## Setting up the database

### Creating A Database

When you have logged in and a mysql prompt is displayed, let's then take a look at the databases we currently have. To do so, we use the *SHOW DATABASES* command.

**user $** mysql -uroot -q

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 673

Server version: 5.1.62-log Gentoo Linux mysql-5.1.62-r1

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its

affiliates. Other names may be trademarks of their respective

owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>

**mysql>** SHOW DATABASES;

+----------+

| Database |

+----------+

| mysql |

| test |

+----------+

2 rows in set (0.09 sec)

**Important**

Please remember that MySQL commands should end with a semicolon (;)

Despite the fact that a test database is already created, we are going to create our own. Databases are created using the *CREATE DATABASE* command. We'll create one named "gentoo".

**mysql>** SHOW CREATE DATABASE gentoo;

Query OK, 1 row affected (0.08 sec)

The response lets us know that the command was executed without any errors. In this case, 1 row was modified. This is a reference to the main mysql database, which carries a list of all the databases. We can verify the database was created by running the *SHOW DATABASES* command again.

**mysql>** SHOW DATABASES;

+----------+

| Database |

+----------+

| gentoo |

| mysql |

| test |

+----------+

3 rows in set (0.00 sec)

Indeed our database has been created. In order to work with creating tables for our new gentoo database, we need to select it as our current database. To do so, we use the *USE* command. The *USE* command takes the name of the database you wish to use as your current database.

**mysql>** USE gentoo;

Database changed

And the current database is now our previously created gentoo database. Now that we're using it, we can start to create some tables and put information in them.

## Working With Tables In MySQL

### Creating a Table

In the structure of MySQL, there are databases, tables, records, and fields. Databases hold together tables, tables hold together records, records hold together fields, which contain the actual information. This structure lets users select how they want to access their information. So far we've dealt with databases, now let's work with tables. First off, tables can be listed similiarly to databases using the *SHOW TABLES* command. Right now there are no tables in our gentoo database, as running the command will show us:

**mysql>** SHOW TABLES;

Empty set (0.00 sec)

Use the *CREATE TABLE* command to create some tables. However, this command is quite different from simple '*CREATE DATABASE* command. This command takes a list of arguments. The form is as follows:

**mysql>** CREATE TABLE [table\_name] ([field\_name] [field\_data\_type]([size]);

table\_name is the name of the table we wish to create. In this case, let's make a table named developers. This table will contain the developer's name, email and job. field\_name will contain the name of the field. We have 3 required names in this case: name, email, and job. The field\_data\_type is what type of information will be stored. The different formats available can be found at the MySQL Column Types Page. For our purposes, we'll use the VARCHAR data type for all of our fields. VARCHAR comes to working with strings. size is how much of data a single field will store. In this case, we'll use 128. This means that the field can have VARCHAR data that is 128 bytes.

**mysql>** CREATE TABLE developers ( name VARCHAR(128), email VARCHAR(128), job VARCHAR(128));

Query OK, 0 rows affected (0.11 sec)

Looks like our table was created ok. Let's check it with the SHOW TABLES command:

**mysql>** SHOW TABLES;

+------------------+

| Tables\_in\_gentoo |

+------------------+

| developers |

+------------------+

1 row in set (0.00 sec)

Yes, there's our table. However, it doesn't seem to have any information on the types of fields we setup. For that, we use the DESCRIBE command (or DESC for short), which takes the name of the table as its argument. Let's see what that gives us for our developers table.

**mysql>** DESCRIBE developers;

+-------+--------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-------+--------------+------+-----+---------+-------+

| name | varchar(128) | YES | | NULL | |

| email | varchar(128) | YES | | NULL | |

| job | varchar(128) | YES | | NULL | |

+-------+--------------+------+-----+---------+-------+

3 rows in set (0.00 sec)

This shows the different fields and their types. It also shows a few extra attributes that are beyond the scope of this howto.

