Getting Started with MySQL

This article is the first in an educational series offered by MySQL AB aimed towards providing the reader with valuable insight into the MySQL database server. Although future articles will delve into some of the more complicated topics surrounding MySQL, including replication, ODBC and optimization, it was thought to be prudent if the first tutorial started, well, at the beginning. Therefore the goal of this article is to thoroughly acquaint the reader with various topics surrounding the basic functioning of MySQL. A synopsis of what is covered is shown in the Table of Contents, listed below. You can go to any topic listed in the Table of Contents simply by clicking on its title.

Assumptions

At this point, it is assumed that the reader has successfully installed the MySQL database server. If MySQL has not yet been installed, please take some time to review the information provided in the installation section of the MySQL documentation. It is also assumed that mysql database has been created (using mysql_install_db), and the MySQL database server has been started using safe_mysqld. If this has not yet been accomplished, take a moment to read "Post-Installation Setup and Testing", located in the MySQL documentation.

It is also assumed that the reader has at least a basic comprehension of SQL (Structured Query Language) syntax. For those readers new to the world of SQL, the following links point to a few particularly useful SQL tutorials:

- Philip Greenspun's "SQL for Web Nerds"
- Mike Chapple's Introduction to SQL
- James Hoffman's Introduction to SQL

Table of Contents

So You've Installed MySQL. Now What?

The installation instructions were scrutinized, the latest distribution was downloaded, coffee was brewed and drank and brewed again. The familiar *configure*, *make* and *make install* were wielded to once again produce another beautifully compiled application. Nods were exchanged, pats on the back traded, frothy capuccino toasts are proposed in succession. Yes, there is reason to celebrate in the office today, as the MySQL database server has been successfully installed.

You lounge back in your deskchair, surrounded by colleagues hailing the wisdom of you, the newly-christened MySQL administrator. If they only knew the pain and anguish swirling around in your mind right now, as you ponder the question, "So now what?".

The purpose of this tutorial is to acquaint new MySQL users with several of the key aspects of this wonderful database server. Issues regarding general server functionality, security, user and privilege administration, working with databases and tables, and data backups will all be introduced to some degree. While the reader will likely find much of this material easy to understand, keep in mind that these concepts lay much of the foundation for efficiently and properly working with the MySQL database server, in addition to implementing more complicated aspects which will be discussed in later tutorials. Therefore it is suggested that the reader take the time to not only read the tutorial, but also to actually follow along with the steps described herein, experimenting with his own MySQL installation.

The MySQL Configuration File: my.cnf

It's very likely that the first task the administrator will want to undertake is proper configuration of MySQL's configuration file. This file, entitled *my.cnf*, stores default startup options for both the server and for clients. Correct configuration of this file can go a long way towards optimizing MySQL, as various memory buffer settings and other valuable options can be set here.

Interestingly, the scope of this file can be set according to its location. The settings will be considered global to all MySQL servers if stored in */etc/my.cnf*. It will be global to a specific server if located in the directory where the MySQL databases are stored (*/usr/local/mysql/data* for a binary installation, or */usr/local/var* for a source installation). Finally, its scope could be limited to a specific user if located in the home directory of the MySQL user (*~/.my.cnf*). Keep in mind that even if MySQL does locate a my.cnf file in /etc/my.cnf (global to all MySQL servers on that machine), it will *continue* its search for a server-specific file, and then a user-specific file. You can think of the final configuration settings as being the result of the /etc/my.cnf, mysql-data-dir/my.cnf, and ~/.my.cnf files.

In order to aid administrator's in the proper configuration of this file, the MySQL developers have included four sample *my.cnf* files within the distribution. Their names are my-huge.cnf.sh, my-large.cnf.sh, my-medium.cnf.sh, and my-small.cnf.sh, and each denotes recommended configuration settings in accordance with system resource availability.

