CHAPTER 28

Securing MySQL

It's almost a natural reaction; when exiting your automobile, you take a moment to lock the doors and set the car alarm, if you have one. You do so because you know that the possibility of the car or its contents being stolen dramatically increases if you do not take such rudimentary yet effective precautions. Ironically, the IT industry at large seems to take the opposite approach when creating the vehicles used to maintain enterprise data. Both IT systems and applications are rife with open doors, leading to intellectual property theft, damage, and even destruction as a result of electronic attacks. Often, such occurrences take place not because the technology does not offer deterrent features, but simply because the developers never bothered to put these deterrents into effect.

This chapter introduces several key aspects of MySQL's configuration and highly effective security model. In particular, this chapter describes MySQL's user privilege system in great detail, showing you how to create users, manage privileges, and change passwords. Additionally, MySQL's secure (SSL) connection feature is introduced. You'll also learn how to place limitations on user resource consumption. After completing this chapter, you should be familiar with the following topics:

- Steps to take immediately after starting the mysqld daemon for the first time
- How to secure the mysqld daemon
- MySQL's access privilege system
- The `GRANT` and `REVOKE` functions
- User account management
- Creating secure MySQL connections with SSL

Let's start at the beginning: What you should do *before doing anything else* with your MySQL database server.

What You Should Do First

This section outlines several rudimentary yet very important tasks that you should undertake immediately after completing the installation and configuration process outlined in Chapter 25:

- **Patch the operating system and any installed software**: Software security alerts seem to be issued on a weekly basis these days, and although they are annoying, it's absolutely necessary that you take the steps to ensure that your system is fully patched. With exploit instructions and tools readily available on the Internet, a malicious user with even little experience in such matters will have little trouble taking advantage of an unpatched server. Even if you're using a managed server, don't blindly depend on the service provider to perform the necessary upgrades; instead, monitor support updates to ensure that matters are being taken care of.

- **Disable all unused system services**: Always take care to eliminate all unnecessary potential server attack routes before you place the server on the network. These attack vectors are almost exclusively the result of insecure system services, often ones running on the system unbeknownst to the system administrator. In short, if you're not going to use a service, disable it.

- **Close the firewall**: Although shutting off unused system services is a great way to lessen the probability of a successful attack, it doesn't hurt to add a second layer of security by closing all unused ports. For a dedicated database server, consider closing all ports below 1024 except for the designated SSH port, 3306 (MySQL), and a handful of "utility" ports, such as 123 (NTP). In short, if you don't intend for traffic to travel on a given port, close it off altogether. In addition to making such adjustments on a dedicated firewall appliance or router, also consider taking advantage of the operating system's firewall. Both Microsoft Windows Server 2000/2003 and Unix-based systems have built-in firewalls at your disposal.

- **Audit the server's user accounts**: Particularly if a pre-existing server has been repurposed for hosting the organization's database, make sure that all nonprivileged users are disabled or, better yet, deleted. Although MySQL users and operating system users are completely unrelated, the mere fact that they have access to the server environment raises the possibility that damage could be done, inadvertently or otherwise, to the database server and its contents. To completely ensure that nothing is overlooked during such an audit, consider reformatting all attached drives and reinstalling the operating system.

- **Set the MySQL root user password**: By default, the MySQL root (administrator) account password is left blank. Although many find this practice questionable, this has long been the standard procedure, and it will likely be this way for some time. You must take care to add a password immediately! You can do so with the `SET PASSWORD` command, like so:

```
%>mysql -u root mysql
%>SET PASSWORD FOR root@localhost=PASSWORD('secret');
%>FLUSH PRIVILEGES;
```

- Of course, choose a password that is a tad more complicated than `secret`. MySQL will let you dig your own grave in the sense that passwords such as 123, abc, and your dog's name are perfectly acceptable. Consider choosing a password that is at least eight characters in length, and consists of a mixture of numeric and alphabetical characters of varying case.

Securing the mysqld Daemon

There are several security options that you can use when you start the mysqld daemon:

- `--skip-networking`: Prevents the use of TCP/IP sockets when connecting to MySQL, meaning that remote connections aren't accepted regardless of the credentials provided. If your application and database reside on the same server, you should definitely consider including this option.
- `--skip-name-resolve`: Prevents the use of hostnames when connecting to the MySQL database, instead allowing only IP addresses or localhost.
- `--safe-show-database`: Causes the `SHOW DATABASES` command to return only those databases for which the user possesses some sort of privilege. If you're running version 4.02 or higher, this option is enabled by default.
- `--skip-show-database`: Prevents any user that does not possess the `SHOW DATABASES` privilege from using the command entirely. As of version 4.02, the `Show_db_priv` column located in the `user` table mimics this feature. (See the next section for more information about the `user` table.)
- `--local-infile`: Disabling this option by setting it to 0 disables use of the command `LOAD DATA LOCAL INFILE`, which when enabled allows the client to load a file from their local machine. See Chapter 37 for more information about this command.
- `--safe-user-create`: Prevents any user from creating new users via the `GRANT` command if they do not also possess the `INSERT` privilege for the `user` table.

The MySQL Access Privilege System

Protecting your data from unwarranted review, modification, or deletion, accidental or otherwise, should always be your primary concern. Yet balancing a secure database with an expected level of user convenience and flexibility is often a difficult affair. The delicacy of this balance becomes obvious when you consider the wide array of access scenarios that might exist in any given environment. For example, what if a user requires modification privileges, but not insertion privileges? How do you authenticate a user who might need to access the database from a number of different IP addresses? What if you want to provide a user with read access to only certain table columns, while restricting the rest? You can imagine the nightmarish code that might result from incorporating such features into the application logic. Thankfully, the MySQL developers have relieved you of these tasks, integrating fully featured authentication and authorization capabilities into the server. This is commonly referred to as the MySQL access privilege system.

How the Privilege System Works

MySQL's privilege system revolves around two general concepts:

- **Authentication**: Determines whether a user is even allowed to connect to the server.
- **Authorization**: Determines whether the user possesses adequate privileges to execute query requests.

Because authorization cannot take place without successful authentication, you can think of this process as taking place in two stages.

