Working with Database and Tables (MySQL)

Working with Databases

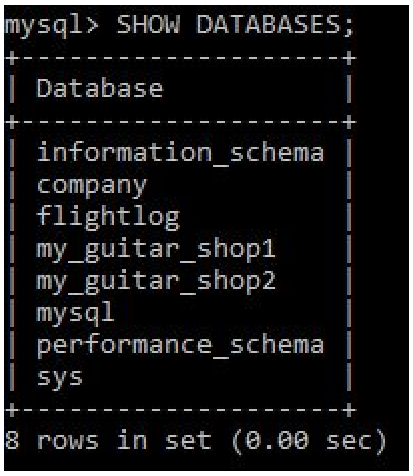
Learning how to manage and navigate MySQL databases and tables is one of the first tasks you'll want to master. This section highlights several key tasks and demonstrates how to view, create, select, and delete MySQL databases.

Viewing Databases

It's often useful to retrieve a list of databases located on the server. To do so, execute the SHOW DATABASES command:

```
mysql>SHOW DATABASES;
```

Note: *the screenshot below is based on the number of databases I have on my home computer, this view may differ from what you get at the time the above command is run.*



```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| company            |
| flightlog          |
| my_guitar_shop1    |
| my_guitar_shop2    |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
8 rows in set (0.00 sec)
```

Keep in mind that your ability to view all the available databases on a given server is affected by user privileges.

Note that using the SHOW DATABASES command is the standard methodology prior to MySQL version 5.0.0, although the command is still available for versions 5.0.0 and greater.

Creating a Database

The easiest way is to create a database by using the CREATE DATABASE command from within the mysql client:

```
mysql>CREATE DATABASE company;

Query OK, 1 row affected (0.00 sec)
```

Common problems for failed database creation include insufficient or incorrect permissions, or an attempt to create a database that already exists.

Using a Database

Once the database has been created, you can designate it as the default working database by "using" it, done with the USE command:

```
mysql>USE company;

Database changed
```

Deleting a Database

You delete a database in much the same fashion as you create one. You can delete it from within the mysql client with the DROP command, like so:

```
mysql>DROP DATABASE company;

Query OK, 1 row affected (0.00 sec)
```

Creating a Table

A table is created using the CREATE TABLE statement. Although there are a vast number of options and clauses specific to this statement, it seems a bit impractical to discuss them all in what is an otherwise informal introduction. Instead, this section covers various features of this statement as they become relevant in future sections. Nonetheless, general usage will be demonstrated here. As an example, the following creates the employees table:

```
CREATE TABLE employees (
employee_id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
last_name VARCHAR(30) NOT NULL,
first_name VARCHAR(30) NOT NULL,
email VARCHAR(100) NOT NULL,
hire_date DATE NOT NULL,
notes MEDIUMTEXT
PRIMARY KEY (employee_id),
INDEX (last_name),
UNIQUE (email) );
```

Keep in mind that a table must consist of at least one column. Also, you can always go back and alter a table structure after it has been created. Later in this section, you'll learn how this is accomplished via the ALTER TABLE statement.

You can also create a table regardless of whether you're currently using the target database. Simply prepend the table name with the target database name like so:

```
database_name.table_name
```

Conditionally Creating a Table

By default, MySQL generates an error if you attempt to create a table that already exists. To avoid this error, the CREATE TABLE statement offers a clause that can be included if you want to simply abort the table-creation attempt if the target table already exists. For example, suppose you want to distribute an application that relies on a MySQL database for storing data. Because some users will download the latest version as a matter of course for upgrading and others will download it for the first time, your installation script requires an easy means for creating the new users' tables while not causing undue display of errors during the upgrade process. This is done via the IF NOT EXISTS clause. So, if you want to create the employees table only if it doesn't already exist, do the following:

```
CREATE TABLE IF NOT EXISTS employees (
employee_id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
last_name VARCHAR(30) NOT NULL,
first_name VARCHAR(30) NOT NULL,
email VARCHAR(100) NOT NULL,
hire_date DATE NOT NULL,
notes MEDIUMTEXT
PRIMARY KEY (employee_id),
INDEX (last_name),
UNIQUE (email) );
```

One oddity of this action is that the output does not specify whether the table was created. Both variations display the "Query OK" message before returning to the command prompt.

Copying a Table

It's a trivial task to create a new table based on an existing one. The following query produces an exact copy of the employees table, naming it employees2:

```
CREATE TABLE employees2 SELECT * FROM employees;
```

An identical table, employees2, will be added to the database.

Sometimes you need to create a table based on just a few columns found in a preexisting table. You can do so by simply specifying the columns within the CREATE SELECT statement:

```
CREATE TABLE employees3 SELECT first_name, last_name FROM employees;
```

Creating a Temporary Table

Sometimes it's useful to create tables that will have a lifetime that is only as long as the current session. For example, you might need to perform several queries on a subset of a particularly large table. Rather than repeatedly run those queries against the entire table, you can create a temporary table for that subset and then run the queries against it instead. This is accomplished by using the TEMPORARY keyword in conjunction with the CREATE TABLE statement:

```
CREATE TEMPORARY TABLE emp_temp SELECT first_name, last_name FROM employees;
```

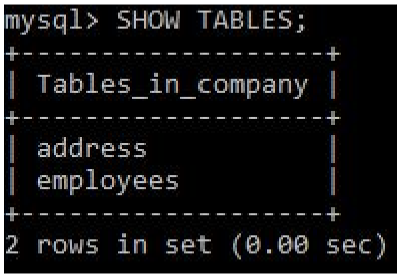
Temporary tables are created just as any other table would be, except that they're stored in the operating system's designated temporary directory, typically /tmp or /usr/tmp on Linux. You can override this default by setting MySQL's TMPDIR environment variable.