Further Reading

- Option Files

The MySQL Privilege Tables

Before delving into the many examples that consititute this tutorial, a brief introduction of one of the most important (and most misunderstood!) aspects of the MySQL server is in order; that is the mechanism from which MySQL secures its data and integrity: The MySQL privilege tables. The MySQL privilege tables are responsible for authenticating user access to the MySQL server, and subsequently associating those users granted access with a set of privileges. This privilege set decides what a user is capable of doing while connected to the MySQL server, controlling the user's activities on a server-wide, database, tabular and even columnar level. For example, an administrator could grant a user only enough privileges to connect to one specific MySQL database, and restrict access to all others. Furthermore, that same user might be granted only certain privileges while connected to that database, selection, insertion, and modification privileges for example. Associated with only these three privileges, that user would be denied any attempt to delete data, since the user has not been granted the deletion privilege.

Although in-depth introduction to the privilege tables is out of the scope of this article, it is important that the reader understands the very important role these tables play in securing the MySQL server. While working through the examples presented throughout the remainder of this article, keep in mind that the privilege tables are playing a role in *every single* query and command that takes place, ensuring that the user executing these commands/queries has proper permissions for doing so. It is also strongly recommended that the reader takes some time to review the links listed below, as each describes in great detail the underlying mechanics of these tables.

Further Reading

- How the Privilege System Works
- How to Make MySQL Secure Against Crackers
- Adding New User Privileges to MySQL

Connecting to the MySQL Server For the First Time

The MySQL client program, also known as the MySQL monitor, is an interface that allows the user to connect to the MySQL server, create and modify databases, and execute queries and view their results. This program is started by executing the command mysql at the shell prompt. In general, the syntax for this command is:

```
%>mysql [options] [database]
```

Where [options] can be one or a series of options used in conjunction with the mysql program, and [database] is the name of the database to use. Since it is assumed to be the reader's first time using the MySQL monitor, take a moment to review all offered options by executing the following command:

```
%>mysql --help
```

This produces a long list of options that can be used in conjunction with the mysql program. For the moment, however, the main goal is to simply connect to the database server. Therefore, execute the following command:

```
%>mysql -u root
```

The following should appear:

```
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8 to server version: 3.23.28-gamma-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer

mysql>
```

Congratulations, you are now connected to the MySQL monitor as the almighty root user. Your first official action as this supreme leader of the MySQL database server should be to ensure that nobody else can declare this position. Therefore, make it possible to only connect as root in the future by supplying a password. Change the password from its current blank (or null) value, to something difficult to guess using the following command:

```
mysql>SET PASSWORD FOR 'root'@'localhost' = PASSWORD('secret_password');
```

The 'root', which is the username, and 'localhost', which is the hostname, constitute a unique user in MySQL. For those readers perhaps unfamiliar with networking terminology, 'localhost' is a name used to specify the local server; that is, the server upon which MySQL resides. Therefore, by stating 'root'@'localhost', this command is telling the MySQL server to set the password for a user named 'root' that will connect specifically from the local server (thus 'localhost'). More specifically, this command will change the password by updating what are commonly known as the MySQL privilege tables. These tables, collectively located in the mysql database, contain information regarding the connection and usage capabilities of all users intended to use the MySQL database server. More specifically, this command will update the user table, updating the password field of the row in which the user field's value is 'root'. The password field will be updated with the encrypted value of the string enclosed within the Password() function.

Of course, do not forget this password. Since it is stored in encrypted text on the database server, it cannot simply be looked up if forgotten.

There is also an alternative method for updating a password:

```
%>mysqladmin -u root  password 'secret_password'
```

This command will accomplish the same results as the one previously introduced.

Exiting and Reconnecting to the MySQL Monitor

In order to test the new password, exit the MySQL database using the following command:

```
mysql>\q
```

This will return you to the system shell. Now log back into the monitor, this time using the following command:

```
%>mysql -u root -p
```

Doing so will result in a prompt for the root user password, as follows:

```
Enter password:
```

Go ahead and enter the password supplied within the update command that was used to set the root password. Assuming it is entered correctly, the standard MySQL greeting will appear, and root will be connected to the MySQL server once again.

Careful With That Password!