The Two Stages of Access Control

The general privilege control process takes place in two distinct stages: *connection authentication* and *request verification*. Together, these stages are carried out in five distinct steps:

1. MySQL uses the contents of the `user` table to determine whether the incoming connection should be accepted or rejected. This is done by matching the specified host and the user to a row contained within the `user` table. MySQL also determines whether the user requires a secure connection to connect, and whether the number of maximum allowable connections per hour for that account has been exceeded. The execution of Step 1 completes the authentication stage of the privilege control process.
2. Step 2 initiates the authorization stage of the privilege control process. If the connection is accepted, MySQL verifies whether the maximum allowable number of queries or updates per hour for that account has been exceeded. Next, the corresponding privileges as granted within the `user` table are examined. If any of these privileges are enabled (set to `y`), then the user has global privileges, for any database, to act in the capacity granted by that privilege. Of course, in most cases, all of these privileges are disabled, which causes Step 3 to occur.
3. The `db` table is examined, verifying which databases this user is allowed to interact with. Any corresponding privileges enabled in this table correspond to all tables within those databases that the user is allowed to interact with.
4. If a row in the `db` table is found to have a matching user but an empty host value, the `host` table is then examined. If a matching host value is found, the user has those privileges for that database as indicated in the `host` table, and not in the `db` table. This is done to allow for host-specific access on a given database.
5. Finally, if a user attempts to execute a command that has not been granted in the `user`, `db`, or `host` tables, the `tables_priv` and `columns_priv` tables are examined, to determine whether the user is able to execute that command on the table(s) or column(s) in question.

As you may have gathered from the process breakdown, the system examines privileges by starting with the very broad and ending with the very specific. Let's consider a concrete example.

■ **Note** Only as of MySQL 4.0.2 was it possible to impose maximum hourly connections, updates, and queries for a user. As of MySQL 5.0.3, it's possible to set the maximum number of simultaneous connections for a user.

Tracing Through a Real-World Connection Request

Suppose user `jason` would like to insert a new row into the `widgets` table. This example takes the following variables into account:

- Database: `company`
- Table: `widgets`

- User: jason
- Connecting from: www.example.com
- Password: secret

MySQL first determines whether jason is authorized to connect to the database, and, if so, then determines whether he's allowed to execute the INSERT request:

1. Does user jason@www.example.com require a secure connection? If yes, and user jason@www.example.com has attempted to connect without the required security certificate, deny the request and end the authentication procedure. If no, proceed to Step 2.
2. If MySQL version 4.0.2 or higher is running, determine whether the jason account has exceeded the maximum allowable number of hourly connections, denying the authentication procedure. If not, and MySQL version 5.0.3 or higher is running, MySQL determines whether the maximum number of simultaneous connections has been exceeded. If both tasks pass muster, proceed to Step 3.
3. Does user jason@www.example.com possess the necessary privileges to connect to the database server? If yes, proceed to Step 4. If no, deny access and end the control procedure. This step ends the authentication component of the privilege control mechanism.
4. Has user jason@www.example.com exceeded the maximum number of allowable updates or queries? If not, proceed to Step 5.
5. Does user jason@www.example.com possess global INSERT privileges? If yes, accept and execute the insertion request. If no, proceed to Step 6.
6. Does user jason@www.example.com possess INSERT privileges for the company database? If yes, accept and execute the insertion request. If no, proceed to Step 7.
7. Does user jason@www.example.com possess INSERT privileges for the widget table columns specified in the insertion request? If yes, accept and execute the insertion request. If no, deny the request and end the control procedure.

By now you should be beginning to understand the generalities surrounding MySQL's access-control mechanism. However, the picture isn't complete until you're familiar with the technical underpinnings of this process. This matter is introduced next.

Where Is Access Information Stored?

MySQL's privilege verification information is stored in the mysql database, which is installed by default along with the database server. Specifically, six tables found in this database play an important role in the authentication and privilege verification process:

- user: Determines which users can log in to the database server from which host
- db: Determines which users can access which databases
- host: An extension of the db table, offering additional hostnames from which a user can connect to the database server

- `tables_priv`: Determines which users can access specific tables of a particular database
- `columns_priv`: Determines which users can access specific columns of a particular table
- `procs_priv`: Governs the use of stored procedures

This section delves into the details pertinent to the purpose and structure of each privilege table.

The user Table

The `user` table is unique in the sense that it is the only privilege table to play a role in both stages sof the privilege request procedure. During the authentication stage, the `user` table is solely responsible for granting user access to the MySQL server, determining whether the user has exceeded the maximum allowable connections per hour (MySQL 4.0.2 and greater), and determining whether the user has exceeded the maximum simultaneous connections (MySQL 5.0.3 and greater). During this stage, the `user` table also determines whether SSL-based authorization is required; if it is, the `user` table checks the necessary credentials. See the later section "Secure MySQL Connections" for more information about this feature.

In the request authorization stage, the `user` table determines whether those users granted access to the server have been assigned *global* privileges for working with the MySQL server. That is, any privilege enabled in this table allows a user to work in some capacity with all databases located on that MySQL server. During this stage, the `user` table also determines whether the user has exceeded the maximum number of allowable queries and updates per hour. See the later section "Limiting User Resources" for more information about controlling resource usage on a per-user basis.

The `user` table possesses another defining characteristic: It is the only privilege table to store privileges pertinent to the administration of the MySQL server. For example, this table is responsible for determining which users are allowed to execute commands relevant to the general functioning of the server, such as shutting down the server, reloading user privileges, and viewing and even killing existing client processes. Thus, this table plays quite an important role in the access privilege procedure.

Because of its wide-ranging responsibilities, `user` is the largest of the privilege tables, containing a total of 37 fields: three scope, and the rest privilege. Table 28-1 offers information regarding the columns found in the `user` table, including their names, datatypes, attributes, and default values. Following the table, a more thorough introduction of each column's purpose is offered.

Table 28-1. *Overview of the user Table*

Column	Datatype	Null	Default
`Host`	`char(60) binary`	No	No default
`User`	`char(16) binary`	No	No default
`Password`	`char(41) binary`	No	No default
`Select_priv`	`enum('N','Y')`	No	`N`
`Insert_priv`	`enum('N','Y')`	No	`N`

Table 28-1. *Overview of the user Table*

Column	Datatype	Null	Default
Update_priv	enum('N','Y')	No	N
Delete_priv	enum('N','Y')	No	N
Create_priv	enum('N','Y')	No	N
Drop_priv	enum('N','Y')	No	N
Reload_priv	enum('N','Y')	No	N
Shutdown_priv	enum('N','Y')	No	N
Process_priv	enum('N','Y')	No	N
File_priv	enum('N','Y')	No	N
Grant_priv	enum('N','Y')	No	N
References_priv	enum('N','Y')	No	N
Index_priv	enum('N','Y')	No	N
Alter_priv	enum('N','Y')	No	N
Show_db_priv	enum('N','Y')	No	N
Super_priv	enum('N','Y')	No	N
Create_tmp_table_priv	enum('N','Y')	No	N
Lock_tables_priv	enum('N','Y')	No	N
Execute_priv	enum('N','Y')	No	N
Repl_slave_priv	enum('N','Y')	No	N
Repl_client_priv	enum('N','Y')	No	N
Create_view_priv	enum('N','Y')	No	N
Show_view_priv	enum('N','Y')	No	N
Create_routine_priv	enum('N','Y')	No	N
Alter_routine_priv	enum('N','Y')	No	N
Create_user_priv	enum('N','Y')	No	N
ssl_type	enum('','ANY','X509','SPECIFIED')	No	0
ssl_cipher	blob	No	0
x509_issuer	blob	No	0
x509_subject	blob	No	0
max_questions	int(11) unsigned	No	0
max_updates	int(11) unsigned	No	0
max_connections	int(11) unsigned	No	0
max_user_connections	int(11) unsigned	No	0