Viewing a Database's Available Tables

You can view a list of the tables made available to a database with the SHOW TABLES statement:

```
mysql>SHOW TABLES;
```

Note: *the screenshot below is based on the number of databases I have on my home computer, this view may differ from what you get at the time the above command is run.*



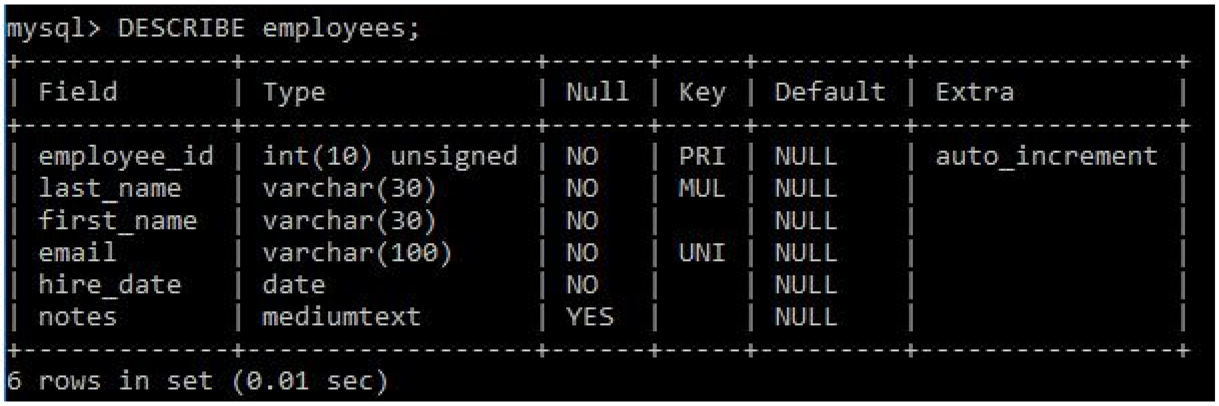
```
mysql> SHOW TABLES;
+-------------------+
| Tables_in_company |
+-------------------+
| address           |
| employees         |
+-------------------+
2 rows in set (0.00 sec)
```

Viewing a Table Structure

You can view a table structure using the DESCRIBE statement:

```
mysql>DESCRIBE employees;
```

Note: *the screenshot below is based on the number of databases I have on my home computer, this view may differ from what you get at the time the above command is run.*


```
mysql> DESCRIBE employees;
+-------------+------------------+------+-----+---------+----------------+
| Field       | Type             | Null | Key | Default | Extra          |
+-------------+------------------+------+-----+---------+----------------+
| employee_id | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| last_name   | varchar(30)      | NO   | MUL | NULL    |                |
| first_name  | varchar(30)      | NO   |     | NULL    |                |
| email       | varchar(100)     | NO   | UNI | NULL    |                |
| hire_date   | date             | NO   |     | NULL    |                |
| notes       | mediumtext       | YES  |     | NULL    |                |
+-------------+------------------+------+-----+---------+----------------+
6 rows in set (0.01 sec)
```

Alternatively, you can use the SHOW command like so to produce the same result:

```
mysql>SHOW columns IN employees;
```

Deleting a Table

Deleting a table, or dropping it, is accomplished via the DROP TABLE statement. Its syntax follows:

```
DROP [TEMPORARY] TABLE [IF EXISTS] tbl_name [, tbl_name,...]
```

For example, you could delete your employees table as follows:

```
DROP TABLE employees;
```

You could also simultaneously drop employees2 and employees3 tables like so:

```
DROP TABLE employees2, employees3;
```

Altering a Table Structure

You'll find yourself often revising and improving your table structures, particularly in the early stages of development. However, you don't have to go through the hassle of deleting and re-creating the table every time you'd like to make a change. Rather, you can alter the table's structure with the ALTER statement. With this statement, you can delete, modify, and add columns as you deem necessary. Like CREATE TABLE, the ALTER TABLE statement offers a vast number of clauses, keywords, and options. It's left to you to look up the gory details in the MySQL manual. This section offers several examples intended to get you started quickly, beginning with adding a column. Suppose you want to track each employee's birth date with the employees table:

```
ALTER TABLE employees ADD COLUMN birth_date DATE;
```

The new column is placed at the last position of the table. However, you can also control the positioning of a new column by using an appropriate keyword, including FIRST, AFTER, and LAST. For example, you could place the birthdate column directly after the lastname column, like so:

```
ALTER TABLE employees ADD COLUMN birth_date DATE AFTER last_name;
```

Whoops, you forgot the NOT NULL clause! You can modify the new column:

```
ALTER TABLE employees CHANGE birth_date birth_date DATE NOT NULL;
```

Finally, after all that, you decide that it isn't necessary to track the employees' birth dates. Go ahead and delete the column:

```
ALTER TABLE employees DROP birth_date;
```