### Populating Our MySQL Database

We populate a table (or add data) using the INSERT command. Like the CREATE TABLE command, it also has a specific format:

**mysql>** INSERT INTO table (col1, col2, ...) VALUES('value1', 'value2', ...);

This command is used to insert a record into table. table contains the MySQL table we wish to enter the information into. The table name may be followed by the list of columns to insert data into and VALUES() contains the values you wish to insert into the table. You may omit the list of columns if you insert a value into each one and if you write the values in the same order the columns have been defined. In this case, we want to insert data into the developers table. Let's insert sample records:

**mysql>** INSERT INTO developers VALUES('Joe Smith', 'joesmith@gentoo.org', 'toolchain');

Query OK, 1 row affected (0.06 sec)

If you don't know the order of the columns in the table or want to insert an incomplete record:

**mysql>** INSERT INTO developers (job, name) VALUES('outsourced', 'Jane Doe');

Query OK, 1 row affected (0.01 sec)

According to our return result, it appears that the record was inserted correctly. What if we want to input more information than just one record? That's where the LOAD DATA command comes into play. This loads records from a tab separated file. Let's try that by editing a file in our home directory with the records. We'll call this file records.txt. Here's a sample:

Code

John Doe johndoe@gentoo.org portage

Chris White chriswhite@gentoo.org documentation

Sam Smith samsmith@gentoo.org amd64

**Important**

Be sure you know what data you're dealing with. It's very unsafe to use LOAD DATA when you are uncertain of the file's contents!

Now the LOAD DATA command has a somewhat enlongated definition, but we'll use the simplest form here.

**mysql>** LOAD DATA LOCAL INFILE '/path/to/filename' INTO TABLE table;

/path/to/filename is the directory and filename that will be used. table is the name of our table. In this case, our file is ~/records.txt and the table is developers.

**mysql>** LOAD DATA LOCAL INFILE '~/records.txt' INTO TABLE developers;

Query OK, 3 rows affected (0.00 sec)

Records: 3 Deleted: 0 Skipped: 0 Warnings: 0

**Important**

If you come up with any strange behaviour, make sure your fields are separated by tabs. If you paste information into your infile from another source, it may convert your tabs to spaces.

That worked well. However, this simply inserts records, and does not give you any sort of control over MySQL. Many web applications use sql scripts in order to setup MySQL quickly and easily. If you want to use an sql script, you'll need to run mysql in batch mode, or source the file. Here's an example of running mysql in batch mode:

**user $** mysql -u root -h localhost -p < sqlfile

Like LOAD DATA, be sure you can tell what sqlfile does. Failure to do so may cause your database to be compromised! Another way you can accomplish this is to use the source command. This command will run mysql commands from an sql file while in the mysql interactive mode. Here is how to source an sql file:

**mysql>** source sqlfile;

If you see a web application wanting you to run an sql file, the two above commands can be used to accomplish that task. We have our table setup, so how do we check our fields? We do this by searching our table with queries.

### Browsing MySQL Tables With Queries

Queries are one of the main features of any SQL database. They help us turn data in our tables into something useful. Most queries are done with the SELECT command. The SELECT command is fairly complex, and we're only going to look at three main forms of the command in this document.

(Select all entries in a table)

**mysql>** SELECT \* FROM table;

(Select specific entries in a table)

**mysql>** SELECT \* FROM table WHERE field=value;

(Select specific fields)

**mysql>** SELECT field1,field2,field3 FROM table [WHERE field=value];

Let's take a quick look at the first form. It's relatively simple and gives you an overall look of your table. We'll go ahead and run it to see what data we have so far.

**mysql>** SELECT \* FROM developers;

+-------------+-----------------------+----------------+

| name | email | job |

+-------------+-----------------------+----------------+

| Joe Smith | joesmith@gentoo.org | toolchain |

| John Doe | johndoe@gentoo.org | portage |

| Chris White | chriswhite@gentoo.org | documentation |

| Sam Smith | samsmith@gentoo.org | amd64 |

| Jane Doe | NULL | Outsourced job |

+-------------+-----------------------+----------------+

5 rows in set (0.00 sec)

We see both the data we inserted through INSERT and those inserted by LOAD DATA present. Now, let's say that we just want to see the record for Chris White. We can do so with the second form of select as shown below.

**mysql>** SELECT \* FROM developers WHERE name = 'Chris White';

+-------------+-----------------------+---------------+

| name | email | job |

+-------------+-----------------------+---------------+

| Chris White | chriswhite@gentoo.org | documentation |

+-------------+-----------------------+---------------+

1 row in set (0.08 sec)

As expected, the specific entry that we were looking for has been selected. Now, let's say we only wanted to know the person's job and email address, not their name. We can do so with the third form of SELECT as shown here.

