Article: Learning SQL using phpMyAdmin

Learning SQL Using phpMyAdmin

Learning SQL Using phpMyAdmin

This tutorial is aimed at programmers, analysts, and designers of dynamic Web sites who want to learn the basics of SQL. I will use the MySQL (www.mysql.com) database server and its powerful phpMyAdmin (www.phpmyadmin.net) interface. The latter is an excellent tool for learning SQL.

Author: Marc Delisle

What is SQL?

Structured Query Language is a non-procedural language used to define, manipulate and retrieve data. It was developed by IBM (System/R project) in 1974-1979. The American National Standards Institute (ANSI) published in 1986 the first official standard of the language (later revised in 1989, 1992 and 1999), and since then, the industry has widely adopted SQL as *the* relational database language. Virtually every database system nowadays is interfaced through SQL.

The specific data architecture addressed by SQL is called the *relational architecture.* The various pieces of data (columns) are grouped into tables contained in databases, and we retrieve data using *relations* expressed between the tables.

In this article, we will use MySQL, a popular open-source implementation of SQL that is deployed by most Web host providers.

Toolkit for this guide

To be able to do the exercises in this guide, you will need an access to a MySQL server. Your interface to MySQL will be phpMyAdmin, a PHP application running on a PHP-enabled Web server. The book **Mastering phpMyAdmin for effective MySQL Management** is recommended for a comprehensive coverage of the phpMyAdmin tool.

This guide will show you the SQL syntax, sometimes by asking you to enter statements, and sometimes by letting you see how phpMyAdmin generates SQL statements based on your actions using the interface.

Creating Sample Tables

We will use a geographical information system as an example. We decide that we need information about cities and countries, so we design two tables, which will be part of a database called *geodb* (although any database name would do). To create the tables, we can use phpMyAdmin's Structure sub-page in Database view, or we can use the SQL query box to enter the appropriate statement:

The table creation is accomplished with the `CREATE TABLE` statement, in which we give the new table's name. The statement begins with `CREATE TABLE`, followed by the table name. Then, enclosed in brackets, we put the list of columns, and information about the keys. Each column is assigned a name, data type, the `NULL` or `NOT NULL` attribute (here, `NOT NULL` means the column cannot have a `NULL` value) and a default value, if appropriate.

```
CREATE TABLE cities (
  id int(11) NOT NULL auto_increment,
  city_name varchar(50) NOT NULL default '',
  latitude varchar(15) NOT NULL default '',
  longitude varchar(15) NOT NULL default '',
  population int(11) NOT NULL default '0',
  country_code char(2) NOT NULL default '',
  PRIMARY KEY  (id)
) TYPE=MyISAM AUTO_INCREMENT=1 ;
```

The id column is our primary key, a column which uniquely identifies each city. Its data type is `INT` (an integer number), and MySQL will assign unique numbers to it, thanks to the `auto_increment` attribute. Note that we cannot use the city name as a primary key, as some city names are not unique in the world. We also use an integer for the population data.

The other columns use character (`CHAR`) or variable character (`VARCHAR`) data types. When we know the exact length of data, it's better to use `CHAR`, specifying the length of the column as in `CHAR(2)`. Otherwise we use a variable character data type, which will take only the space needed by each piece of data, and we specify the maximum length, as in `VARCHAR(15)`.

After the columns list, we have some table-specific information, like its type, and the first value for the auto-increment column. SQL statements end with a semi-colon.

Having created our cities table, we do the same operation, this time for the countries table.

```
CREATE TABLE countries (
  country_code char(2) NOT NULL default '',
  country_name varchar(100) NOT NULL default ''
) TYPE=MyISAM;
```

We notice that the `country_code` column is present in both tables. This shows the relational principle: the `country_code` in `cities` refers to the same column in `countries`. This way, we save on space, having each country name only once in our database.

Another article (Migrating to InnoDB) will explain in greater details this relational technique.

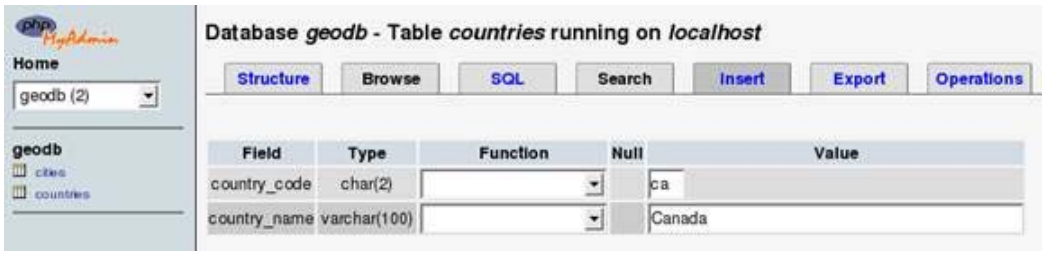
We are now ready to enter some data in our tables.

