1  Flush tables: 3
Open tables: 1 Queries per second avg: 0.001
+----+------+----------+------+-------+------+-----+------------------+
| Id | User | Host     | db   |Command| Time |State| Info             |
+----+------+----------+------+-------+------+-----+------------------+
| 52 | root | localhost| mysql| Query | 0    |     | show processlist |
+----+------+----------+------+-------+------+-----+------------------+
```

Each of the two options to mysqladmin shown here provides useful information. The version information shows the mysqladmin version is 8.23 and the number assigned to this distribution of the mysql server is 3.23.57. The binary package was created for PC versions of Linux/GNU on the i686 processor. The connection to the server is through a UNIX socket (mysql.sock) on the local host. The server has been up for 2 days, 10 hours, 47 minutes, and 35 seconds. Statistics show that there is one thread (connection to the server) currently active. There have been 184 requests to the server.

The proc option shows that one client is currently connected to the server. That client is logged into MySQL as the root user on the localhost. The client that has an Id of 52 (which we could use, as the server's administrator, if we wanted to disconnect the user) is currently querying the MySQL database.

If the server were not running at the moment, the mysqladmin command shown above would result in a failure message:

```
mysqladmin: connect to server at 'localhost' failed.
```

Recommended remedies are to try to restart the server (by typing /etc/init.d/mysqld restart) or to make sure that the socket exists (/var/lib/mysql/mysql.sock).

Working with MySQL Databases:

The first time we start the MySQL server (using the start-up script described previously), the system creates the initial grant tables for the MySQL database. It does this by running the mysql_install_dbcommand.

Note: If for some reason the grant tables are not initialized, we can run the mysql_install_db command ourself (as root user). Running the command more than once won't hurt anything.

The mysql_install_db command starts us off with two databases: mysql and test. As we create data for these databases, that information is stored in the /var/lib/mysql/mysql and /var/lib/mysql/testdirectories, respectively.

☑ Starting The Mysql Command:

To get started creating databases and tables, we can use the mysql command. From any Terminal window, open the mysql database on our computer by typing the following:

```
# mysql -u root -p mysql
Enter password: *********


Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 39 to server version: 3.23.57
Type 'help;' or '\h' for help. Type '\c' to clear the buffer


mysql>
```

Type in the root user's MySQL password as prompted. The mysql> prompt appears, ready to accept commands for working with the mysql default database on the localhost. If we are connecting to the MySQL server from another host computer, add -h *hostname* to the command line (where *hostname* is the name or IP address of the computer on which the MySQL server is running). Remember, we can also log in as any valid mysql login we created, regardless of which Linux login account we are currently logged in under.

As the mysql text notes above, be sure to end each command that we type with a semicolon (;) or a \g. If we type a command and it appears to be waiting for more input, it's probably because we forgot to put a semicolon at the end.

Before we begin using the mysql interface to create databases, try checking the status of the MySQL server using the status command. The following is an example of output from the status command:

```
mysql> status
--------------mysql  Ver 11.18 Distrib 3.23.57, for pc-linux-gnu (i686)

Connection id:        43
Current database:     mysql
Current user:         root@localhost
Current pager:        stdout
Using outfile:        ''
Server version:       3.23.57
Protocol version:     10
Connection:           Localhost via UNIX socket
Client characterset:  latin1
Server characterset:  latin1
UNIX socket:          /var/lib/mysql/mysql.sock
Uptime:               1 day 2 hours 57 min 19 sec

Threads: 1  Questions: 136  Slow queries: 0  Opens: 12
Flush tables: 1  Open tables: 6 Queries per second avg: 0.001
--------------
```

The status information tells us about the version of the MySQL server (11.18) and the distribution (3.23.57). The output also reminds us of the current database (mysql) and our user name (root@localhost). We can see how long the server has been up (Uptime). We can also see how many threads are currently active and how many commands have been run to query this server (Questions).

☑ CREATING A DATABASE WITH MYSQL

Within an interactive mysql session, we can create and modify databases and tables. If we are not already connected to a mysql session, type the following command (assuming the mysql user name of root):

```
# mysql -u root -p
Enter password: *******
mysql>
```

The general steps for creating a MySQL database include creating the database name, identifying the new database as the current database, creating tables, and adding data to the tables. While

we are connected to a mysql session, we can run the procedure below to create a sample database.

1. To create a new database name, use the CREATE DATABASE command at the mysql> prompt. For example, to create a database named allusers, type the following:

```
mysql> CREATE DATABASE allusers;
```

This action creates a database called allusers in the /var/lib/mysql directory.

Note: *We could alternatively have created a database from the command line using the mysqladmin command. For example, to create the database named allusers with mysqladmin, we could have typed the following:****mysqladmin -u root -p create allusers***

2. To see what databases are available for our mysql server, type the following from the mysql> command prompt:

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| allusers |
| mysql    |
| test     |
+----------+
  3 rows in set (0.00 sec)
```

The databases shown here are named allusers, mysql, and test. The allusers database is the one created in the previous step. The mysql database contains user access data. The test database is created automatically for creating test mysql databases.

