# CHAPTER 14

# Security, Access Control, and Privileges

Most users concentrate on MySQL's databases and tables—after all, that's where most of the action takes place—and they don't usually look deeper to understand how it handles access privileges, passwords, and security. This approach is usually more than adequate for most development activities—unless you happen to be a database administrator whose job involves setting up and securing the databases against unauthorized usage or malicious mischief.

With that in mind, this chapter examines the MySQL access control system and throws some light on the MySQL grant tables. These tables, which are an integral part of the server's security system, offer database administrators a great deal of power and flexibility in deciding the rules that govern access to the system. Additionally, this chapter also discusses the management of user accounts and passwords in the MySQL access control system, explaining how passwords (especially the all-important root password) can be modified and how to reset a lost superuser password.

## The MySQL Grant Tables

When MySQL is first installed, the MySQL installer automatically creates two databases: the `test` database, which serves as a sandbox for new users to play in, and the `mysql` database, which contains the five MySQL grant tables.

This is clearly visible from the following sample output:

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
+----------+
```

```
2 rows in set (0.00 sec)
mysql> SHOW tables FROM mysql;
+-----------------+
| Tables_in_mysql |
+-----------------+
| columns_priv    |
| db              |
| func            |
| host            |
| tables_priv     |
| user            |
+-----------------+
6 rows in set (0.00 sec)
```

With the exception of the `func` table (used for user-defined functions), each of these tables has a different role to play in deciding whether a user has access to a specific database, table, or table column. Access rules can be set up on the basis of username, connecting host, or database requested.

The following sections examine each of these tables in greater detail.

## The `user` Table

Of the five grant tables, the most important one is the `user` table. Here is a list of the fields it contains.

```
CREATE TABLE `user` (
  `Host` varchar(60) binary NOT NULL default '',
  `User` varchar(16) binary NOT NULL default '',
  `Password` varchar(45) binary NOT NULL default '',
  `Select_priv` enum('N','Y') NOT NULL default 'N',
  `Insert_priv` enum('N','Y') NOT NULL default 'N',
  `Update_priv` enum('N','Y') NOT NULL default 'N',
  `Delete_priv` enum('N','Y') NOT NULL default 'N',
  `Create_priv` enum('N','Y') NOT NULL default 'N',
  `Drop_priv` enum('N','Y') NOT NULL default 'N',
  `Reload_priv` enum('N','Y') NOT NULL default 'N',
  `Shutdown_priv` enum('N','Y') NOT NULL default 'N',
  `Process_priv` enum('N','Y') NOT NULL default 'N',
  `File_priv` enum('N','Y') NOT NULL default 'N',
  `Grant_priv` enum('N','Y') NOT NULL default 'N',
  `References_priv` enum('N','Y') NOT NULL default 'N',
  `Index_priv` enum('N','Y') NOT NULL default 'N',
  `Alter_priv` enum('N','Y') NOT NULL default 'N',
  `Show_db_priv` enum('N','Y') NOT NULL default 'N',
  `Super_priv` enum('N','Y') NOT NULL default 'N',
  `Create_tmp_table_priv` enum('N','Y') NOT NULL default 'N',
```

```
  `Lock_tables_priv` enum('N','Y') NOT NULL default 'N',
  `Execute_priv` enum('N','Y') NOT NULL default 'N',
  `Repl_slave_priv` enum('N','Y') NOT NULL default 'N',
  `Repl_client_priv` enum('N','Y') NOT NULL default 'N',
  `ssl_type` enum('','ANY','X509','SPECIFIED') NOT NULL default '',
  `ssl_cipher` blob NOT NULL,
  `x509_issuer` blob NOT NULL,
  `x509_subject` blob NOT NULL,
  `max_questions` int(11) unsigned NOT NULL default '0',
  `max_updates` int(11) unsigned NOT NULL default '0',
  `max_connections` int(11) unsigned NOT NULL default '0',
  PRIMARY KEY  (`Host`,`User`)
) TYPE=MyISAM COMMENT='Users and global privileges'
```

The first three fields (referred to as *scope fields*) define which users are allowed to connect to the database server, their passwords, and the hosts from which they can connect—MySQL uses a combination of both user and host identification as the basis for its security system.

Consider the following extract from this table:

```
+--------------------+--------+----------+
| Host               | User   | Password |
+--------------------+--------+----------+
| turkey.domain.com  | john   |          |
+--------------------+--------+----------+
```

This implies that the user `john` (password null) is allowed to connect from the host `turkey.domain.com`.

IP addresses can be used instead of host names. The following is a perfectly valid entry:

```
+--------------------+--------+----------+
| Host               | User   | Password |
+--------------------+--------+----------+
| 240.56.78.99       | john   |          |
+--------------------+--------+----------+
```

**What's in a Name?**

Note, MySQL users are not the same as system users, on either Windows or UNIX. MySQL users exist only within the context of the MySQL RDBMS and need not have accounts or home directories on the system. While the MySQL command-line client on UNIX does default to using the currently logged-in user's name to connect to the server, this behavior can be overridden by specifying a user name to the client via the `--user` parameter.

It's also possible to specify wildcards when setting up such access rules. The following example would allow access to a user named `john`, regardless of the host from which the connection is requested.

```
+-----------------+--------+
| Host  | User | Password  |
+-----------------+--------+
| %     | john |           |
+-----------------+--------+
```

The % character is used as a wildcard. The following example would match any user named `john` connecting from any host in the `loudbeep.com` domain.