Many readers may be tempted to instead try to include the password on the same line as the mysql connection command, as follows:

```
%>mysql -u root -psecret_password
```

Do not do this! Not only is it a highly insecure method for entering the password, but it will not produce the expected results! It is insecure not only because it will allow any onlookers the possibility of seeing the password in its plaintext format, but also because any user can use the Unix 'ps' command to look at what commands you are executing and view the password in its plaintext format from there.

It might be a good idea to store the password in your my.cnf configuration file, located in ~/.my.cnf. If you don't know what this file is, please read the earlier section entitledThe MySQL Configuration File: my.cnf.

Selecting a Database

Of course, simply connecting to the MySQL server isn't going to accomplish much. Chances are you will want to select a database to work with. This is accomplished in one of two ways:

1. Including the name of the database along with the mysql connection command. For example, the command used to both connect to the MySQL server and select the mysql database is:

   ```
   %>mysql -u root -p mysql
   ```

   This might be misleading for some readers, as it seems as if the intent is to input mysql as the password. This is not correct. Take a moment to review the syntax as described in the mysql --help output, and it will be apparent that -u root -p actually comprise the [options] component of the syntax, and mysql comprises the [database] component.

2. Once connected, select the database using the use command, as follows:

   ```
   mysql>use mydatabase
   ```

Once executed, all queries not explicitly specifying a database name will be directed towards the hypothetical *mydatabase* database.

mysqladmin

The *mysqladmin* program is used to administrate various aspects of the MySQL database server. Using it, the administrator can perform tasks such as: create and delete databases, shutdown the database server, update the privilege tables, and view running MySQL processes. The

general syntax is:

```
%>mysqladmin [options] command(s)
```

Where [options] can be one or a series of options used in conjunction with the *mysqladmin* program, and [database] is the name of the database to use. Since it is assumed to be the reader's first time using the MySQL monitor, take a moment to review all offered options by executing the following command:

```
%>mysqladmin --help
```

This produces a long list of options that can be used in conjunction with the *mysqladmin* program. As a demonstration of how these options are used, let's use *mysqladmin* to create a new database named widgets, which will be used throughout the remainder of this article to demonstrate various other useful MySQL functions. A new database is created as follows:

```
%>mysqladmin -u root -p create widgets
Enter Password:
```

Upon execution, *mysqladmin* will create the database and return to the shell prompt. Typically, the next step is to secure the new database by modifying the privilege tables. Details regarding how this is accomplished is the subject of the next section.

Further Reading

- Administering a MySQL Server

Securing a Database

Security should be the first thought that comes to a MySQL administrator's mind after creating a new database. Privilege as was discussed in the section, "The Privilege Tables", securing a database is accomplished through modifications made to the tables found in the mysql database. In this section, the reader will learn how to secure the newly created widgets database. Before doing so, a brief summary of exactly how the privilege tables are modified is in order.

There are two methods used to modify the privilege tables. The first is through the use of typical SQL statements such as INSERT, UPDATE and DELETE. However, this method has largely been deprecated through the introduction of the second method, which involves the use of the special commands GRANT and REVOKE. Therefore, only this second method will be discussed in this section.

The Grant Command

The GRANT function is used both to create new users, and to assign privileges to users. Its syntax is:

```
mysql>GRANT priv_type [(column_list)] [, priv_type [(column_list)] ...]
      ON {tbl_name | * | *.* | db_name.*}
      TO user_name [IDENTIFIED BY 'password']
      [, user_name [IDENTIFIED BY 'password'] ...]
      [WITH GRANT OPTION]
```

An understanding of how GRANT works is best gained through examples. In the first example, GRANT is used to add a new user to the database. This user will be used to access the widgets database:

```
mysql>GRANT usage ON *.* TO widgetAdmin@localhost
    ->IDENTIFIED BY 'ilovewidgets';
```

This will create a new user named *widgetAdmin*, capable of connecting to the MySQL database server via the host *localhost* using the password *ilovewidgets*. Keep in mind that this only grants connection privileges. It will not allow the user to do anything with the MySQL server! Go ahead and switch to the *mysql* database and execute the following query:

```
mysql>SELECT * FROM user;
```

Notice that the row containing the *widgetAdmin* user has N values for all of the privileges. This is good, since the user table contains a user's global privilege settings. To clarify this, if a *Y* value is set for any user's privilege in the *user* table, that user can apply that privilege to *any* MySQL database. Therefore, it is almost always best to set all privileges to *N* within this table.