3. To work with the database we just created (allusers), we need to make allusers our current database. To do that, type the following from the mysql> command prompt:

```
mysql> USE allusers;
Database changed
```

4. Creating a table for our database requires some planning and understanding of table syntax. We can type in the following commands and column information to try out creating a table. For more detailed information on creating tables and using different data types, refer to the section "Understanding MySQL Tables" later in this chapter.

To create a table called "names," use the following CREATE TABLE command from a mysql> prompt:

```
mysql> CREATE TABLE names (
-> firstname      varchar(20)    not null,
```

```
-> lastname       varchar(20)    not null,
-> streetaddr     varchar(30)    not null,
-> city           varchar(20)    not null,
-> state          varchar(20)    not null,
-> zipcode        varchar(10)    not null
-> );
```

Query OK, 0 rows affected (0.00 sec)

We have now created a table called names for a database named allusers. It contains columns called firstname, lastname, streedaddr, city, state, and zipcode. Each column allows record lengths of between 10 and 30 characters. Although MySQL supports several different database formats, because none is specified here, the default MyISAM database type is used.

With a database and one table created, we can now add data to the table.

☑ Adding Data To A Mysql Database Table

After the database is created and the structure of a database table is in place, we can begin working with the database. We can add data to our MySQL database by manually entering each record during a mysql session or by adding the data into a plain-text file and loading that file into the database.

Manually Entering Data:

To do the procedure in this section, let's assume we have an open interactive mysql session and that we have created a database and table as described in the previous section. If we are not already connected to a mysql session, type the following command (assuming the mysql user name of root):

```
# mysql -u root -p
Enter password: *******
mysql>
```

To add data to an existing MySQL database, the following procedure describes how to view the available tables and load data into those tables manually. The next section describes how to create a plain-text file containing database data and how to load that file into our database.

1. To make the database we want to use our current database (in this case, allusers), type the following command from the mysql> prompt:

```
mysql> USE allusers;
Database changed
```

2. To see the tables that are associated with the current database, type the following command from the mysql> prompt:

```
mysql> SHOW tables;
```

```
+--------------------+
| Tables_in_allusers |
+--------------------+
| names              |
+--------------------+
1 row in set (0.00 sec)
```

We can see that the only table defined so far for the all users database is the one called names.

3. To display the format of the names table, type the following command from the mysql> prompt:

```
mysql> DESCRIBE names;
+-----------+------------+------+-----+---------+-------+
| Field     | Type       | Null | Key | Default | Extra |
+-----------+------------+------+-----+---------+-------+
| firstname | varchar(20)|      |     |         |       |
| lastname  | varchar(20)|      |     |         |       |
| streetaddr| varchar(30)|      |     |         |       |
| city      | varchar(20)|      |     |         |       |
| state     | varchar(20)|      |     |         |       |
| zipcode   | varchar(10)|      |     |         |       |
+-----------+------------+------+-----+---------+-------+
```

4. To add data to the new table, we can use the INSERT INTO command from the mysql> prompt. Here is an example of how to add a person's name and address to the new table:

```
mysql> INSERT INTO names
-> VALUES ('Jerry','Wingnut','167 E Street',
-> 'Roy','UT','84103');
```

In this example, the INSERT INTO command identifies the names table. Then it indicates that values for a record in that table include the name Jerry Wingnut at the address 167 E Street, Roy, UT 84103.

5. To check that the data has been properly entered into the new table, type the following command from the mysql>prompt:

```
mysql> SELECT * FROM names;
  +-----------+----------+-------------+-------+-------+---------+
  | firstname | lastname | streetaddr  | city  | state | zipcode |
```

```
+-----------+----------+-------------+-------+-------+---------+
| Jerry     | Wingnut  | 167 E Street| Roy   | UT    | 84103   |
+-----------+----------+-------------+-------+-------+---------+
```

The resulting output shows the data we just entered, displayed in the columns we defined for the names table. If we like, we can continue adding data in this way.