```
+---------------+-----+-----------+
| Host        | User | Password   |
+---------------+-----+-----------+
| %.loudbeep.com     | john |     |
+---------------+-----+-----------+
```

Once the users, passwords, and hosts are specified, it becomes necessary to specify the privileges each user has—which is where the other 21 columns (or *privilege fields*) come in. Table 14–1 specifies what each of these fields represents.

The remaining fields in the `user` table are related to SSL encryption and resource usage limits per user—some of these are discussed in the section "Limiting Resource Usage."

At this point, note that the security privileges assigned to each user in the `user` table are globally valid; they apply to every database on the system. Therefore, the record

```
+-----------------+------+-----------+--------------+
| Host            | User | Password  | Delete_priv  |
+-----------------+------+-----------+--------------+
| apple.pie.com   | joe  | secret    | Y            |
+-----------------+------+-----------+--------------+
```

would imply that user `joe` has the ability to `DELETE` records from any table in any database on the server—not a Good Thing if Joe happens to be in a bad mood. For this reason, most administrators (and the MySQL manual) recommend leaving all privileges in this table to *N* (the default value) for every user, and using the `host` and `db` tables to assign more focused levels of access (the `host` and `db` tables are discussed in the section "The `db` and `host` Tables").

---

***TIP*** *Wondering how to create users and maintain user passwords? Flip to the sections "Granting, Revoking, and Viewing User Privileges" and "Changing User Passwords" for detailed instructions.*

| Field | Privilege Name | Users With This Privilege Can |
|---|---|---|
| Select_priv | SELECT | Execute a SELECT query to retrieve rows from a table |
| Insert_priv | INSERT | Execute an INSERT query |
| Update_priv | UPDATE | Execute an UPDATE query |
| Delete_priv | DELETE | Execute a DELETE query |
| Create_priv | CREATE | CREATE databases and tables |
| Drop_priv | DROP | DROP databases and tables |
| Reload_priv | RELOAD | Reload/refresh the MySQL server |
| Shutdown_priv | SHUTDOWN | Shut down a running MySQL server |
| Process_priv | PROCESS | Track activity on a MySQL server |
| File_priv | FILE | Read and write files on the server |
| Grant_priv | GRANT | GRANT other users the same privileges the user possesses |
| Index_priv | INDEX | Create, edit, and delete table indexes |
| Alter_priv | ALTER | ALTER tables |
| References_priv | REFERENCES | Create, edit, and delete foreign key references |
| Show_db_priv | SHOW DATABASES | View available databases on the server |
| Super_priv | SUPER | Execute administrative commands |
| Create_tmp_table_priv | CREATE TEMPORARY TABLES | Create temporary tables |
| Lock_tables_priv | LOCK TABLES | Create and delete table locks |
| Execute_priv | EXECUTE | Execute stored procedures |
| Repl_slave_priv | REPLICATION SLAVE | Read master binary logs in a replication context |
| Repl_client_priv | REPLICATION CLIENT | Request information on masters and slaves in a replication context |

**TABLE 14-1** MySQL Privilege Levels

Here's another example—the following record implies the existence of a superuser named `superapple`, with complete access to all MySQL privileges.

```
+-----------------+-------------+-----------+----------+
| Host            | User        | Password  | All_priv |
+-----------------+-------------+-----------+----------+
| apple.pie.com   | superapple  | secret    | Y        |
+-----------------+-------------+-----------+----------+
```

***NOTE*** *An `All_priv` column doesn't appear in any of the grant tables. We've simply used it as shorthand in these examples to indicate the same value is present in all the privilege (`*_priv`) columns.*

## The `db` and `host` Tables

The `host` and `db` tables are used together—they control which databases are available to which users, and which operations are possible on those databases. Take a look at the fields in a typical `db` table:

```
CREATE TABLE `db` (
  `Host` char(60) binary NOT NULL default '',
  `Db` char(64) binary NOT NULL default '',
  `User` char(16) binary NOT NULL default '',
  `Select_priv` enum('N','Y') NOT NULL default 'N',
  `Insert_priv` enum('N','Y') NOT NULL default 'N',
  `Update_priv` enum('N','Y') NOT NULL default 'N',
  `Delete_priv` enum('N','Y') NOT NULL default 'N',
  `Create_priv` enum('N','Y') NOT NULL default 'N',
  `Drop_priv` enum('N','Y') NOT NULL default 'N',
  `Grant_priv` enum('N','Y') NOT NULL default 'N',
  `References_priv` enum('N','Y') NOT NULL default 'N',
  `Index_priv` enum('N','Y') NOT NULL default 'N',
  `Alter_priv` enum('N','Y') NOT NULL default 'N',
  `Create_tmp_table_priv` enum('N','Y') NOT NULL default 'N',
  `Lock_tables_priv` enum('N','Y') NOT NULL default 'N',
  PRIMARY KEY  (`Host`,`Db`,`User`),
  KEY `User` (`User`)
) TYPE=MyISAM COMMENT='Database privileges'
```

Again, the first three fields are scope fields, which link a specific user and host to one or more databases. The remaining privilege fields are used to specify the type of operations the user can perform on the named database (refer to Table 14–1 for details on what each field represents).

A record like the following one would imply that the user `bill`, connecting from host `cranberry.domain.com`, would be able to use the database `darkbeast` only:

```
+----------------------+------+-----------+---------+
| Host            |       User | Db        | All_priv|
+----------------------+------+-----------+---------+
| cranberry.domain.com  | bill | darkbeast | Y       |
+----------------------+------+-----------+---------+
```

On the other hand, this next record would imply that any user, connecting from any host, would have complete access to the `test` database:

```
+------+------+---------+-----------+
| Host | User | Db      | All_priv |
+------+------+---------+-----------+
| %    |      | test    | Y         |
+------+------+---------+-----------+
```

A blank entry in the `Host` field of the `db` table implies that the list of allowed hosts should be obtained from the third table, the `host` table, which looks like this:

```
CREATE TABLE `host` (
  `Host` char(60) binary NOT NULL default '',
  `Db` char(64) binary NOT NULL default '',
  `Select_priv` enum('N','Y') NOT NULL default 'N',
  `Insert_priv` enum('N','Y') NOT NULL default 'N',
  `Update_priv` enum('N','Y') NOT NULL default 'N',
  `Delete_priv` enum('N','Y') NOT NULL default 'N',
  `Create_priv` enum('N','Y') NOT NULL default 'N',
  `Drop_priv` enum('N','Y') NOT NULL default 'N',
  `Grant_priv` enum('N','Y') NOT NULL default 'N',
  `References_priv` enum('N','Y') NOT NULL default 'N',
  `Index_priv` enum('N','Y') NOT NULL default 'N',
  `Alter_priv` enum('N','Y') NOT NULL default 'N',
  `Create_tmp_table_priv` enum('N','Y') NOT NULL default 'N',
  `Lock_tables_priv` enum('N','Y') NOT NULL default 'N',
  PRIMARY KEY  (`Host`, `Db`)
) TYPE=MyISAM COMMENT='Host privileges;  Merged with database privileges'
```

This separation between host records and database records is more useful than you might think. In the absence of the `host` table, if you want the same user to have different privileges based on the host from which he or she is connecting, you would need to create a separate record for each host in the `db` table and assign privileges accordingly. However, because the `host` table exists, you can place the various host names in the `host` table, link them to a single entry (with a blank `Host` field) in the `db` table, and then set privileges on a per-host basis. When a connection is attempted from one of the named hosts, MySQL will assign privileges based on the rules set for that host in the `host` table.

Here's an example of how this works (the first snippet is from the `db` table, the second is the corresponding records in the `host` table):

```
+------+------+-------+
| Host | User | Db    |
+------+------+-------+
|      | jim  | title |
+------+------+-------+
```

```
+------------------+-------+----------------+----------------+
| Host             | Db    | Select_priv    | Insert_priv    |
+------------------+-------+----------------+----------------+
| turkey.ix6.com   | title | Y              | Y              |
+------------------+-------+----------------+----------------+
| blackbox.glue.net | title | Y             | N              |
+------------------+-------+----------------+----------------+
| fireball.home.net | title | Y             | Y              |
+------------------+-------+----------------+----------------+
```

In this case, `jim` will be able to connect to the MySQL server from any of the hosts listed in the `host` table, but the privileges he has will differ on the basis of the host from which he is connecting.

## The `tables_priv` and `columns_priv` Tables

In addition to the three tables already discussed, newer versions of MySQL also come with two additional tables: the `tables_priv` and `columns_priv` tables. These allow a database administrator to restrict access to specific tables in a database and to specific columns of a table, respectively.

Here's what the `tables_priv` table looks like:

```
CREATE TABLE `tables_priv` (
  `Host` char(60) binary NOT NULL default '',
  `Db` char(64) binary NOT NULL default '',
  `User` char(16) binary NOT NULL default '',
  `Table_name` char(60) binary NOT NULL default '',
  `Grantor` char(77) NOT NULL default '',
  `Timestamp` timestamp(14) NOT NULL,
  `Table_priv` set('Select', 'Insert', 'Update', 'Delete', 'Create',
'Drop', 'Grant', 'References', 'Index', 'Alter') NOT NULL default '',
  `Column_priv` set('Select', 'Insert', 'Update', 'References')
NOT NULL default '',
  PRIMARY KEY  (`Host`, `Db`, `User`, `Table_name`),
  KEY `Grantor` (`Grantor`)
) TYPE=MyISAM COMMENT='Table privileges'
```

The following record would restrict the user `john` to performing `SELECT` operations on table `cream` only—any attempt to run a `SELECT` query on another table within the same database would result in an error.

```
+-----------------+-------+------+------------+---------------+
| Host            | Db    | User | Table_name | Table_priv    |
+-----------------+-------+------+------------+---------------+
|lost.soul.com    | db563 | john | cream      | Select        |
+-----------------+-------+------+------------+---------------+
```

Similarly, the following record would allow the user `logger` to perform `SELECT`, `INSERT`, and `UPDATE` (but not `DELETE`) queries on the table named `logs` in the `db1` database:

```
+-----------+-----+----------+------------+----------------------+
| Host      | Db  | User     | Table_name | Table_priv           |
+-----------+-----+----------+------------+----------------------+
| localhost | db1 | logger   | logs       | Select,Insert,Update |
+-----------+-----+----------+------------+----------------------+
```

For even more fine-grained control, MySQL offers the `columns_priv` table, which makes it possible to set access privileges on a per-column basis—as the following table structure illustrates.

```
CREATE TABLE `columns_priv` (
  `Host` char(60) binary NOT NULL default '',
  `Db` char(64) binary NOT NULL default '',
  `User` char(16) binary NOT NULL default '',
  `Table_name` char(64) binary NOT NULL default '',
  `Column_name` char(64) binary NOT NULL default '',
  `Timestamp` timestamp(14) NOT NULL,
  `Column_priv` set('Select', 'Insert', 'Update', 'References')
NOT NULL default '',
  PRIMARY KEY  (`Host`, `Db`, `User`, `Table_name`, `Column_name`)
) TYPE=MyISAM COMMENT='Column privileges'
```

The following is a case in point. The following rules imply that users logging in with the account `hr_users` can read only the employee ID, employee name, and department from the `db1.employees` table, while supervisors (using the account `hr_supervisors`) can additionally view the employee's compensation and update his or her name and department information.