So how then are user privileges assigned for a particular database? This is easily done just like the usage privilege was set in the previous example. For example, assume that the administrator wanted to grant user *widgetAdmin* with SELECT, INSERT, UPDATE and DELETE privileges for the *widget* database. This is accomplished using the following GRANT command:

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

Upon execution, the user *widgetAdmin* can immediately begin using these privileges.

The privileges introduced thus far are not the only ones available to the administrator. Table 1-1 provides a listing of all available privileges.

Table 1-1: Privileges available for use within GRANT and REVOKE commands

ALL PRIVILEGES	FILE	RELOAD
ALTER	INDEX	SELECT
CREATE	INSERT	SHUTDOWN
DELETE	PROCESS	UPDATE
DROP	REFERENCES	USAGE

To view the new privilege table updates, execute the following query:

```
mysql>SELECT * FROM db;
```

Notice that a row has been added to the *db* table for user *widgetAdmin*, with Y values assigned to the SELECT, INSERT, UPDATE and DELETE fields.

Incidentally, it is also possible to bypass the usage query, instead both creating the new user and assigning user privileges simply by executing a variation of the previous query:

```
mysql>GRANT SELECT, INSERT, UPDATE, DELETE
    ->ON widgets.* TO widgetAdmin@localhost
    ->IDENTIFIED BY 'ilovewidgets';
```

Assuming that the user widgetAdmin did not yet exist when this query is executed, both the *user* and *db* tables will be updated with the necessary rows.

Of course, the administrator can revoke previously granted privileges. Exactly how this is accomplished in discussed in the following section.

The Revoke Command

The REVOKE command is used to rescind privileges previously granted to a user. Its syntax is:

```
REVOKE priv_type [(column_list)] [, priv_type [(column_list)] ...]
       ON {tbl_name | * | *.* | db_name.*}
       FROM user_name [, user_name ...]
```

As is the case with the GRANT command, perhaps the best way to really understand how it operates is to experiment with several examples. Assume that the administrator wants to repeal the DELETE privilege from the user *widgetAdmin*. This is accomplished using the following command:

```
mysql>REVOKE DELETE ON widgets.*
    ->FROM widgetAdmin@localhost;
```

Refer to Table 1-1 for a complete listing of privilege types which can be used within the REVOKE command.

One point to keep in mind is that while REVOKE can remove all privileges (including connection privileges) from a user, it does *not* explicitly remove that user from the privilege tables. To illustrate this, consider the following command:

```
mysql>REVOKE ALL PRIVILEGES ON widgets.*
    ->FROM widgetAdmin@localhost;
```

While this would result in all privileges being revoked from the user *widgetAdmin*, it would not delete the relevant rows from the privilege tables! If completely removing the user from the database is the intention, the rows would have to be removed using the delete command, as follows:

```
mysql>DELETE FROM user WHERE user = 'widgetAdmin';
Query OK, 1 row affected (0.00 sec)
mysql>flush privileges;
```

This will effectively deny that user from connecting to the MySQL server. Note that rows from the *user* table will have to be explicitly removed using DELETE should the administrator wish to entirely remove the user from the privilege tables.

Further Reading

- Grant and Revoke Syntax

Database Backups

The final concept discussed in this tutorial is indeed an important one: data backups. In this section, two methods for making backups of MySQL data and database structures are discussed, namely *mysqldump* and *mysqlhotcopy*.

mysqldump

The utility *mysqldump* provides a rather convenient way to dump existing data and table structures. Note that while *mysqldump* is not the most efficient method for creating backups (mysqlhotcopy is, described next), it does offer a convenient method for copying data and table structures

which could then be used to repopulate another SQL server, that server not even necessarily being MySQL.