Typing each data item individually can be tedious. As an alternative, we can add our data to a plain-text file and load it into our MySQL database, as described in the following section.

Loading Data from A File:

Using the LOAD DATA command during a mysql session, we can load a file containing database records into our MySQL database. Here are a few things we need to know about creating a data file to be loaded into MySQL.

- We can create the file using any Linux text editor.
- Each record, consisting of all the columns in the table, must be on its own line. (A line feed indicates the start of the next record.)
- Separate each column by a Tab character.
- We can leave a column blank for a particular record by placing a \N in that column.
- Any blank lines we leave in the file result in blank lines in the database table.

In this example, the text shown below is added into a plain-text file. The text is in a format that can be loaded into the "names" table created earlier in this chapter. To try it out, type the following text into a file. Make sure that we insert a Tab character between each value (indicated here as multiple spaces).

```
Chris    Smith      175 Harrison Street   Gig Harbor   WA   98999
John     Jones      18 Talbot Road NW     Coventry     NJ   54889
Howard   Manty      1515 Broadway         New York     NY   10028
```

When we are done entering the data, save the text to any accessible filename (for example, /tmp/name.txt). Remember the filename so that we can use it later. If we are not already connected to a mysql session, type the following command (assuming mysql user name root):

```
# mysql -u root -p
Enter password: *******
mysql>
```

Next, identify the database (allusers in this example) as the current database by typing the following:

```
mysql> USE allusers;
```

```
Database changed
```

To actually load the file into the names table in the allusers database, type the following command to load the file (in this case, /tmp/name.txt) from the mysql> prompt.

Note: *Either enter the full path to the file or have it in the directory where the mysql command starts. In the latter case, we can type the filename without indicating its full path.*

```
mysql> LOAD DATA INFILE "/tmp/name.txt" INTO TABLE names;
Query OK, 3 rows affected (0.02 sec)
Records: 3  Deleted: 0 Skipped: 0 Warnings: 0
```

Type the following at the mysql> prompt to make sure that the records have been added correctly:

```
mysql> SELECT * FROM names;
+----------+---------+--------------------+------------+-------+-------+
| firstname| lastname| streetaddr         | city       | state |zipcode|
+----------+---------+--------------------+------------+-------+-------+
| Chris    | Smith   | 175 Harrison Street| Gig Harbor | WA    | 98999 |
| John     | Jones   | 18 Talbot Road NW  | Coventry   | NJ    | 54889 |
| Howard   | Manty   | 1515 Broadway      | New York   | NY    | 10028 |
+----------+---------+--------------------+------------+-------+-------+
```

Understanding MySQL Tables

We have a lot of flexibility when it comes to setting up MySQL tables. To have our MySQL database operate as efficiently as possible, we want to have the columns be assigned to the most appropriate size and data type to hold the data we need to store.

Use the following tables as a reference to the different data types that can be assigned to our columns. Data types available for use in MySQL fall into these categories: numbers, time/date, and character strings. Here are a few things we need to know as we read these tables:

- The maximum display size for a column is 255 characters. An M data type option sets the number of characters that are displayed and, in most cases, stored for the column.
- There can be up to 30 digits following the decimal point for floating-point data types. A D option to a data type indicates the number of digits allowed for a floating-point number following the decimal point. (The value should be no more than two digits less than the value of the display size being used.)
- The UNSIGNED option (shown in braces) indicates that only positive number to be allowed in the column. This allows the column to hold larger positive numbers.

- The ZEROFILL option (shown in braces) indicates that the data in the column will be padded with zeros. For example, the number 25 in a column with a data type of INTEGER(7) ZEROFILL would appear as 0000025. (Any ZEROFILL column automatically becomes UNSIGNED.)
- All values shown in braces are optional.
- The parentheses shown around the (M) and (D) values are necessary if we enter either of those values. In other words, don't type the braces, but do type the parentheses.

Table below shows numeric data types that we can use with MYSQL.