```
+-----+----------------+------------+-------------+-----------------+
| Db  | User           | Table_name | Column_name |  Column_priv    |
+-----+----------------+------------+-------------+-----------------+
| db1 | hr_supervisors | employees  | name        | Select, Update |
| db1 | hr_supervisors | employees  | id          | Select         |
| db1 | hr_supervisors | employees  | dept        | Select, Update |
| db1 | hr_users       | employees  | id          | Select         |
| db1 | hr_users       | employees  | name        | Select         |
| db1 | hr_users       | employees  | dept        | Select         |
+-----+----------------+------------+-------------+-----------------+
```

### Interaction Between the Grant Tables

The various grant tables interact with each other to create comprehensive access rules that MySQL uses when deciding how to handle a particular user request. Access control takes place at two stages: the connection stage and the request stage.

- **The connection stage** When a user requests a connection to the database server from a specific host, MySQL will first check whether an entry exists for

the user in the `user` table, if the user's password is correct, and if the user is allowed to connect from that specific host. If the check is successful, a connection will be allowed to the server.

- **The request stage** Once a connection is allowed, every subsequent request to the server—`SELECT`, `DELETE`, `UPDATE`, and other queries—will first be vetted to ensure that the user has the security privileges necessary to perform the corresponding action. A number of different levels of access are possible—some users might only have the ability to `SELECT` from the tables, while others might have `INSERT` and `UPDATE` capabilities, but not `DELETE` capabilities.

In the hierarchy of the MySQL grant tables, the `user` table comes first, with the `db` and `host` tables below it, and the `tables_priv` and `columns_priv` tables at the bottom. A table at a lower level is referred to only if a higher-level table fails to provide the necessary scope or privileges.

When deciding whether to allow a particular database operation, MySQL takes the privilege fields in all three tables into account. It starts with the `user` table and checks to see if the user has appropriate privileges for the operation being attempted. If not, the `db` and `host` tables are checked to see if privileges are available. Only after logically parsing the privileges in the different tables does MySQL allow or disallow a specific database request.

When MySQL encounters a request for an administrative action—`RELOAD`, `PROCESS`, and so forth—by a user, it decides whether to permit that action based solely on the corresponding permissions for that user in the `user` table. None of the other grant tables are consulted to make this determination. This is because these administrative privileges apply to the system as a whole and not to specific databases or tables, therefore, the corresponding columns make an appearance in the `user` table only.

### The Default Setup

Now, let's look at the default tables that ship with MySQL, so you can understand the implications of running an out-of-the-box MySQL setup.

```
mysql> SELECT Host, User, Password FROM mysql.user;
+-----------+------+----------+
| Host      | User | Password |
+-----------+------+----------+
| localhost | root |          |
| localhost |      |          |
+-----------+------+----------+
2 rows in set (0.00 sec)
mysql> SELECT Host, Db, User FROM mysql.db;
+------+---------+------+
| Host | Db      | User |
+------+---------+------+
| %    | test    |      |
| %    | test\_% |      |
+------+---------+------+
```

What this means is, out of the box:

1. MySQL gives the user connecting as `root` from the local machine `localhost` complete access to all databases on the system.
2. MySQL gives any other user connecting from the local machine complete access (a) to all databases, on Windows or (b) to the `test` database, on UNIX.
3. Users from other hosts are denied access.

## Granting, Revoking, and Viewing User Privileges

Modifying the grant tables in the `mysql` database requires superuser access to the MySQL database server. So, the first order of business is to ensure that you have this level of access and can alter table records.

If you've installed the server yourself, on your own development machine, you would have been told to enter a root password at the time of installation (Chapter 3 has details on this process). If you paid attention to the installation instructions in Chapter 3, you would have done this (leaving the password blank opens up a security hole in the system), and noted the password you set for future reference.

To verify that you have the required access, log in to the server as the `root` user, as in the following:

```
[root@host] $ mysql -u root -p
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
mysql>
```

Now, ensure that you can access the grant tables in the `mysql` database:

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

Such root-level access is typically available only to the database administrator. Other users have a lower security rating and, consequently, limited access. Each of these "ordinary" users will typically connect to the database by supplying his or her own user name and password. As noted in the previous section, the purpose of the MySQL grant tables is to make it possible to manipulate security settings for these ordinary users and to customize each user's level of access to a fine degree.

MySQL offers two methods of altering access rights in the grant tables—you can either use `INSERT`, `UPDATE`, and `DELETE` DML queries to hand-alter the information in the tables or you can use the `GRANT` and `REVOKE` commands. The latter is the preferred method; direct modification of the grant tables is advisable only for unusual tasks or situations, and is generally not recommended.

## Using the GRANT and REVOKE Commands

The recommended way of setting access privileges for a user in the MySQL grant tables is via the MySQL GRANT and REVOKE commands, designed specifically for the task. Here's what they look like:

```
GRANT privilege (field-name, field-name, ...), privilege (field-name, field-name,
...), ... ON database-name.table-name TO user@domain IDENTIFIED BY password,
user@domain IDENTIFIED BY password, ...
REVOKE privilege (field-name, field-name, ...), privilege (field-name, field-name,
...), ... ON database-name.table-name FROM user@domain, user@domain, ...
```

To illustrate this, let's consider a few examples. This first example assigns SELECT, INSERT, UPDATE, and DELETE privileges on the table db1.logs to the user logger connecting from localhost with password timber:

```
mysql> GRANT SELECT, INSERT, UPDATE ON db1.logs TO logger@localhost
IDENTIFIED BY 'timber';
```

Now, look at what happens when this user logs in to MySQL and tries attempting different types of queries:

```
[user@host] $ mysql -h localhost -u logger -p
Enter password: ******
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 4.0.14
Type 'help;' or '\h' for help. Type '\c' to clear the buffer
mysql> USE mysql;
ERROR 1044: Access denied for user: 'logger@localhost' to database 'mysql'
mysql> USE db1;
Database changed
mysql> SELECT * FROM employees;
ERROR 1142: select command denied to user: 'logger@localhost' for table 'employees'
mysql> SELECT * FROM logs;
+----+---------------------------+--------+
| id | message                   | sender |
+----+---------------------------+--------+
| 34 | Apache core error         | httpd  |
| 35 | Root login failure on tty1 | pamd   |
| 36 | Root login failure on tty1 | pamd   |
+----+---------------------------+--------+
3 rows in set (0.00 sec)

mysql> INSERT INTO logs VALUES (37, 'Sendmail restart, 102 messages in queue', 'sendmail');
Query OK, 1 row affected (0.00 sec)
mysql> DELETE FROM logs WHERE id = 37;
ERROR 1142: delete command denied to user: 'logger@localhost' for table 'logs'
```

Thus, only the commands specified in the GRANT command are permitted to the user. All other commands are denied. Every time the user requests a specific command, MySQL refers to its grant tables and only permits the command to be executed if the privilege rules allow it.

The REVOKE command does the opposite of the GRANT command, making it possible to revoke privileges assigned to a user. Consider the following example, which rescinds logger's INSERT and UPDATE rights on the db1.logs table:

```
mysql> REVOKE INSERT, UPDATE ON db1.logs FROM logger@localhost;
```

This change goes into effect immediately. Now, when logger tries to INSERT a new record into the table (an operation previously permitted), look at what happens:

```
mysql> INSERT INTO logs VALUES (38, 'System powerdown signal received', 'apmd');
ERROR 1142: insert command denied to user: 'logger@localhost' for table 'logs'
```

MySQL allows the use of the * wildcard when referring to databases and tables—this next query assigns RELOAD, PROCESS, SELECT, DELETE, and INSERT privileges to all databases to the user admin on the host named medusa:

```
mysql> GRANT RELOAD, PROCESS, SELECT, DELETE, INSERT ON *.* TO admin@medusa
IDENTIFIED BY 'secret';
```

This next example assigns SELECT privileges on the table employees.compensation to the supervisor user only:

```
mysql> GRANT SELECT ON employees.compensation TO supervisor;
```

This next example takes things one step further, assigning SELECT and UPDATE privileges to specific columns of the grades table to harry and john, respectively:

```
mysql> GRANT SELECT (id, name, subj, grade) ON db1.grades TO harry;
mysql> GRANT SELECT (id, name, subj, grade), UPDATE (name, grade) ON
db1.grades TO john;
Query OK, 0 rows affected (0.00 sec)
```

The following example rescinds user tim's CREATE and DROP rights on database db2003a:

```
mysql> REVOKE CREATE, DROP ON db2003a.* FROM tim@funhouse.com;
```

### Turning the Tables

The table(s) or field(s) named in the GRANT command must exist as a prerequisite to assigning corresponding table- and column-level privileges. However, this rule does not hold true when dealing with database-level privileges. MySQL permits you to assign database-level privileges, even if the corresponding database does not exist. This dissonance between table- and database-level privileges is a common cause of newbie error, so be forewarned!

As with databases, so with tables and columns. This next example revokes the `webmaster` user's `DELETE` rights on the `menu` table:

```
mysql> REVOKE DELETE ON www.menu FROM webmaster@localhost;
```

And this one removes the user `sarah`'s rights to `UPDATE` the `name` and `address` columns of the `customer` database:

```
mysql> REVOKE UPDATE (name, address) ON sales.customers FROM
sarah@work.domain.net;
```

MySQL also provides the `ALL` privilege level as shorthand for "all privileges"; this can help to make your `GRANT` and `REVOKE` statements more compact. This is illustrated in the next example, which assigns all privileges on the database named `web` to the user `www` connecting from any host in the `melonfire.com` domain:

```
mysql> GRANT ALL ON web.* TO www@'%.melonfire.com' IDENTIFIED BY
'abracadabra';
```

In a similar manner, the `REVOKE` command can be used to revoke all privileges for a specific user:

```
mysql> REVOKE ALL ON web.* FROM www@'%.melonfire.com';
```

MySQL also provides the special `USAGE` privilege level for situations when you need to create a user without assigning him or her any privileges. Here are two examples:

```
mysql> GRANT USAGE ON content.* TO joe@localhost;
mysql> GRANT USAGE ON *.* TO test@some.domain.com;
```

### The `GRANT` Privilege

MySQL lets users grant other users the same privileges they have, via the special `WITH GRANT OPTION` clause of the `GRANT` command. When this clause is added to a `GRANT` command, users to whom it applies can assign the same privileges they possess to other users. Consider the following example:

```
mysql> GRANT SELECT, DELETE, INSERT, UPDATE, CREATE, DROP, INDEX ON
inventory.* TO david@localhost WITH GRANT OPTION;
```

> **Wild at Heart**
>
> When using wildcards like `%` and _ in host names or database names with the `GRANT` and `REVOKE` commands, MySQL requires you to surround the corresponding name in quotes. Wildcards are not, however, supported for user names. To create an anonymous user, use the empty string `''` as the user name.

**Manual Labor**
When the GRANT command is invoked for a particular user, it automatically creates an entry for that user in the user table, if one does not already exist. The corresponding REVOKE command does not delete that entry from the user table, however, even if its invocation results in all the user's privileges being stripped. Thus, though a user record can be automatically added to the system via GRANT, it is never automatically removed using REVOKE. A user record must always be manually deleted using a DELETE query.

The user david@localhost will now have the GRANT privilege and can assign his rights to other users, as clearly demonstrated in the following snippet:

```
mysql> SELECT Host, Db, User, Grant_priv FROM db WHERE Host = 'localhost';
+-----------+-----------+-------+------------+
| Host      | Db        | User  | Grant_priv |
+-----------+-----------+-------+------------+
| localhost | inventory | david | Y          |
+-----------+-----------+-------+------------+
1 row in set (0.00 sec)
```



The user david@localhost can now log in to MySQL and GRANT other users all or some of the privileges he himself possesses, as in the following:

```
mysql> GRANT SELECT ON inventory.* TO joe@localhost IDENTIFIED BY 'joey2839';
mysql> SELECT Host, Db, User, Grant_priv FROM db WHERE Host = 'localhost';
+-----------+-----------+-------+------------+
| Host      | Db        | User  | Grant_priv |
+-----------+-----------+-------+------------+
| localhost | inventory | david | Y          |
| localhost | inventory | joe   | Y          |
+-----------+-----------+-------+------------+
2 rows in set (0.00 sec)
```

The GRANT privilege can be reversed by using the GRANT OPTION clause in a standard REVOKE command, as the following shows:

```
mysql> REVOKE GRANT OPTION ON inventory.* FROM david@localhost;
```

The user david@localhost will now no longer have the capability to grant other users privileges. This is clearly visible from both the user table snippet and the output of the GRANT command run by him, as shown in the following:

```
mysql> GRANT SELECT ON inventory.* TO joe@localhost IDENTIFIED BY 'joey2839';
ERROR 1044: Access denied for user: 'david@localhost' to database 'inventory'
```

```
mysql> SELECT Host, Db, User, Grant_priv FROM db WHERE User = 'david';
+-----------+-----------+-------+------------+
| Host      | Db        | User  | Grant_priv |
+-----------+-----------+-------+------------+
| localhost | inventory | david | N          |
+-----------+-----------+-------+------------+
2 rows in set (0.00 sec)
```

Care should be taken when assigning users the GRANT privilege. Users with different access levels can combine them and, thereby, obtain a higher level of access than they are normally allowed.

### Limiting Resource Usage

New in MySQL 4.*x* is the capability to limit resource usage on the MySQL server, on a per-user basis. This is accomplished by the addition of three new fields to the user table: max_questions, max_updates, and max_connections, which can be used to limit the number of queries, table or record updates, and new connections by a particular user per hour, respectively. These three fields map into three optional clauses to the GRANT command.

The first of these is the MAX_QUERIES_PER_HOUR clause, which limits the number of queries that can be run by a user in an hour. Here's an example:

```
mysql> GRANT SELECT ON *.* TO sarah WITH MAX_QUERIES_PER_HOUR 5;
```

The MAX_QUERIES_PER_HOUR clause controls the total number of queries permitted per hour, regardless of whether these are SELECT, INSERT, UPDATE, DELETE, or other queries. If this is too all-encompassing, you can also set a limit on the number of queries that change the data in the database, via the MAX_UPDATES_PER_HOUR clause, as in the following:

```
mysql> GRANT SELECT ON *.* TO sarah WITH MAX_UPDATES_PER_HOUR 5;
```

The number of new connections opened by the named user(s) in an hour can be controlled via the MAX_CONNECTIONS_PER_HOUR clause, as the following shows.

```
mysql> GRANT SELECT ON *.* TO sarah WITH MAX_CONNECTIONS_PER_HOUR 3;
```

You can also use these clauses in combination with each other. The following is a perfectly valid GRANT:

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE ON *.* TO supervisor WITH
MAX_QUERIES_PER_HOUR 50 MAX_UPDATES_PER_HOUR 10 MAX_CONNECTIONS_PER_HOUR 4;
```

Note, such usage limits cannot be specified per-database or per-table. They can only be specified in the global context, by using an ON *.* clause in the GRANT command.

Here's an example of what MySQL says when a resource limit is exceeded:

```
mysql> INSERT INTO customers (id, name) VALUES (2892, 'Iola J');
ERROR 1226: User 'sarah' has exceeded the 'max_questions' resource
(current value: 5)
```

The server maintains internal counters, on a per-user basis, for each of these three resource limits. These counters could be reset at any time with the new `FLUSH USER_RESOURCES` command, as in the following:

```
mysql> FLUSH USER_RESOURCES;
Query OK, 0 rows affected (0.00 sec)
```

---

***NOTE*** *You need the `RELOAD` privilege to execute the `FLUSH` command.*

## Using the `INSERT`, `UPDATE`, and `DELETE` Commands

The `GRANT` and `REVOKE` commands described in the preceding section are the recommended way of making changes to the MySQL grant tables. However, because the grant tables are, ultimately, regular MySQL tables, you can also manipulate them using standard `INSERT`, `UPDATE`, and `DELETE` queries.