The function *mysqldump* can be used to backup all databases, several databases, one database, or just certain tables within a given database. In this section, the syntax involved with each scenario is provided, followed with a few examples.

Using *mysqldump* to backup just one database:

```
%>mysqldump [options] db_name
```

Using *mysqldump* to backup several tables within a database:

```
%>mysqldump [options] db_name table1 table2 . . . tableN
```

Using *mysqldump* to backup several databases:

```
%>mysqldump [options] --databases [options] db_name1 db_name2 . . . db_nameN
```

Using *mysqldump* to backup all databases:

```
%>mysqldump [options] --all-databases [options]
```

The options can be viewed by executing the following command:

```
%>mysqldump --help
```

Examples:

Backing up both the structure and data found within the widgets database would be accomplished as follows:

```
%>mysqldump -u root -p --opt widgets
```

Alternatively, perhaps just a backup of the data is required. This is accomplished by including the option *--no-create-info*, which means no table creation data:

```
%>mysqldump -u root -p --no-create-info widgets
```

Another variation is just to backup the table structure. This is accompished by including the option *--no-data*, which means no table data:

```
%>mysqldump -u root -p --no-data widgets
```

If you are planning on using *mysqldump* for reason of backing up data so it can be moved to another MySQL server, it is recommended that you use the option '--opt'. This will give you an optimized dump which will result in a faster read time when you later load it to another MySQL server.

While *mysqldump* provides a convenient method for backing up data, there is a second method which is both faster and more efficient. It is described in the next section.

mysqlhotcopy

The *mysqlhotcopy* utility is a perl script that uses several basic system and SQL commands to backup a database. More specifically, it will lock the tables, flush the tables, make a copy, and unlock the tables. Although it is the fastest method available for backing up a MySQL database, it is limited to backing up only those databases residing on the same machine as where it is executed.

The function *mysqlhotcopy* can be executed to backup one database, a number of databases, or only those databases matching a name specified by a regular expression. In this section, the syntax involved with each scenario is provided, followed with a few examples.

Using *mysqlhotcopy* to backup just one database:

```
%>mysqlhotcopy [options] db_name /path/to/new_directory
```

Using *mysqlhotcopy* to backup just several databases:

```
%>mysqlhotcopy [options] db_name_1 ... db_name_n /path/to/new_directory
```

Using *mysqlhotcopy* to backup only those tables within a given database that match a regular expression:

```
%>mysqlhotcopy [options] db_name./regex/
```

The options can be viewed by executing the following command:

```
%>mysqlhotcopy --help
```

Examples:
Experiment with *mysqlhotcopy* by backing up the widgets database to the directory path "/usr/mysql/backups/". Execute the following command:

```
%>mysqlhotcopy -u root -p widgets /usr/mysql/backups
```

As a second example, assume that the widgets database contains the tables: "products2000", "products2001", "clientele2000", and "clientele2001", with the four digits at the end of each name representing the year for which that data represents. The administrator wants to backup only those tables relative to the year "2000":

```
%>mysqlhotcopy -u root -p widgets./^.+('2000')$/ /usr/mysql/backups
```

In the above example, the regular expression /^.+('2000')$/ tells *mysqlhotcopy* to only backup those tables ending with the string "2000".

Further Reading

- Dumping MySQL Database and Table Structures and Data
- Replication in MySQL
- Copying MySQL Databases and Tables

Conclusion

The goal of this tutorial was to introduce the essential topics which surround the basic functionality of the MySQL server. It is suggested that the novice reader devote some time to experimenting with all of the examples, as "learning by doing" is certainly the fastest way to become comfortable with MySQL.

Next time, we'll take a look at some general configuration issues, focusing on the my.cnf files first introduced at the beginning of this tutorial.

W.J. Gilmore is the author of A Programmer's Introduction to PHP 4.0 (January, 2001 Apress). He is the Assistant Editorial Director of Web and Open Source Technologies at Apress, and is a regular contributor to several of the Web's most prominent developer publications.