Data Type	Description	Space Needed
BIGINT[(M)] [UNSIGNED] [ZEROFILL]	Can contain large integers with the following allowable values: -9223372036854775808 to 9223372036854775807 (unsigned) 0 to 18446744073709551615 (signed)	Uses 8 bytes
DECIMAL[(M[,D])] [ZEROFILL]	Contains an unpacked floating-point number (signed only). Each digit is stored as a single character. When we choose the display value (M), decimal points and minus signs are not counted in that value. The value of (M) is 10 by default. Setting D to zero (which is the default) causes only whole numbers to be used.	Uses M+2 bytes if D is greater than 0 Uses M+1 bytes if D is equal to 0
DOUBLE[(M,D)] [ZEROFILL]	Contains a double-precision, floating-point number of an average size. Values that are allowed include: -1.7976931348623157E+308 to -2.2250738585072014E-308 0 2.2250738585072014E-308 to 1.7976931348623157E+308.	Uses 8 bytes
DOUBLE PRECISION	Same as DOUBLE.	Same as DOUBLE.
FLOAT(X) [ZEROFILL]	Contains a floating-point number. For a single-precision floating-point number X can be less than or equal to 24. For a double-precision floating-point number, X can be between 25 and 53. The display size and number of decimals are undefined.	Uses 4 bytes

FLOAT[(M,D)] [ZEROFILL]	Contains a single-precision floating-point number. Values that are allowed include: -3.402823466E+38 to -1.175494351E-38 0 1.175494351E-38 to 3.402823466E+38. If the display value (M) is less than or equal to 24, the number is a single-precision floating-point number.	Uses 4 bytes if X is less than or equal to 24 Uses 8 bytes if X is greater than or equal to 25 and less than or equal to 53
INT[(M)] [UNSIGNED] [ZEROFILL]	Contains an integer of normal size. The range is -2147483648 to 2147483647 if it's signed and 0 to 4294967295 if unsigned.	Uses 4 bytes
INTEGER[(M)] [UNSIGNED] [ZEROFILL]	Same as INT.	Same as INT
MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]	Contains an integer of medium size. The range is -8388608 to 8388607 if it's signed and 0 to 16777215 if unsigned.	Uses 3 bytes
NUMERIC(M,D) [ZEROFILL]	Same as DECIMAL.	Same as DECIMAL
REAL	Same as DOUBLE.	Same as DOUBLE.
SMALLINT[(M)] [UNSIGNED] [ZEROFILL]	Contains an integer of small size. The range is -32768 to 32767 if it's signed and 0 to 65535 if it's unsigned.	Uses 2 bytes
TINYINT[(M)] [UNSIGNED] [ZEROFILL]	A very small integer, with a signed range of -128 to 127 and a 0 to 255 unsigned range.	Uses 1 byte

The default format of dates in MySQL is YYYY-MM-DD, which stands for the year, month, and day. Any improperly formatted date or time values will be converted to zeros. Table 24-3 shows time and date data types that we can use with MYSQL.

Table: Time/Date Data Types for Columns		
Data Type	**Description**	**Space Needed**
DATE	Contains a date between the range of January 1, 1000 (1000-01-01) and December 31, 9999 (9999-12-31).	Uses 3 bytes
DATETIME	Contains a combination of date and time between zero hour of January 1, 1000 (1000-01-01 00:00:00) and the last second of December 31, 9999 (9999-12-31 23:59:59).	Uses 8 bytes

TIMESTAMP[(M)]	Contains a timestamp from between zero hour of January 1, 1970 (1970-01-01 00:00:00) and a time in the year 2037. It is stored in the form: YYYYMMDDHHMMSS. Using (M), we can reduce the size of the TIMESTAMP displayed to less than the full 14 characters (though the full 4-byte TIMESTAMP is still stored).	Uses 4 bytes
TIME	Contains a time between -838:59:59 and 838:59:59. The format of the field is in hours, minutes, and seconds (HH:MM:SS).	Uses 3 bytes
YEAR[(2|4)]	Contains a year, represented by either two or four digits. For a four-digit year, YEAR mean 1901–2155 (0000 is also allowed). For a two-digit year, the digits 70-69 can .represent 1970-2069	Uses 1 byte

Table 24-4: String Data Types for Columns

Data Type	Description	Space Needed
BLOB	Contains a binary large object (BLOB) that varies in size, based on the actual value of the data, rather than on the maximum allowable size. Searches on a BLOB column are case-sensitive.	Uses up to L+2 bytes, where L is less than or equal to 65535
[NATIONAL] CHAR(M) [BINARY]	Contains a character string of fixed length, with spaces padded to the right to meet the length. To display the value, the spaces are deleted. The value of (M) determines the number of characters (from 1 to 255). If the BINARY keyword is used, sorting of values is case-sensitive (it is case-insensitive by default). The NATIONAL keyword indicates that the default CHARACTER set should be used.	Uses between 1 and 255 bytes, based on the value of (M)
ENUM('val1','val2',...)	Contains enumerated strings that are typically chosen from a list of values indicated when we create the column. For example, we set a column definition to ENUM("dog","cat","mouse"). Then, if we set the value of that column to "1" the value displayed would be "dog", "2" would be "cat" and "3" would be mouse. It lets us take a number as input and have a string as output. Up to 65535 values are allowed.	Uses either 1 byte (for up to about 255 values) or 2 bytes, (for up to 65535 values)
LONGBLOB	Contains a binary large object (BLOB) that varies in size, based on the actual value of the data, rather than on the maximum allowable size. LONGBLOB allows larger	Uses up to L+4 bytes, where L is less than or equal to 4294967295