Therefore, while you can certainly use the following `GRANT` command,

```
mysql> GRANT SELECT, INSERT, UPDATE, DELETE, ON recipes.* TO tom@localhost
IDENTIFIED BY 'tommygun';
```

you can accomplish the same task with the following two (equivalent) `INSERT` commands:

```
mysql> INSERT INTO user (Host, User, Password) VALUES('localhost', 'tom',
PASSWORD('tommygun'));
mysql> INSERT INTO db (Host, Db, User, Select_priv, Insert_priv, Update_priv,
Delete_priv, Create_priv, Drop_priv) VALUES ('localhost', 'recipes', 'tom', 'Y',
'Y', 'Y', 'Y', 'N', 'N');
mysql> FLUSH PRIVILEGES;
```

Similarly, while you can set up an administrative user with the following `GRANT`,

```
mysql> GRANT ALL PRIVILEGES ON *.* TO admin@localhost IDENTIFIED BY
'master';
```

you can also do it with the following `INSERT`:

```
mysql> INSERT INTO user (Host, User, Password, Select_priv, Insert_priv,
Update_priv, Delete_priv, Create_priv, Drop_priv, Reload_priv, Shutdown_priv,
Process_priv, File_priv, Grant_priv, References_priv, Index_priv, Alter_priv)
VALUES ('localhost', 'admin', PASSWORD('master'), 'Y', 'Y', 'Y', 'Y', 'Y', 'Y',
'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'Y');
mysql> FLUSH PRIVILEGES;
```

## Viewing Privileges

MySQL enables you to view the privileges assigned to a particular user with the `SHOW GRANTS` command, which accepts a username as argument and displays a list of all the privileges granted to that user. The following examples illustrate this:

```
mysql> SHOW GRANTS FOR sarah@localhost;
+-----------------------------------------+
| Grants for sarah@localhost              |
+-----------------------------------------+
| GRANT USAGE ON *.* TO 'sarah'@'localhost'  |
|  IDENTIFIED BY PASSWORD '4837a3954ee01ece' |
| GRANT SELECT ON sales.customers TO         |
| 'sarah'@'localhost'                        |
+-----------------------------------------+
2 rows in set (0.00 sec)
mysql> SHOW GRANTS FOR root;
+-------------------------------------------------------------+
| Grants for root@%                                           |
+-------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'%' WITH GRANT OPTION |
+-------------------------------------------------------------+
1 row in set (0.06 sec)
```

***TIP*** *The UNIX distribution of MySQL includes a script named `mysqlaccess`, which can be used to generate reports on a particular user's access level and privileges. Type* ***`mysqlaccess --howto`*** *for usage examples.*

## Reloading the Grant Tables

Privileges set using `GRANT` and `REVOKE` are immediately activated (as demonstrated in one of the examples in the preceding section). Privileges set via regular SQL queries, however, require a server reload to come into effect. A server reload can be accomplished via the `FLUSH PRIVILEGES` command, as in the following:

```
mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.05 sec)
```

***NOTE*** *You need the `RELOAD` privilege to run the `FLUSH` command.*

MySQL normally reads the grant tables once when it starts up, but you can also force MySQL to reload the grant tables via the `mysqladmin` tool, as in the following example. This example is for UNIX. Simply replace the path with the correct path to your MySQL installation for Windows:

```
[root@host]# /usr/local/mysql/bin/mysqladmin -u root reload
```

### Resetting the Grant Tables

If you want to reset the grant tables to their initial default settings, the process is as follows:

1. If the server is running, stop it in the usual manner:

   ```
   [root@host] $ /usr/local/mysql/support-files/mysql.server stop
   ```

2. Change to the data directory of your MySQL installation, and then delete the `mysql/` folder. Because databases in MySQL are represented as directories on the file system, this will effectively erase the grant tables.

   ```
   [root@host] $ rm -rf /usr/local/mysql/data/mysql
   ```

3. On UNIX, reinstall the grant tables by running the initialization script, `mysql_install_db`, which ships with the program:

   ```
   [root@host]# /usr/local/mysql/scripts/mysql_install_db
   ```

   On Windows, because this initialization script is not part of the binary distribution, you need to reinstall the package into the same directory to revert to the original grant tables.

4. On UNIX, change back to the data directory of your MySQL installation and alter the ownership of the newly created MySQL directory, so it is owned by the `mysql` user:

   ```
   [root@host]# chown -R mysql.mysql /usr/local/mysql/data/mysql
   ```

5. Restart the server.

   ```
   [root@host] $ /usr/local/mysql/support-files/mysql.server stop
   ```

The MySQL grant tables should now be reset to their default values. You can now log in as the superuser and make changes to them using the `GRANT` and `REVOKE` commands.