Host

The `Host` column specifies the hostname that determines the host address from which a user can connect. Addresses can be stored as either hostnames, IP addresses, or wildcards. Wildcards can consist of either the `%` or _ character. In addition, netmasks may be used to represent IP subnets. Several example entries follow:

- `www.example.com`
- `192.168.1.2`
- `%`
- `%.example.com`
- `192.168.1.0/255.255.255.0`
- `localhost`

User

The `User` column specifies the case-sensitive username capable of connecting to the database server. Although wildcards are not permitted, blank values are. If the entry is empty, any user arriving from the corresponding `Host` entry will be allowed to log in to the database server. Example entries follow:

- `jason`
- `Jason_Gilmore`
- `secretary5`

Password

The `Password` column stores the encrypted password supplied by the connecting user. Although wildcards are not allowed, blank passwords are. Therefore, make sure that all users are provided with a corresponding password to alleviate potential security issues.

Passwords are stored in a one-way hashed format, meaning that they cannot be converted back to their plain-text format. Furthermore, as of version 4.1, the number of bytes required to store a password increased from 16 bytes to 41 bytes. Therefore, if you're importing data from a pre-4.1 version, and you want to take advantage of the added security offered by the longer hashes, you need to increase the size of the `Password` column to fit the new space requirement. You can do so either by manually altering the table with the `ALTER` command or by running the utility `mysql_fix_privilege_tables`. If you choose not to alter the table, or cannot, then MySQL will still allow you to maintain passwords, but will simply continue to use the old method for doing so.

USER IDENTIFICATION

MySQL identifies a user not just by the supplied username, but by the combination of the supplied username and the originating hostname. For example, `jason@localhost` is entirely different from `jason@www.wjgilmore.com`. Furthermore, keep in mind that MySQL will always apply the most specific set of permissions that matches the supplied `user@host` combination. Although this may seem obvious, sometimes unforeseen consequences can happen. For example, it's often the case that multiple rows match the requesting user/host identity; even if a wildcard entry that satisfies the supplied `user@host` combination is seen before a later entry that perfectly matches the identity, the privileges corresponding to that perfect match will be used instead of the wildcard match. Therefore, always take care to ensure that the expected privileges are indeed supplied for each user. Later in this chapter, you'll see how to view privileges on a per-user basis.

The User Privilege Columns

The next 26 columns listed in Table 28-1 comprise the user privilege columns:

- `Select_priv`: Determines whether the user can select data via the `SELECT` command.
- `Insert_priv`: Determines whether the user can insert data via the `INSERT` command.
- `Update_priv`: Determines whether the user can modify existing data via the `UPDATE` command.
- `Delete_priv`: Determines whether the user can delete existing data via the `DELETE` command.
- `Create_priv`: Determines whether the user can create new databases and tables.
- `Drop_priv`: Determines whether the user can delete existing databases and tables.
- `Reload_priv`: Determines whether the user can execute various commands specific to flushing and reloading of various internal caches used by MySQL, including logs, privileges, hosts, queries, and tables.
- `Shutdown_priv`: Determines whether the user can shut down the MySQL server. You should be very wary of providing this privilege to anybody except the root account.
- `Process_priv`: Determines whether the user can view the processes of other users via the `SHOW PROCESSLIST` command.
- `File_priv`: Determines whether the user can execute the `SELECT INTO OUTFILE` and `LOAD DATA INFILE` commands.
- `Grant_priv`: Determines whether the user can grant privileges already granted to that user to other users. For example, if the user can insert, select, and delete information located in the `foo` database, and has been granted the `GRANT` privilege, that user can grant any or all of these privileges to any other user located in the system.

- `References_priv`: Currently just a placeholder for some future function; it serves no purpose at this time.
- `Index_priv`: Determines whether the user can create and delete table indexes.
- `Alter_priv`: Determines whether the user can rename and alter table structures.
- `Show_db_priv`: Determines whether the user can view the names of all databases residing on the server, including those for which the user possesses adequate access privileges. Consider disabling this for all users unless there is a particularly compelling reason otherwise.
- `Super_priv`: Determines whether the user can execute certain powerful administrative functions, such as the deletion of user processes via the `KILL` command, the changing of global MySQL variables using `SET GLOBAL`, and the execution of various commands pertinent to replication and logging.
- `Create_tmp_table_priv`: Determines whether the user can create temporary tables.
- `Lock_tables_priv`: Determines whether the user can block table access/modification using the `LOCK TABLES` command.
- `Execute_priv`: Determines whether the user can execute stored procedures. This privilege is only relevant for MySQL 5.0 and greater.
- `Repl_slave_priv`: Determines whether the user can read the binary logging files used to maintain a replicated database environment. This user resides on the master system, and facilitates the communication between the master and the client machines.
- `Repl_client_priv`: Determines whether the user can determine the location of any replication slaves and masters.
- `Create_view_priv`: Determines whether the user can create a view. This privilege is only relevant for MySQL 5.0 and greater. See Chapter 33 for more information about views.
- `Show_view_priv`: Determines whether the user can see a view or learn more about how it executes. This privilege is only relevant for MySQL 5.0 and greater. See Chapter 33 for more information about views.
- `Create_routine_priv`: Determines whether the user can create stored procedures and functions. This privilege is only relevant for MySQL 5.0 and greater.
- `Alter_routine_priv`: Determines whether the user can alter or drop stored procedures and functions. This privilege is only relevant for MySQL 5.0 and greater.
- `Create_user_priv`: Determines whether the user can execute the `CREATE USER` statement, which is used to create new MySQL accounts.

The Remaining Columns

The remaining eight columns listed in Table 28-1 are so interesting that entire sections are devoted to them later in this chapter. You can learn more about the `max_questions`, `max_updates`, `max_connections`, and `max_user_connections` columns in the section "Limiting User Resources."

You can learn more about the ssl_type, ssl_cipher, x509_issuer, and x509_subject columns in the section "Secure MySQL Connections."