	values than MEDIUMBLOB. Searches on a LONGBLOB column are case-sensitive.	
LONGTEXT	Same as LONGBLOB, except that searching is done on these columns in case-insensitive style.	Uses up to L+4 bytes, where L is less than or equal to 4294967295
MEDIUMBLOB	Contains a binary large object (BLOB) that varies in size, based on the actual value of the data, rather than on the maximum allowable size. MEDIUMBLOB allows larger values than BLOB. Searches on a MEDIUMBLOB column are case-sensitive.	Uses up to L+3 bytes, where L is less than or equal to 16777215
MEDIUMTEXT	Same as MEDIUMBLOB, except that searching is done on these columns in case-insensitive style.	Uses up to L+3 bytes, where L is less than or equal to 16777215
SET('val1','val2',...)	Contains a set of values. A SET column can display zero or more values from the list of values contained in the SET column definition. Up to 64 members are allowed.	Uses 1, 2, 3, 4 or 8 bytes, varying based on how many of the up to 64 set members are used.
TEXT	Same as BLOB, except that searching is done on these columns in case-insensitive style.	Uses up to L+2 bytes, where L is less than or equal to 65535
TINYBLOB	Contains a binary large object (BLOB) that varies in size, based on the actual value of the data, rather than on the maximum allowable size. TINYBLOB allows smaller values than BLOB. Searches on a TINYBLOB column are case-sensitive.	Uses up to L+1 bytes, where L is less than or equal to 255
TINYTEXT	Same as TINYBLOB, except that searching is done on these columns in case-insensitive style.	Uses up to L+1 bytes, where L is less than or equal to 255
[NATIONAL] VARCHAR(M) [BINARY]	Contains a character string of variable length, with no padded spaces added. The value of (M) determines the number of characters (from 1 to 255). If the BINARY keyword is used, sorting of values is case-sensitive (it is case-insensitive by default). The NATIONAL keyword indicates that the default CHARACTER set should be used.	Uses L+1 bytes, where L is less than or equal to M and M is from 1 to 255 characters

Displaying MySQL Databases:

There are many different ways of sorting and displaying database records during a mysql session. If we are not already connected to a mysql session, type the following command (assuming the mysql user name of root):

```
# mysql -u root -p
Enter password: *******
mysql>
```

When we are in our mysql session (and have chosen a database), we can display all or selected table records, choose which columns are displayed, or choose how records are sorted.

☑ Displaying All or Selected Records:

Assuming that the current database is allusers (as shown in the previous examples), type the following command to choose (SELECT) all records (*) from the names table and display them in the order in which they were entered into the database.

```
mysql> SELECT * FROM names;
+-----------+--------+------------------+------------+-------+--------+
| firstname |lastname|streetaddr        | city       | state | zipcode|
+-----------+--------+------------------+------------+-------+--------+
| Chris     |Smith   |175 Harrison Street| Gig Harbor | WA    | 98999  |
| John      |Jones   |18 Talbot Road NW  | Coventry   | NJ    | 54889  |
| Howard    |Manty   |1515 Broadway      | New York   | NY    | 10028  |
+-----------+--------+------------------+------------+-------+--------+
```

The following command displays all records from the names table that have the lastname column set to Jones. Instead of using lastname, we could search for a value from any column name used in the table.

```
mysql> SELECT * FROM names WHERE lastname = "Jones";
+-----------+---------+-----------------+------------+-------+--------+
| firstname |lastname |streetaddr       | city       | state | zipcode|
+-----------+---------+-----------------+------------+-------+--------+
| John      |Jones    |18 Talbot Road NW | Coventry   | NJ    | 54889  |
+-----------+---------+-----------------+------------+-------+--------+
```