Data modification

In this section, we will learn the basic syntax for the `INSERT`, `UPDATE`, `DELETE`, and `SELECT` statements.

Adding Data with INSERT

Let's first examine the `INSERT` statement, by looking at the code phpMyAdmin generates when we do an Insert operation. We bring up the Insert sub-page, in Table view for the `countries` table, and we enter data about a country:



When we click `Go`, the data is inserted and phpMyAdmin shows us the `INSERT` statement used:

```
INSERT INTO `countries` ( `country_code` , `country_name` )
VALUES ('ca', 'Canada');
```

After the `INSERT INTO` part, we have the table name. In MySQL, we can enclose table names and column names within backticks, in case there are special characters in them, like reserved words or accented characters. Then we open a first set of

brackets, listing the columns in which we want to insert, separated by commas. The reserved word `VALUES` follows, then the last set of brackets enclosing the values, in the same order as the columns list. If the values have a character data type, we have to enclose them within quotes.

We can now insert a city.

```
INSERT INTO `cities` ( `id` , `city_name` , `latitude` , `longitude` , `population` ,
`country_code` )
VALUES ('', 'Sherbrooke', '45 23 59.00', '-71 46 11.00', 125000, 'ca');
```

Here, we put an empty value for id, because this column's auto-increment attribute will provide a value. We also see that the population value, being numeric, does not need to be surrounded by quotes.

Let's end this section by inserting some data for another country and city, which we will need later.

```
INSERT INTO `countries` ( `country_code` , `country_name` )
VALUES ('zh', 'China');

INSERT INTO `cities` ( `id` , `city_name` , `latitude` , `longitude` , `population` ,
`country_code` )
VALUES ('', 'Shanghai', '31 13 58.00', '121 26 59.99', 11000000, 'zh');
```

Updating Data with UPDATE

We first click on Browse for table cities, displayed our single row of data.

By clicking on the small pencil-shaped icon (or Edit link), we go to the Edit panel for this row. We decide to change the population value to 130000. After a click on Save, phpMyAdmin shows the following statement:

```
UPDATE `cities` SET `population` = '130000' WHERE `id` = '1' LIMIT 1 ;
```

Here we have the `UPDATE` keyword, followed by the table name. The `SET` keyword introduces the list of modifications (here only the population), which follows the format `column = new value`.

We now see the condition `WHERE `id` = '1'`, which uses the primary key information to limit the change to only this row, i.e. only this city.

The `limit 1` part is a safeguard added by phpMyAdmin, in case there would be no primary key defined, to avoid doing the change to more than one row.

More than one column can be changed in a single `UPDATE` operation:

```
UPDATE `cities` SET `city_name` = 'Sherbrooke, Qu&#65533;bec',
`population` = '130001' WHERE `id` = '1' LIMIT 1 ;
```

Deleting Data with DELETE

In Browse mode on table cities, clicking on the small red trash-can icon (or Delete link) brings up a dialog to confirm the execution of the following statement:

```
DELETE FROM `cities` WHERE `id` = '1' LIMIT 1 ;
```

The syntax is simple, involving just the table name, and the condition to apply for the delete operation.

> Omitting the WHERE condition in an UPDATE or DELETE operation is perfectly legal in SQL, but then the operation takes place on every rows of the table!