**mysql>** SELECT email,job FROM developers WHERE name = 'Chris White';

+-----------------------+---------------+

| email | job |

+-----------------------+---------------+

| chriswhite@gentoo.org | documentation |

+-----------------------+---------------+

1 row in set (0.04 sec)

This method of selection is a lot easier to manage, expecially with larger amounts of information, as we'll see later on. Right now, being the root mysql user, we have unlimited permissions to do what we wish with the MySQL database. In a server environment, a user with such privileges can be quite problematic. In order to control who does what with the databases, we setup privileges.

## MySQL Privileges

### Granting Privileges with GRANT

Privileges are what kind of access users have to databases, tables, pretty much anything. Right now in the gentoo database, the MySQL root account is the only account that can access it, given its permissions. Now, let's create two somewhat generic users, guest and admin, who will access the gentoo database and work with the information in it. The guest account will be a restricted one. All it will be able to do is get information from the database, and that's it. admin will have the same control as root, but only for the gentoo database (not the main mysql databases). Before we start on that, let's have a closer look at this somewhat simplified format of the GRANT command.

**mysql>** GRANT [privileges] ON database.\* TO '[user]'@'[host]' IDENTIFIED BY '[password]';

**Note**

GRANT is considered to be the way to create a user. Later versions of MySQL, however, do contain a CREATE\_USER function, though GRANT is still preferred.

First we have the privileges we wish to assign. With what we've learned so far, here are some of the privileges you can set:

[[Collapse](http://wiki.gentoo.org/wiki/MySQL/First_Steps)]

Code

ALL - Gives the all privilege control for the database

CREATE - Allows users to create tables

SELECT - Allows users to query tables

INSERT - Allows users to insert data into a table

SHOW DATABASES - Allows users to see a list of databases

USAGE - User has no privileges

GRANT OPTION - Allows users to grant privileges

**Note**

If you're running MySQL to communicate data to a web application, CREATE, SELECT, INSERT (discussed here), DELETE and UPDATE (for further infomation look up the MySQL Reference Manual - GRANT and REVOKE Syntax section) are the only permissions you will most likely need. A lot of people make the mistake of granting all permissions when it's not really necessary. Check with the application developers to see if such permissions will cause issues with general operation.

For our admin user, ALL will do. For the guest user, SELECT will be sufficient for read only access. database is the database we wish the user to have these permissions on. In this example, gentoo is the database. The .\* means all tables. If you wanted to, you could apply per table access. user is the name of the user and host is the hostname the user will be accessing from. In most cases, this will be localhost. Finally, password is the user's password. Given the information, let's go ahead and create our users.

(admin)

**mysql>** mysql> GRANT ALL ON gentoo.\* TO 'admin'@'localhost' IDENTIFIED BY 'password';

(guest)

**mysql>** GRANT SELECT ON gentoo.\* TO 'guest'@'localhost' IDENTIFIED BY 'password';

Now that we have the users setup, let's test them out. First we quit mysql by typing quit at the command prompt:

**mysql>** quit

We're now back at the console. Now that we have our users setup, let's go ahead and see what they can do.

### Testing User Permissions

We shall now attempt to login as the guest user. Currently, the guest user has SELECT only privileges. This basically comes down to being able to search and nothing more. Go ahead and login with the guest account.

**user $** mysql -u guest -h localhost -p

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 6 to server version: 4.0.25

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

mysql>

Now we should test the user restriction(s). Let's switch to the gentoo database:

**mysql>** USE gentoo;

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

Now let's try to do something we are not supposed to. We'll attempt to create a table.

**mysql>** CREATE TABLE test (test VARCHAR(20), foobar VARCHAR(2));

ERROR 1044: Access denied for user: 'guest@localhost' to database 'gentoo'

As you can see, this function fails, as our user does not have the appropriate access. However, one access we did grant is the SELECT statement. Let's go ahead and try that:

**mysql>** SELECT \* FROM developers;

+-------------+-----------------------+----------------+

| name | email | job |

+-------------+-----------------------+----------------+

| Joe Smith | joesmith@gentoo.org | toolchain |

| John Doe | johndoe@gentoo.org | portage |

| Chris White | chriswhite@gentoo.org | documentation |

| Sam Smith | samsmith@gentoo.org | amd64 |

| Jane Doe | NULL | Outsourced job |

+-------------+-----------------------+----------------+

5 rows in set (0.00 sec)

The command succeeds, and we're given a glimpse of what user permissions can do. We did, however, create an admin account as well. This was created to show that even all permissions granted users can still have limitations. Go ahead and quit MySQL and login as the admin.