## Changing User Passwords

When adding a user to the `user` table with the `GRANT` command, MySQL also enables you to specify a password for that user with the optional `IDENTIFIED BY` clause. This password must be provided by the user when logging in to the server to gain access to its databases. When using the `mysql` client program, the username and password can be specified on the command line via the `-u` and `-p` arguments, respectively, as in the following example:

```
[user@host] $ mysql -h localhost -u logger -p
Enter password: ******
Welcome to the MySQL monitor ...
```

If you don't want to be prompted for a password (for example, if you're using the client noninteractively through a script), you can specify it on the command line after the `-p` argument, as in the following:

```
[user@host] $ mysql -h localhost -u logger -ptimber
Welcome to the MySQL monitor ...
```

Passwords are stored in the `Password` field of the `user` table in the `mysql` database and must be encrypted prior to insertion with the MySQL `PASSWORD()` function, as in the following examples:

```
mysql> INSERT INTO user (Host, User, Password) VALUES('melonfire.net',
'timothy', PASSWORD('1r0ck'));
mysql> UPDATE user SET Password = PASSWORD('1r0ck') WHERE Host =
'melonfire.net' AND User = 'timothy';
mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.05 sec)
```

Passwords can also be set with the MySQL `SET PASSWORD` command. The following example is equivalent to the previous one.

```
mysql> SET PASSWORD FOR timothy@melonfire.net = PASSWORD('1rock');
```

When setting a password using the `IDENTIFIED BY` clause of the `GRANT` command or via the `mysqladmin` tool, you do not need to encrypt the password string first with the `PASSWORD()` function. The `GRANT` command and the `mysqladmin` utility automatically take care of this for you. Therefore, the following example is equivalent to the previous one:

```
mysql> GRANT USAGE ON *.* TO timothy@melonfire.net IDENTIFIED BY '1rock';
```

The first method requires `FLUSH PRIVILEGES` to work and generally is not recommended. The second method is recommended.

The `IDENTIFIED BY` clause of the `GRANT` command is optional. Creating a `GRANT` for a new user without using this clause will set an empty password for that user. This opens a security hole in the system, so administrators should always make it a point to assign a password to new users.

**Safety First!**

The MySQL manual recommends against placing your password after the `-p` option on the `mysql` command line and, instead, suggests entering it at the interactive password prompt, which masks your password as you type it. This will reduce the risk of someone viewing your password over your shoulder and, thereby, gaining unauthorized access to your account on the MySQL server.

**Out with the Old...**
The `PASSWORD()` function in MySQL 4.1 and better generate a longer, 41-byte hash value that is not compatible with older versions (which used a 16-byte value). Therefore, when you upgrade a MySQL server installation older than 4.1 to MySQL 4.1 or better, you must run the `mysql_fix_privilege_tables` script in the `scripts/` directory of your MySQL installation to update the grant tables, so they can handle the longer hash value.

When a user logs in to the MySQL server and provides a password string, MySQL first encrypts the provided password string using the `PASSWORD()` function, and then compares the resulting value with the value in the `Password` field of the corresponding user record in the `user` table. If the two values match (and other access rules permit it), the user is granted access. If the values do not match, access is denied.

## Setting the `root` Password

When MySQL is first installed, access to the database server is restricted to the MySQL administrator, aka `root`. By default, this user is initialized with a null password, which is generally considered a Bad Thing. You should, therefore, rectify this as soon as possible by setting a password for this user via the `mysqladmin` tool and using the following syntax. (This example is for UNIX. Simply replace the path with the correct path to your MySQL installation for Windows.)

```
[root@host]# /usr/local/mysql/bin/mysqladmin -u root password 'new-password'
```

This password change goes into effect immediately, with no requirement to restart the server or flush the `privilege` table.

The password can also be changed using either the `UPDATE` command or the `SET PASSWORD` command, as described in the preceding section.

## Resetting the `root` Password

If you forget the MySQL `root` password and are locked out of the grant tables, take a deep breath, and then follow these steps to get things up and running again:

1. Log in to the system as the system administrator (`root` on UNIX) and stop the MySQL server. This can be accomplished via the `mysql.server` startup and shutdown script in the `support-files/` directory of your MySQL installation, as follows:

   ```
   [root@host] $ /usr/local/mysql/support-files/mysql.server stop
   ```

   In case you're working on a UNIX system that came with MySQL preinstalled or installed MySQL from a binary RPM, you can also stop (and start) MySQL with the `/etc/rc.d/init.d/mysqld` scripts.

2. Next, start up MySQL again with the special `--skip-grant-tables` startup option.

```
[root@host] $ /usr/local/mysql/bin/safe_mysqld
--skip-grant-tables --skip-networking
```

   This bypasses the grant tables, enabling you to log in to the server as the MySQL `root` user without providing a password. The additional `--skip-networking` option tells MySQL not to listen for TCP/IP connections and ensures that no one can break in over the network while you are fixing the privileges.

3. Use the `UPDATE command`, as described in the preceding section, to set a new password for the MySQL `root` user.

```
[root@host]# mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 4.0.14
Type 'help;' or '\h' for help. Type '\c' to clear the buffer
mysql> USE mysql;
Database changed
mysql> UPDATE user SET Password = PASSWORD('new-password') WHERE User = 'root';
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

4. Log out of the server, stop it, and restart it again in the normal manner.

```
[root@host] $ /usr/local/mysql/support-files/mysql.server stop
[root@host] $ /usr/local/mysql/support-files/mysql.server start
```

MySQL should now enable you to log in as the `root` user with the new password set in step 3.

---

***NOTE*** *Read more about MySQL security in the online MySQL manual, at* `http://www.mysql.com/doc/en/General_security.html` *and* `http://www.mysql.com/doc/en/Privileges_provided.html`.

---

## Summary

MySQL comes with a five-tiered access control system, giving you fine-grained control over the privileges each user possesses in relation to databases, tables, and individual fields. This chapter discussed the access control and privilege system in detail, examining the MySQL grant tables, and explaining the `GRANT` and `REVOKE` commands used to manage user privileges. It also examined the topics of limiting resource usage on the server on a per-user basis, changing user passwords, recovering from a lost superuser password, and resetting the grant tables, as well as highlighted some of the new security features available in MySQL 4.*x*.