Using the OR operator, we can select records that match several different values. In the following command, records that have either Chris or Howard as the firstname are matched and displayed.

```
mysql> SELECT * FROM names WHERE firstname = "Chris" OR firstname = "Howard";
+-----------+----------+------------------+------------+-------+--------+
| firstname | lastname | streetaddr       | city       | state | zipcode|
```

```
+-----------+----------+--------------------+------------+-------+--------+
| Chris     | Smith    | 175 Harrison Street| Gig Harbor | WA    | 98999  |
| Howard    | Manty    | 1515 Broadway      | New York   | NY    | 10028  |
+-----------+----------+--------------------+------------+-------+--------+
```

To match and display a record based on the value of two columns in a record, we can use the AND operator. In the following command, any record that has Chris as the firstname and Smith as the lastname is matched.

```
mysql> SELECT * FROM names WHERE firstname = "Chris" AND lastname = "Smith";
+----------+---------+--------------------+------------+-------+---------+
| firstname| lastname| streetaddr         | city       | state | zipcode |
+----------+---------+--------------------+------------+-------+---------+
| Chris    | Smith   | 175 Harrison Street | Gig Harbor | WA    | 98999   |
+----------+---------+--------------------+------------+-------+---------+
```

☑ Displaying Selected Columns:

We don't need to display every column of data. Instead of using the asterisk (*) shown in previous examples to match all columns, we can enter a comma-separated list of column names. The following command displays the firstname, lastname, and zipcode records for all of the records in the "names" table:

```
mysql> SELECT firstname,lastname,zipcode FROM names;
+------------+----------+---------+
| firstname  | lastname | zipcode |
+------------+----------+---------+
| Chris      | Smith    | 98999   |
| John       | Jones    | 54889   |
| Howard     | Manty    | 10028   |
+------------+----------+---------+
```

Likewise, we can sort columns in any order we choose. Type the following command to show the same three columns with the zipcode column displayed first:

```
mysql> SELECT zipcode,firstname,lastname FROM names;
+---------+-----------+----------+
| zipcode | firstname  | lastname |
+---------+-----------+----------+
| 98999   | Chris      | Smith    |
| 54889   | John       | Jones    |
```

```
| 10028   | Howard     | Manty    |
+---------+------------+----------+
```

We can also mix column selection with record selection as shown in the following example:

```
mysql> SELECT firstname,lastname,city FROM names WHERE firstname = "Chris";
+------------+----------+------------+
| firstname  | lastname | city       |
+------------+----------+------------+
| Chris      | Smith    | Gig Harbor |
+------------+----------+------------+
```

☑ Sorting Data:

We can sort records based on the values in any column we choose. For example, using the ORDER BY operator, we can display the records based on the lastname column:

```
mysql> SELECT * FROM names ORDER BY lastname;
+----------+---------+-------------------+------------+-------+--------+
| firstname|lastname |streetaddr         | city       | state | zipcode|
+----------+---------+-------------------+------------+-------+--------+
| John     |Jones    |18 Talbot Road NW  | Coventry   | NJ    | 54889  |
| Howard   |Manty    |1515 Broadway      | New York   | NY    | 10028  |
| Chris    |Smith    |167 Small Road     | Gig Harbor | WA    | 98999  |
+----------+---------+-------------------+------------+-------+--------+
```

To sort records based on city name, we could use the following command:

```
mysql> SELECT * FROM names ORDER BY city;
+-----------+---------+------------------+------------+-------+--------+
| firstname |lastname |streetaddr        | city       | state | zipcode|
+-----------+---------+------------------+------------+-------+--------+
| John      |Jones    |18 Talbot Road NW | Coventry   | NJ    | 54889  |
| Chris     |Smith    |167 Small Road    | Gig Harbor | WA    | 98999  |
| Howard    |Manty    |1515 Broadway     | New York   | NY    | 10028  |
+-----------+---------+------------------+------------+-------+--------+
```

Now that we have entered and displayed the database records, we may find that we need to change some of them. The following section describes how to update database records during a mysql session.

Making Changes to Tables and Records:

As we begin to use our MySQL database, we will find that we need to make changes to both the structure and content of the database tables. The following section describes how we can alter the structure of our MySQL tables and change the content of MySQL records. If we are not already connected to a mysql session, type the following command (assuming the mysql user name of root):

```
# mysql -u root -p
Enter password: *******
mysql>
```

To use the examples shown in the following sections, identify the database (allusers in this example) as the current database by typing the following:

```
mysql> USE allusers;
Database changed
```

☑ Altering MySQL tables:

After we have created our database tables, there will inevitably be changes we want to make to them. This section describes how to use the ALTER command during a mysql session for the following tasks: adding a column, deleting a column, renaming a column, and changing the data type for a column.