**mysql>** quit

**user $** mysql -u admin -h localhost -p

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 7 to server version: 4.0.25

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

mysql>

To begin, we'll try creating a new database with our admin user. This admin user will have access similiar to the root MySQL account, and will be able to do any kind of modification to the gentoo database it chooses. This will test the user's access to the main MySQL database. Remember ealier that we only set permissions to a specific database.

**mysql>** CREATE DATABASE gentoo2;

ERROR 1044: Access denied for user: 'admin@localhost' to database 'gentoo2'

Indeed, the admin user cannot create databases on the main MySQL database, despite all his permissions on the gentoo database. However, we're still able to use the admin account to modify the gentoo database, as shown here by this example data insertion.

**mysql>** USE gentoo;

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

**mysql>** INSERT INTO developers VALUES('Bob Simmons', 'bobsimmons@gentoo.org', 'python');

Query OK, 1 row affected (0.08 sec)

The admin user can access the database as they please. Now sometimes, we need to get rid of user permissions. This could be anything from a problematic user to a retired employee. Let's take a look at how to disable user permissions with the REVOKE command.

### Removing User Access With The REVOKE Command

The REVOKE command lets us deny access to a user. We can either deny full access, or specific access. In fact, the format is very similiar to GRANT.

**mysql>** REVOKE [privileges] ON database.\* FROM '[user]'@'[host]';

Options here are explained in the GRANT command section. In this section however, we're going to deny full access to a user. Let's say we find out the guest account is causing some problems security wise. We decide to revoke all privileges. We login as root and do the needful.

**mysql>** REVOKE ALL ON gentoo.\* FROM 'guest'@'localhost';

Query OK, 0 rows affected (0.00 sec)

**Note**

In this case, user access is simple, so per database revoking is not a problem. However, in larger cases, you would most likely be using \*.\* instead of gentoo.\* to remove user access to all other databases

Now let's quit and attempt to login as a guest user.

**user $** mysql -u guest -h localhost -p

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 9 to server version: 4.0.25

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

mysql>

Although we're able to login, our access to gentoo is now gone.

**mysql>** USE gentoo;

ERROR 1044: Access denied for user: 'guest@localhost' to database 'gentoo'

And our problematic user is no longer able to access the gentoo database. Please note that the user was still able to login. That is because they remain in the main MySQL database. Let's take a look at how to completely remove an account with DELETE and the MySQL user table.

### Removing Accounts Using DELETE

The MySQL user table is a listing of all users and information about them. Make sure you're logged in as root. Then go ahead and use the main MySQL database.

**mysql>** USE mysql;

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

Now let's see what tables the mysql database has:

**mysql>** SHOW TABLES;

+-----------------+

| Tables\_in\_mysql |

+-----------------+

| columns\_priv |

| db |

| func |

| host |

| tables\_priv |

| user |

+-----------------+

6 rows in set (0.00 sec)

The user table is the table we're after. However, the user table contains 30 different fields, making it very hard to read. In order to make things easier to read, we'll go ahead and use the third version of the SELECT statement. The fields we're after are Host and User.

**mysql>** SELECT Host,User FROM user WHERE User = 'guest';

+-----------+-------+

| Host | User |

+-----------+-------+

| localhost | guest |

+-----------+-------+

1 row in set (0.00 sec)

Now that we have our information, we can get rid of the guest user. This is done with the DELETE command and the syntax is shown below.

**mysql>** DELETE FROM table WHERE field='value';

You may notice that DELETE is somewhat similiar to the SELECT statement in its format. In this case, the field will be User, and the value guest. This will delete the record in the user table where the user is guest, successfully deleting our guest user account. Let's go ahead and do that:

**mysql>** DELETE FROM user WHERE User='guest';

Query OK, 1 row affected (0.07 sec)

(FLUSH PRIVILEGES is needed to update user permissions)

**mysql>** FLUSH PRIVILEGES;

Query OK, 0 rows affected (0.00 sec)

It seems to have worked ok. Let's test by logging out and attempting to login as our guest user.

**mysql>** quit

**user $** mysql -u guest -h localhost -p

Enter password:

ERROR 1045: Access denied for user: 'guest@localhost' (Using password: YES)

Our user is now successfully deleted