The db Table

The db table is used to assign privileges to a user on a per-database basis. It is examined if the requesting user does not possess global privileges for the task she's attempting to execute. If a matching User/Host/Db triplet is located n the db table, and the requested task has been granted for that row, then the request is executed. If the User/Host/Db/task match is not satisfied, one of two events occurs:

- If a User/Db match is located, but the host is blank, then MySQL looks to the host table for help. The purpose and structure of the host table is introduced in the next section.
- If a User/Host/Db triplet is located, but the privilege is disabled, MySQL next looks to the tables_priv table for help. The purpose and structure of the tables_priv table is introduced in a later section.

Wildcards, represented by the % and _ characters, may be used in both the Host and Db columns, but not in the User column. Like the user table, the rows are sorted so that the most specific match takes precedence over less-specific matches. An overview of the db table's structure is presented in Table 28-2.

Table 28-2. *Overview of the db Table*

Column	Datatype	Null	Default
Host	char(60)	No	No default
Db	char(64)	No	No default
User	char(16)	No	No default
Select_priv	enum('N','Y')	No	N
Insert_priv	enum('N','Y')	No	N
Update_priv	enum('N','Y')	No	N
Delete_priv	enum('N','Y')	No	N
Create_priv	enum('N','Y')	No	N
Drop_priv	enum('N','Y')	No	N
Grant_priv	enum('N','Y')	No	N
References_priv	enum('N','Y')	No	N
Index_priv	enum('N','Y')	No	N
Alter_priv	enum('N','Y')	No	N
Create_tmp_table_priv	enum('N','Y')	No	N
Lock_tables_priv	enum('N','Y')	No	N
Create_view_priv	enum('N','Y')	No	N

Table 28-2. *Overview of the db Table (Continued)*

Column	Datatype	Null	Default
Show_view_priv	enum('N','Y')	No	N
Create_routine_priv	enum('N','Y')	No	N
Alter_routine_priv	enum('N','Y')	No	N
Execute_priv	enum('N','Y')	No	N

The host Table

The host table comes into play only if the db table's Host field is left blank. You might leave the db table's Host field blank if a particular user needs access from various hosts. Rather than reproducing and maintaining several User/Host/Db instances for that user, only one is added (with a blank Host field), and the corresponding hosts' addresses are stored in the host table's Host field.

Wildcards, represented by the % and _ characters, may be used in both the Host and Db columns, but not in the User column. Like the user table, the rows are sorted so that the most specific match takes precedence over less-specific matches. An overview of the host table's structure is presented in Table 28-3.

Table 28-3. *Overview of the host Table*

Column	Datatype	Null	Default
Host	char(60)	No	No default
Db	char(64)	No	No default
Select_priv	enum('N','Y')	No	N
Insert_priv	enum('N','Y')	No	N
Update_priv	enum('N','Y')	No	N
Delete_priv	enum('N','Y')	No	N
Create_priv	enum('N','Y')	No	N
Drop_priv	enum('N','Y')	No	N
Grant_priv	enum('N','Y')	No	N
References_priv	enum('N','Y')	No	N
Index_priv	enum('N','Y')	No	N
Alter_priv	enum('N','Y')	No	N
Create_tmp_table_priv	enum('N','Y')	No	N
Lock_tables_priv	enum('N','Y')	No	N
Create_view_priv	enum('N','Y')	No	N
Show_view_priv	enum('N','Y')	No	N

Table 28-3. *Overview of the host Table*

Column	Datatype	Null	Default
Create_routine_priv	enum('N','Y')	No	N
Alter_routine_priv	enum('N','Y')	No	N
Execute_priv	enum('N','Y')	No	N

The tables_priv Table

The tables_priv table is intended to store table-specific user privileges. It comes into play only if the user, db, and host tables do not satisfy the user's task request. To best illustrate its use, consider an example. Suppose that user jason from host example.com wants to execute an UPDATE on the table staff located in the database company. Once the request is initiated, MySQL will begin by reviewing the user table to see if jason@example.com possesses global INSERT privileges. If this is not the case, the db and host tables are next reviewed for database-specific insertion privileges. If these tables do not satisfy the request, MySQL then looks to the tables_priv table to verify whether user jason@example.com possesses the insertion privilege for the table staff found in the company database.

An overview of the tables_priv table is found in Table 28-4.

Table 28-4. *Overview of the tables_priv Table*

Column	Datatype	Null	Default
Host	char(60)	No	No default
Db	char(64)	No	No default
User	char(16)	No	No default
Table_name	char(64)	No	No default
Grantor	char(77)	No	No default
Timestamp	timestamp	Yes	Current timestamp
Table_priv	*tableset*	No	No default
Column_priv	*columnset*	No	No default

Because of space limitations, the term *tableset* is used as a placeholder for set(Select, Insert, Update, Delete, Create, Drop, Grant, References, Index, Alter, Create view, Show view). The term *columnset* is a placeholder for set(Select, Insert, Update, References).

All the columns found in the tables_priv table should be familiar, except for the following:

- Table_name: Determines the table to which the table-specific permissions set within the tables_priv table will be applied.
- Grantor: Specifies the username of the user granting the privileges to the user.

- `Timestamp`: Specifies the exact date and time when the privilege was granted to the user.
- `Table_priv`: Determines which table-wide permissions are available to the user. The following privileges can be applied in this capacity: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CREATE`, `DROP`, `GRANT`, `REFERENCES`, `INDEX`, and `ALTER`.
- `Column_priv`: Stores the names of any column-level privileges assigned to that user for the table referenced by the `Table_name` column. The purpose for doing so is undocumented, although one would suspect that it is done in an effort to improve general performance.

The columns_priv Table

The `columns_priv` table is responsible for setting column-specific privileges. It comes into play only if the `user`, `db/host`, and `tables_priv` tables are unable to determine whether the requesting user has adequate permissions to execute the requested task.

An overview of the `columns_priv` table is found in Table 28-5.

Table 28-5. *Overview of the columns_priv Table*

Column	Datatype	Null	Default
`Host`	`char(60) binary`	No	No default
`Db`	`char(64) binary`	No	No default
`User`	`char(16) binary`	No	No default
`Table_name`	`char(60) binary`	No	No default
`Column_name`	`char(64) binary`	No	No default
`Timestamp`	`timestamp`	Yes	`Null`
`Column_priv`	*`columnset`*	No	No default

All columns found in this table should be familiar, except for `Column_name`, which specifies the name of the table column affected by the `GRANT` command.

The procs_priv Table

The `procs_priv` table governs the use of stored procedures and functions. An overview of the `procs_priv` table is found in Table 28-6.