To add a column to the end of our table that displays the current date, type the following:

```
mysql> ALTER TABLE names ADD curdate TIMESTAMP;
```

The previous line tells mysql to change the table in the current database called names (ALTER TABLE names), add a column named curdate (ADD curdate), and assign the value of that column to display the current date (TIMESTAMP data type). If we decide later that we want to remove that column, we can remove it by typing the following:

```
mysql> ALTER TABLE names DROP COLUMN curdate;
```

If we want to change the name of an existing column, we can do so using the CHANGE option to ALTER. Here is an example:

```
mysql> ALTER TABLE names CHANGE city town varchar(20);
```

In the previous example, the names table is chosen (ALTER TABLE names) to change the name of the city column to town (CHANGE city town). The data type of the column must be entered as well (varchar(20)), even if we are not changing it. In fact, if we just want to change the data type of a column, we would use the same syntax as the previous example but simply repeat the column name twice. Here's an example:

```
mysql> ALTER TABLE names CHANGE zipcode zipcode INTEGER;
```

The previous example changes the data type of the zipcode column from its previous type (varchar) to the INTEGER type.

☑ Updating and deleting MySQL records:

We can select records based on any value we choose and update any values in those records. When we are in our mysqlsession, we can use UPDATE to change the values in a selected table. Here is an example:

```
mysql> UPDATE names SET streetaddr = "933 3rd Avenue" WHERE firstname = "Chris";
Query OK, 1 row affected (0.00 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

The previous example attempts to update the names table (UPDATE names). In this example, each record that has the firstname column set to "Chris" will have the value of the streetaddr column for that record changed to "933 3rd Avenue" instead. Note that the Query found 1 row that matched. That one row matched was also changed, with no error warnings necessary. We can use any combination of values to match records (using WHERE) and change column values (using SET) that we would like. After we have made a change, it is a good idea to display the results to make sure that the change was made as we expected.

To remove an entire row (that is, one record), we can use the DELETE command. For example, if we wanted to delete any row where the value of the firstname column is "Chris", we could type the following:

```
mysql> DELETE FROM names WHERE firstname = "Chris";
Query OK, 1 row affected (0.00 sec)
```

The next time we show the table, there should be no records with the first name Chris.

Adding and Removing User Access:

There are several different methods we can use to control user access to our MySQL databases. To begin with, assign a user name and password to every user that accesses our MySQL databases. Then we can use the GRANT and REVOKE commands of mysql to specifically indicate the databases and tables users and host computers can access, as well as the rights they have to those databases and tables.

Caution: *Database servers are common targets of attacks from crackers. While this chapter gives some direction for granting access to our MySQL server, we need to provide much more stringent protection for the server if we are allowing Internet access. Refer to the General Security Issues section of the MySQL manual (/usr/share/doc/mysql*/manual.html) for further information on securing our MySQL server.*

☑ Adding Users And Granting Access:

Although we have a user account defined to create databases (the root user, in this example), to make a database useful, we may want to allow access to other users as well. The following procedure describes how to grant privileges for our MySQL database to other users.

Note: *If we are upgrading our MySQL from a version previous to 3.22, run the mysql_fix_privilege_tables script. This script adds new GRANT features to our databases. If we don't run the script, we will be denied access to the databases.*

In this example, we are adding a user named bobby that can log in to the MySQL server from the localhost. The password for bobby is i8yer2shuz. (Remember that there does not have to be a Red Hat Linux user account named bobby. So any user on the localhost with the password for bobby can log in to that MySQL account.)

1. If we are not already connected to a mysql session, type the following command (assuming the mysql user name of root):

```
# mysql -u root -p
Enter password: *******
mysql>
```

2. To create the user named bobby and a password i8yer2shuz, use the GRANT command as follows:

```
mysql> GRANT USAGE ON *.*
    -> TO bobby@localhost IDENTIFIED BY "i8yer2shuz";
```

At this point, someone could log in from the localhost using the name bobby and i8yer2shuz password (mysql -u bobby -p). But the user would have no privilege to work with any of the databases. Next we need to grant privileges.

3. To grant bobby privileges to work with the database called all users, we could type the following:

```
mysql> GRANT DELETE,INSERT,SELECT,UPDATE ON allusers.*
    -> TO bobby@localhost;
```

In this example, the user named bobby is allowed to log in to the MySQL server on the localhost and access all tables from the allusers database (USE allusers). For that database, bobby can use the DELETE, INSERT, SELECT, and UPDATE commands.

4. To see the privileges that we just granted, we can select mysql as our current database, then select the db table as follows:

```
mysql> USE mysql;
Database changed
mysql> SELECT * FROM db WHERE db="allusers";
+---------+--------+-----+-----------+-----------+-----------+-----------
|Host     |Db      |User |Select_priv|Insert_priv|Update_priv|Delete_priv
+---------+--------+-----+-----------+-----------+-----------+-----------
|localhost|allusers|bobby| Y         | Y         | Y         | Y
```

```
+---------+--------+-----+-----------+-----------+-----------+-----------
```

The output here shows all users who are specifically granted privileges to the allusers database. Only part of the output is shown here because it is very long. We should either make a very wide Terminal window to view the output or learn how to read wrapped text. Other privileges on the line will be set to N (no access).

☑ Revoking Access:

We can revoke privileges we grant using the REVOKE command. To revoke all privileges for a user to a particular database, use the following procedure:

1. If we are not already connected to a mysql session, type the following command (assuming the mysql user name of root):

```
# mysql -u root -p
Enter password: *******
mysql>
```

2. To revoke all privileges of a user named bobby to use a database named all users on our MySQL server, we could type the following:

```
mysql> REVOKE ALL PRIVILEGES ON allusers.*
    -> FROM bobby@localhost;
```

At this point, bobby has no privileges to use any of the tables in the allusers databases.

3. To see the privileges that we just granted, we can select mysql as our current database, they select the db table as follows:

```
mysql> USE mysql;
Database changed
mysql> SELECT * FROM db WHERE db="allusers";
```

The output should show that the user named bobby is no longer listed as having access to the all users database. (The results might just say Empty set.)

Checking and Fixing Databases:

Over time, databases can become corrupted or store information inefficiently. MySQL comes with commands that we can use to check and repair our databases. The myisamchk and isamchk commands are available to check MyISAM and ISAM database tables, respectively.

MyISAM tables are used by default with MySQL. The tables are stored in the directory /var/lib/mysql/*dbname* by default, where *dbname* is replaced by the name of the

database we are using. For each table, there are three files in this directory. Each file begins with the table name and ends with one of the following three suffixes:

.frm	Contains the definition (or form) or the table
.MYI	Contains the table's index.
.MYD	Contains the table's data.

The following procedure describes how to use the myisamchk command to check our MyISAM tables. (The procedure is the same for checking ISAM tables, except that we use the isamchk command instead.)

Caution: Do a backup of our database tables before running a repair with myisamchk. Though myisamchk is unlikely to damage our data, backups are still a good precaution.

1. Stop MySQL temporarily by typing the following from a Terminal window as root user:

```
# /etc/init.d/mysqld stop
```

2. We can check all or some of our database tables at once. The first example shows how to check a table called names in the allusers database.

```
# myisamchk /var/lib/mysql/allusers/names.MYI
Checking MyISAM file: /var/lib/mysql/allusers/names.MYI
Data records:       5   Deleted blocks:       0
- check file-size
- check key delete-chain
- check record delete-chain
- check index reference
- check record links
```

We could also check tables for all our databases at once as follows:

```
# myisamchk /var/lib/mysql/*/*.MYI
```

The preceding example shows a simple, five-record database where no errors were encountered. If instead of the output shown above, we see output like the following, we may need to repair the database:

```
Checking MyISAM file: names.MYI
Data records:       5   Deleted blocks:       0
- check file-size
myisamchk: warning: Size of datafile is: 89 Should be: 204
- check key delete-chain
- check record delete-chain
- check index reference
```

```
- check record links
myisamchk: error: Found wrong record at 0
MyISAM-table 'names.MYI' is corrupted
Fix it using switch "-r" or "-o"
```

3. To fix a corrupted database, we could run the following command

```
# myisamchk -r /var/lib/mysql/allusers/names.MYI
- recovering (with keycache) MyISAM-table 'names.MYI'
Data records: 5
Found wrong stored record at 0
Data records: 4
```

4. If for some reason the -r options doesn't work, we can try running the myisamchk command with the -o option. This is a slower, older method of repair, but it can handle a few problems that the -r option cannot. Here is an example:

```
# myisamchk -o /var/lib/mysql/allusers/names.MYI
```

If our computer has a lot of memory, raise the key buffer size value on the myisamchk command line, which will lessen the time it takes to check the databases. For example, we could use the following command line:

```
myisamchk -r -O --key_buffer_size=64M *.MYI
```

This would set the key buffer size to 64MB.