Table 28-6. *Overview of the procs_priv Table*

Column	Datatype	Null	Default
`Host`	`char(60) binary`	No	No default
`Db`	`char(64) binary`	No	No default
`User`	`char(16) binary`	No	No default
`Routine_name`	`char(64) binary`	No	No default

Table 28-6. *Overview of the procs_priv Table*

Column	Datatype	Null	Default
Routine_type	enum	No	No default
Grantor	char(77) binary	No	No default
Proc_priv	*columnset*	No	No default
Timestamp	timestamp	Yes	Null

The term *columnset* is a placeholder for set(Execute, Alter Routine, Grant). The Routine_type column can take the following values: FUNCTION and PROCEDURE.

User and Privilege Management

The tables located in the mysql database are no different from any other relational tables in the sense that their structure and data can be modified using typical SQL commands. In fact, up until version 3.22.11, this was exactly how the user information found in this database was managed. However, with the release of version 3.22.11 came a new, arguably much more intuitive method for managing this crucial data: using the GRANT and REVOKE commands. With these commands, users can be both created and disabled, and their access privileges can be both granted and revoked on the fly. Their exacting syntax eliminates potentially horrendous mistakes that could otherwise be introduced due to a malformed SQL query (for example, forgetting to include the WHERE clause in an UPDATE query).

As of version 5.0, yet another feature was added to further improve the ease with which new users can be added, deleted, and renamed. As you'll soon learn, it's possible to create and effectively delete users by using the GRANT and REVOKE commands. However, the fact that you can use these commands for such purposes may seem a tad nonintuitive given the command names, which imply the idea of granting privileges to and revoking privileges from existing users. Therefore, in version 5.0, two new commands were added to MySQL's administration arsenal: CREATE USER and DROP USER. A third command, RENAME USER, was added for renaming existing users.

CREATE USER

```
CREATE USER user [IDENTIFIED BY [PASSWORD] 'password']
 [, user [IDENTIFIED BY [PASSWORD] 'password']] ...
```

The CREATE USER command is used to create new user accounts. No privileges are assigned at the time of creation, meaning you next need to use the GRANT command to assign privileges. An example follows:

```
mysql>CREATE USER jason@localhost IDENTIFIED BY 'secret';
Query OK, 0 rows affected (0.47 sec)
```

As you can see from the command prototype, it's also possible to simultaneously create more than one user.

DROP USER

```
DROP USER user [, user]...
```

If an account is no longer needed, you should strongly consider removing it to ensure that it can't be used for potentially illicit activity. This is easily accomplished with the `DROP USER` command, which removes all traces of the user from the privilege tables. An example follows:

```
mysql>DROP user jason@localhost;
Query OK, 0 rows affected (0.03 sec)
```

As you can see from the command prototype, it's also possible to simultaneously delete more than one user.

■ **Caution** The `DROP USER` command was actually added in MySQL 4.1.1, but it could only remove accounts with no privileges. This behavior changed in MySQL 5.0.2, and now it can remove an account regardless of privileges. Therefore, if you're running MySQL version 4.1.1 through 5.0.1 and use this command, note the command response, because the user may indeed continue to exist even though you thought it had been removed.

RENAME USER

```
RENAME USER old_user TO new_user
 [old_user TO new_user]...
```

On occasion you may want to rename an existing user. This is easily accomplished with the `RENAME USER` command. An example follows:

```
mysql>RENAME USER jason@localhost TO jasongilmore@localhost;
Query OK, 0 rows affected (0.02 sec)
```

As the command prototype indicates, it's also possible to simultaneously rename more than one user.

The GRANT and REVOKE Commands

The `GRANT` and `REVOKE` commands are used to manage access privileges. As previously stated, you can also use them to create and delete users, although, as of MySQL 5.0.2, you can more easily accomplish this with the `CREATE USER` and `DROP USER` commands. The `GRANT` and `REVOKE` commands offer a great deal of granular control over who can work with practically every conceivable aspect of the server and its contents, from who can shut down the server, to who can modify information residing within a particular table column. Table 28-7 offers a list of all possible privileges that can be granted or revoked using these commands.

■ **Tip** Although modifying the `mysql` tables using standard SQL syntax is deprecated, you are not prevented from doing so. Just keep in mind that any changes made to these tables must be followed up with the `flush privileges` command. Because this is an outmoded method for managing user privileges, no further details are offered regarding this matter. See the MySQL documentation for further information.

Table 28-7. *Privileges Managed by GRANT and REVOKE*

Privilege	Description
`ALL PRIVILEGES`	Affects all privileges except `WITH GRANT OPTION`
`ALTER`	Affects the use of the `ALTER TABLE` command
`CREATE`	Affects the use of the `CREATE TABLE` command
`CREATE TEMPORARY TABLES`	Affects the use of the `CREATE TEMPORARY TABLE` command
`CREATE VIEW`	Affects the use of the `CREATE VIEW` command
`DELETE`	Affects the use of the `DELETE` command
`DROP`	Affects the use of the `DROP TABLE` command
`EXECUTE`	Affects the user's ability to run stored procedures
`FILE`	Affects the use of `SELECT INTO OUTFILE` and `LOAD DATA INFILE`
`GRANT OPTION`	Affects the user's ability to delegate privileges
`INDEX`	Affects the use of the `CREATE INDEX` and `DROP INDEX` commands
`INSERT`	Affects the use of the `INSERT` command
`LOCK TABLES`	Affects the use of the `LOCK TABLES` command
`PROCESS`	Affects the use of the `SHOW PROCESSLIST` command
`REFERENCES`	Placeholder for a future MySQL feature
`RELOAD`	Affects the use of the `FLUSH` command set
`REPLICATION CLIENT`	Affects the user's ability to query for the location of slaves and masters
`REPLICATION SLAVE`	Required privilege for replication slaves
`SELECT`	Affects the use of the `SELECT` command
`SHOW DATABASES`	Affects the use of the `SHOW DATABASES` command
`SHOW VIEW`	Affects the use of the `SHOW CREATE VIEW` command
`SHUTDOWN`	Affects the use of the `SHUTDOWN` command
`SUPER`	Affects the use of administrator-level commands such as `CHANGE MASTER`, `KILL thread`, `mysqladmin debug`, `PURGE MASTER LOGS`, and `SET GLOBAL`
`UPDATE`	Affects the use of the `UPDATE` command
`USAGE`	Connection only, no privileges granted

In this section, the GRANT and REVOKE commands are introduced in some detail, followed by numerous examples demonstrating their usage.

GRANT

You use the GRANT command when you need to assign new privileges to a user or group of users. This privilege assignment could be as trivial as granting a user only the ability to connect to the database server, or as drastic as providing a few colleagues root MySQL access (not recommended, of course, but possible). The command syntax follows:

```
GRANT privilege_type [(column_list)] [, privilege_type [(column_list)] ...]
    ON {table_name | * | *.* | database_name.*}
    TO user_name [IDENTIFIED BY 'password']
        [, user_name [IDENTIFIED BY 'password'] ...]
    [REQUIRE {SSL|X509} [ISSUER issuer] [SUBJECT subject]]
    [WITH GRANT OPTION]
```

At first glance, the GRANT syntax may look intimidating, but it really is quite simple to use. Some examples are presented in the following sections to help you become better acquainted with this command.

■ **Note** As soon as a GRANT command is executed, any privileges granted in that command take effect immediately.

Creating a New User

The first example creates a new user and assigns that user a few database-specific privileges. User michele would like to connect to the database server from IP address 192.168.1.103 with the password secret. The following provides her ACCESS, SELECT, and INSERT privileges for all tables found in the books database:

```
mysql>GRANT select, insert ON books.* TO michele@192.168.1.103
    ->IDENTIFIED BY 'secret';
```

Upon execution, two privilege tables will be modified, namely the user and db tables. Because the user table is responsible for both access verification and global privileges, a new row must be inserted, identifying this user. However, all privileges found in this row will be disabled. Why? Because the GRANT command is specific to just the books database. The db table will contain the user information relevant to map user michele to the books table, in addition to enabling the Select_priv and Insert_priv columns.

Adding Privileges to an Existing User

Now suppose that user michele needs the UPDATE privilege for all tables residing in the books database. This is again accomplished with GRANT:

```
mysql>GRANT update ON books.* TO michele@192.168.1.103;
```

Once executed, the row identifying the user michele@192.168.1.103 in the db table is modified so that the Update_priv column is enabled. Note that there is no need to restate the password when adding privileges to an existing user.

Granting Table-Level Privileges

Now suppose that in addition to the previously defined privileges, user michele@192.168.1.103 requires DELETE privileges for two tables located within the books database, namely the authors and editors tables. Rather than provide this user with carte blanche to delete data from any table in this database, you can limit privileges so that she only has the power to delete from those two specific tables. Because two tables are involved, two GRANT commands are required:

```
mysql>GRANT delete ON books.authors TO michele@192.168.1.103;
Query OK, 0 rows affected (0.07 sec)
mysql>GRANT delete ON books.editors TO michele@192.168.1.103;
Query OK, 0 rows affected (0.01 sec)
```

Because this is a table-specific privilege setting, only the tables_priv table will be touched. Once executed, two new rows will be added to the tables_priv table. This assumes that there are not already pre-existing rows mapping the authors and editors tables to michele@192.168.1.103. If this is the case, those pre-existing rows will be modified accordingly to reflect the new table-specific privileges.

Granting Multiple Table-Level Privileges

A variation on the previous example is to provide a user with multiple permissions that are restricted to a given table. Suppose that a new user, rita, connecting from multiple addresses located within the wjgilmore.com domain, is tasked with updating author information, and thus needs only SELECT, INSERT, and UPDATE privileges for the authors table:

```
mysql>GRANT select,insert,delete ON books.authors TO rita@'%.wjgilmore.com'
     ->IDENTIFIED BY 'secret';
```

Executing this GRANT statement results in two new entries to the mysql database: a new row entry within the user table (again, just to provide rita@%.wjgilmore.com with access permissions), and a new entry within the tables_priv table, specifying the new access privileges to be applied to the authors table. Keep in mind that because the privileges apply only to a single table, there will be just one row added to the tables_priv table, with the Table_priv column set to Select,Insert,Delete.

Granting Column-Level Privileges

Finally, consider an example that affects just the column-level privileges of a table. Suppose that you want to grant UPDATE privileges on books.authors.name for user nino@192.168.1.105:

```
mysql>GRANT update (name) ON books.authors TO nino@192.168.1.105;
```

REVOKE

The REVOKE command is responsible for deleting previously granted privileges from a user or group of users. The syntax follows:

```
REVOKE privilege_type [(column_list)] [, privilege_type [(column_list)] ...]
    ON {table_name | * | *.* | database_name.*}
    FROM user_name [, user_name ...]
```

As with GRANT, the best way to understand use of this command is through some examples. The following examples demonstrate how to revoke permissions from, and even delete, existing users.

■ **Note** If the GRANT and REVOKE syntax is not to your liking, and you'd prefer a somewhat more wizard-like means for managing permissions, check out the Perl script mysql_setpermission. Keep in mind that although it offers a very easy-to-use interface, it does not offer all the features that GRANT and REVOKE have to offer. This script is located in the MYSQL-INSTALL-DIR/bin directory, and assumes that Perl and the DBI and DBD::MySQL modules have been installed. This script is bundled only for the Linux/Unix versions of MySQL.

Revoking Previously Assigned Permissions

Sometimes you need to remove one or more previously assigned privileges from a particular user. For example, suppose you want to remove the UPDATE privilege from user rita@192.168.1.102 for the database books:

```
mysql>REVOKE insert ON books.* FROM rita@192.168.1.102;
```

Revoking Table-Level Permissions

Now suppose you want to remove both the previously assigned UPDATE and INSERT privileges from user rita@192.168.1.102 for the table authors located in the database books:

```
mysql>REVOKE insert, update ON books.authors FROM rita@192.168.1.102;
```

Note that this example assumes that you've granted table-level permissions to user rita@192.168.1.102. The REVOKE command will not downgrade a database-level GRANT (one located in the db table), removing the entry and inserting an entry in the tables_priv table. Instead, in this case it simply removes reference to those privileges from the tables_priv table. If only those two privileges are referenced in the tables_priv table, then the entire row is removed.

Revoking Column-Level Permissions

As a final revocation example, suppose that you have previously granted a column-level DELETE permission to user rita@192.168.1.102 for the column name located in books.authors, and now you would like to remove that privilege:

```
mysql>REVOKE insert (name) ON books.authors FROM rita@192.168.1.102;
```

In all of these examples of using REVOKE, it's possible that user rita could still be able to exercise some privileges within a given database if the privileges were not explicitly referenced in the REVOKE command. If you want to be sure that the user forfeits all permissions, you can revoke all privileges, like so:

```
mysql>REVOKE all privileges ON books.* FROM rita@192.168.1.102;
```

However, if your intent is to definitively remove the user from the `mysql` database, be sure to read the next section.

Deleting a User

A common question regarding `REVOKE` is how it goes about deleting a user. The simple answer to this question is that it doesn't at all. For example, suppose that you revoke all privileges from a particular user, using the following command:

```
mysql>REVOKE all privileges ON books.* FROM rita@192.168.1.102;
```

Although this command does indeed remove the row residing in the `db` table pertinent to `rita@192.168.1.102`'s relationship with the `books` database, it does not remove that user's entry from the `user` table, presumably so that you could later reinstate this user without having to reset the password. If you're sure that this user will not be required in the future, you need to manually remove the row by using the `DELETE` command.

Of course, if you're running MySQL 5.0.2 or greater, consider using the `DROP USER` command to delete the user and all privileges simultaneously.

GRANT and REVOKE Tips

The following list offers various tips to keep in mind when you're working with `GRANT` and `REVOKE`:

- You can grant privileges for a database that doesn't yet exist.
- If the user identified by the `GRANT` command does not exist, it will be created.
- If you create a user without including the `IDENTIFIED BY` clause, no password will be required for login.
- If an existing user is granted new privileges, and the `GRANT` command is accompanied by an `IDENTIFIED BY` clause, the user's old password will be replaced with the new one.
- Table-level `GRANT`s only support the following privilege types: `ALTER`, `CREATE`, `CREATE VIEW`, `DELETE`, `DROP`, `GRANT`, `INDEX`, `INSERT`, `REFERENCES`, `SELECT`, `SHOW VIEW`, and `UPDATE`.
- Column-level `GRANT`s only support the following privilege types: `INSERT`, `SELECT`, and `UPDATE`.
- The _ and % wildcards are supported when referencing both database names and host-names in `GRANT` commands. Because the _ character is also valid in a MySQL database name, you need to escape it with a backslash if it's required in the `GRANT`.
- If you want to create and delete users, and are running MySQL 5.0.2 or greater, consider using the `CREATE USER` and `DROP USER` commands instead.
- You can't reference `*.*` in an effort to remove a user's privileges for all databases. Rather, each must be explicitly referenced by a separate `REVOKE` command.

Reviewing Privileges

Although you can review a user's privileges simply by selecting the appropriate data from the privilege tables, this strategy can become increasingly unwieldy as the tables grow in size. Thankfully, MySQL offers a much more convenient means (two, actually) for reviewing user-specific privileges. Both are examined in this section.

SHOW GRANTS FOR

The `SHOW GRANTS FOR` user command displays the privileges granted for a particular user. For example:

```
mysql>SHOW GRANTS FOR rita@192.168.1.102;
```

This yields the table shown in Figure 28-1.

```
+-----------------------------------------------------------------------------------------+
| Grants for rita@192.168.1.102                                                           |
+-----------------------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'rita'@'192.168.1.102' IDENTIFIED BY PASSWORD '428567f408994404'  |
| GRANT INSERT (name) ON `books`.`authors` TO 'rita'@'192.168.1.102'                      |
+-----------------------------------------------------------------------------------------+
2 rows in set (0.00 sec)
```

Figure 28-1. *Typical results of the SHOW GRANTS FOR command*

As with the `GRANT` and `REVOKE` commands, you must make reference to both the username and the originating host in order to uniquely identify the target user.

Limiting User Resources

Monitoring resource usage is always a good idea, but it is particularly important when you're offering MySQL in a hosted environment, such as an ISP. If you're concerned with such a matter, you will be happy to learn that, as of version 4.0.2, it's possible to limit the consumption of MySQL resources on a per-user basis. These limitations are managed like any other privilege, via the privilege tables. In total, four privileges concerning the use of resources exist, all of which are located in the `user` table:

- `max_connections`: Determines the maximum number of times the user can connect to the database per hour
- `max_questions`: Determines the maximum number of queries (using the `SELECT` command) that the user can execute per hour
- `max_updates`: Determines the maximum number of updates (using the `INSERT` and `UPDATE` commands) that the user can execute per hour
- `max_user_connections`: Determines the maximum number of simultaneous connections a given user can maintain (added in version 5.0.3)

Consider a couple examples. The first limits user dario@%.wjgilmore.com's number of connections per hour to 3,600, or an average of one per second:

```
mysql>GRANT insert, select, update ON books.* TO dario@'%.wjgilmore.com'
    ->IDENTIFIED BY 'secret' WITH max_connections_per_hour 3600;
```

The next example limits the total number of updates user dario@'%.wjgilmore.com' can execute per hour to 10,000:

```
mysql>GRANT insert, select, update ON books.* TO dario@'%.wjgilmore.com'
    ->IDENTIFIED BY 'secret' WITH max_updates_per_hour 10000;
```

Secure MySQL Connections

Data flowing between a client and a MySQL server is not unlike any other typical network traffic; it could potentially be intercepted and even modified by a malicious third party. Sometimes this isn't really an issue, because the database server and clients often reside on the same internal network and, for many, on the same machine. However, if your project requirements result in the transfer of data over insecure channels, you now have the option to use MySQL's built-in security features to encrypt that connection. As of version 4.0.0, it became possible to encrypt all traffic between the mysqld server daemon and any client using SSL and the X509 encryption standard.

To implement this feature, you need to complete the following prerequisite tasks first, unless you're running MySQL 5.0.10 or greater, in which case you can skip these tasks; these versions come bundled with yaSSL support, meaning OpenSSL is no longer needed to implement secure MySQL connections. If you are running MySQL 5.0.10 or greater, skip ahead to the following "Grant Options" section. Regardless of whether you're using yaSSL or require OpenSSL, all of the other instructions are identical.

- Install the OpenSSL library, available for download at http://www.openssl.org/.
- Configure MySQL with the --with-vio and --with-openssl flags.

You can verify whether MySQL is ready to handle secure connections by logging in to the MySQL server and executing:

```
mysql>SHOW VARIABLES LIKE 'have_openssl'
```

Once these prerequisites are complete, you need to create or purchase both a server certificate and a client certificate. The processes for accomplishing either task are out of the scope of this book. You can get information about this process on the Internet, so take a few moments to perform a search and you'll turn up numerous resources.

Grant Options

There are a number of grant options that determine the user's SSL requirements. These options are introduced in this section.

REQUIRE SSL

This grant option forces the user to connect over SSL. Any attempts to connect in an insecure fashion will result in an "Access denied" error. An example follows:

```
mysql>GRANT insert, select, update ON company.* TO jason@client.wjgilmore.com
     ->IDENTIFIED BY 'secret' REQUIRE SSL;
```

REQUIRE X509

This grant option forces the user to provide a valid Certificate Authority (CA) certificate. This would be required if you want to verify the certificate signature with the CA certificate. Note that this option does not cause MySQL to consider the origin, subject, or issuer. An example follows:

```
mysql>GRANT insert, select, update on company.* to jason@client.wjgilmore.com
     ->identified by 'secret' REQUIRE SSL REQUIRE X509;
```

Note that this option doesn't specify which CAs are valid and which are not. Any CA that verified the certificate would be considered valid. If you'd like to place a restriction on which CAs are considered valid, see the next grant option.

REQUIRE ISSUER

This grant option forces the user to provide a valid certificate, issued by a valid CA issuer. Several additional pieces of information must be included with this, including the country of origin, state of origin, city of origin, name of certificate owner, and certificate contact. An example follows:

```
mysql>GRANT insert, select, update ON company.* TO jason@client.wjgilmore.com
   ->IDENTIFIED BY 'secret' REQUIRE SSL REQUIRE ISSUER 'C=US, ST=Ohio,
   ->L=Columbus, O=WJGILMORE,
   ->OU=ADMIN, CN=db.wjgilmore.com/Email=admin@wjgilmore.com'
```

REQUIRE SUBJECT

This grant option forces the user to provide a valid certificate including a valid certificate "subject." An example follows:

```
mysql>GRANT insert, select, update ON company.* TO jason@client.wjgilmore.com
    ->IDENTIFIED BY 'secret' REQUIRE SSL REQUIRE SUBJECT
    ->'C=US, ST=Ohio, L=Columbus, O=WJGILMORE, OU=ADMIN,
    ->CN=db.wjgilmore.com/Email=admin@wjgilmore.com'
```

REQUIRE CIPHER

This grant option enforces the use of recent encryption algorithms by forcing the user to connect using a particular cipher. The options currently available include: EDH, RSA, DES, CBC3, and SHA. An example follows:

```
mysql>GRANT insert, select, update ON company.* TO jason@client.wjgilmore.com
      ->IDENTIFIED BY 'secret' REQUIRE SSL REQUIRE CIPHER 'DES-RSA';
```

SSL Options

The options introduced in this section are used by both the server and the connecting client to determine whether SSL should be used, and if so, the location of the certificate and key files.

--ssl

This option simply acts as a signal that SSL should be used. More specifically, when used in conjunction with the mysqld daemon, it tells the server that SSL connections should be allowed. Used in conjunction with the client, it signals that an SSL connection will be used. Note that including this option does not ensure, nor require, that an SSL connection is used. In fact, tests have shown that the option itself is not even required to initiate an SSL connection. Rather, the accompanying flags, introduced here, determine whether an SSL connection is successfully initiated.

--ssl-ca

This option specifies the location and name of a file containing a list of trusted SSL certificate authorities. For example:

```
--ssl-ca=/home/jason/openssl/cacert.pem
```

--ssl-capath

This option specifies the directory path where trusted SSL certificates in privacy-enhanced mail (PEM) format are stored.

--ssl-cert

This option specifies the location and name of the SSL certificate used to establish the secure connection. For example:

```
--ssl-cert=/home/jason/openssl/mysql-cert.pem
```

--ssl-cipher

This option specifies which encryption algorithms are allowable. The cipher-list syntax is the same as that used by the following command:

```
%>openssl ciphers
```

For example, to allow just the TripleDES and Blowfish encryption algorithms, this option would be set as follows:

```
--ssl-cipher=des3:bf
```

--ssl-key

This option specifies the location and name of the SSL key used to establish the secure connection. For example:

```
--ssl-key=/home/jason/openssl/mysql-key.pem
```

In the next three sections, you'll learn how to use these options on both the command line and within the `my.cnf` file.

Starting the SSL-Enabled MySQL Server

Once you have both the server and client certificates in hand, you can start the SSL-enabled MySQL server like so:

```
%>./bin/mysqld_safe --user=mysql --ssl-ca=$SSL/cacert.pem \
 >--ssl-cert=$SSL/server-cert.pem --ssl-key=$SSL/server-key.pem &
```

`$SSL` refers to the path pointing to the SSL certificate storage location.

Connecting Using an SSL-Enabled Client

You can then connect to the SSL-enabled MySQL server by using the following command:

```
%>mysql --ssl-ca=$SSL/cacert.pem --ssl-cert=$SSL/client-cert.pem \
->--ssl-key=$SSL/client-key.pem -u jason -h www.wjgilmore.com -p
```

Again, `$SSL` refers to the path pointing to the SSL certificate storage location.

Storing SSL Options in the my.cnf File

Of course, you don't have to pass the SSL options via the command line. Instead, you can place them within a `my.cnf` file. An example `my.cnf` file follows:

```
[client]
ssl-ca     = /home/jason/ssl/cacert.pem
ssl-cert   = /home/jason/ssl/client-cert.pem
ssl-key    = /home/jason/ssl/client-key.pem

[mysqld]
ssl-ca     = /usr/local/mysql/ssl/ca.pem
ssl-cert   = /usr/local/mysql/ssl/cert.pem
ssl-key    = /usr/local/mysql/openssl/key.pem
```

FREQUENTLY ASKED QUESTIONS

Because the SSL feature is so new, there is still some confusion surrounding its usage. This FAQ attempts to offer some relief by answering some of the most commonly asked questions regarding this topic.

I'm using MySQL solely as a back end to my Web application, and I am using HTTPS to encrypt traffic to and from the site. Do I also need to encrypt the connection to the MySQL server?

This depends on whether the database server is located on the same machine as the Web server. If this is the case, then encryption will likely be beneficial only if you consider your machine itself to be insecure. If the database server resides on a separate server, then the data could potentially be traveling unsecured from the Web server to the database server, and therefore it would warrant encryption. There is no steadfast rule regarding the use of encryption. You can reach a conclusion only after carefully weighing security and performance factors.

I understand that encrypting Web pages using SSL will degrade performance. Does the same hold true for the encryption of MySQL traffic?

Yes, your application will take a performance hit, because every data packet must be encrypted while traveling to and from the MySQL server.

How do I know that the traffic is indeed encrypted?

The easiest way to ensure that the MySQL traffic is encrypted is to create a user account that requires SSL, and then try to connect to the SSL-enabled MySQL server by supplying that user's credentials and a valid SSL certificate. If something is awry, you'll receive an "Access denied" error.

On what port does encrypted MySQL traffic flow?

The port number remains the same (3306) regardless of whether you're communicating in encrypted or unencrypted fashion.

Summary

An uninvited database intrusion can wipe away months of work and erase inestimable value. Therefore, although the topics covered in this chapter generally lack the glamour of other feats, such as creating a database connection and altering a table structure, the importance of taking the time to thoroughly understand these security topics cannot be understated. It's strongly recommended that you take adequate time to understand MySQL's security features, because they should be making a regular appearance in all of your MySQL-driven applications.

The next chapter introduces PHP's MySQL library, showing you how to manipulate MySQL database data through your PHP scripts. That chapter is followed by an introduction to the MySQLi library, which should be used if you're running PHP 5 and MySQL 4.1 or greater.