Retrieving Data with SELECT

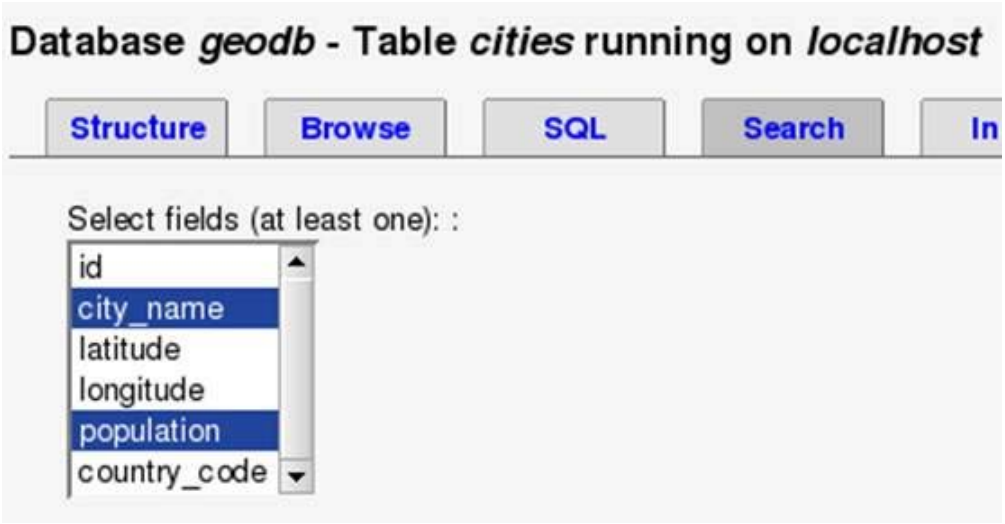
Retrieving information from our tables is probably the operation we do most of the times. This is the way to get answers to questions like "what are the cities with a population over a certain number?".

In fact, we previously did a `SELECT` when we clicked on the Browse link for table cities. This generated a simple form of the `SELECT` statement:

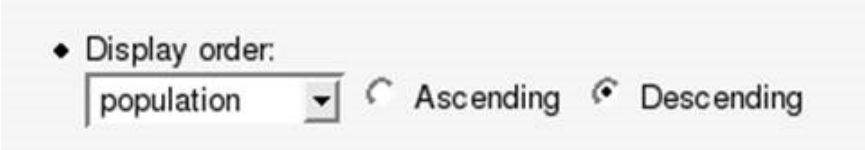
```
SELECT * FROM `cities` LIMIT 0,30;
```

Here, the asterisk means "all the columns". We add `FROM` and the name of the table which we want to query. The `LIMIT 0,30` means to start at row number 0 (the first one), and select a maximum of 30 rows.

Let's try a Search to see more options for the `SELECT`. We go to the Search sub-page for table cities, and we choose only some columns we need:



Then at the bottom of the page, we choose to display by the result by population in descending order:



Executing the search generates the following query:

```
SELECT `city_name` , `population`
FROM `cities`
WHERE 1
ORDER BY `population` DESC LIMIT 0,30
```

We see that the asterisk has been replaced by a comma-separated list of columns. A condition `WHERE 1` has been added by phpMyAdmin, this is a condition which is always true and selects all rows. We will see in a moment that we can replace it with some other condition. Also, the clause `ORDER BY` appears, followed by the column on which we want to sort results, and the keyword `DESC` for descending order (we could also use `ASC` for ascending).

Conditions

To easily add a condition, on the results page we can click on SQL-query: Edit, which brings the Query window popup. We add a condition on the country:

```
SELECT `city_name` , `population`
FROM `cities`
WHERE country_code = 'zh'
ORDER BY `population`  DESC
```

which displays all cities located in China (ok, we were a bit lazy with data entry, but you get the picture).

Conditions can be expressed using a rich array of operators and functions. Here are two examples:

Finding the Canadian cities with a population over 100000:

```
WHERE population > 100000 AND country_code = 'ca'
```

Finding the cities whose name starts with A:

```
WHERE city_name like 'A%'
```

Aggregate functions

Summary information may be generated by grouping on a specific column. Here we ask the average city population per country:

```
SELECT country_code, AVG(population)
FROM cities
```

```
GROUP BY country_code
```

Other possible aggregate functions are `MIN()`, `MAX()`, `SUM()` and `COUNT()`, which compute respectively the minimum value, maximum value, sum of values, and number of values. For example, we could get the number of cities per country with:

```
SELECT country_code, count(city_name)
FROM cities
GROUP BY country_code
```

Joins

Normally, a relational database involves many tables, linked on common keys. We may need at times to run queries on more than one table. Linking, or joining, tables can be done using different techniques; we will focus on a simple method involving key comparison.

In the following query, the `FROM` clause contains a comma-separated list of tables. In the columns list, we use the table name and a dot as a prefix before each column name (not strictly necessary if each column name is only present in one table).

```
SELECT cities.city_name, cities.population, countries.country_name
FROM cities, countries
WHERE cities.country_code = countries.country_code LIMIT 0,30
```

city_name	population	country_name
Sherbrooke, Québec	130001	Canada
Shanghai	11000000	China

Conclusion

The SQL language has much more to it than the basic statements that we have covered here. However, this article has covered the basics of SQL and how to use the phpMyAdmin tool to advance your knowledge of SQL.

About the author

Marc Delisle started contributing to phpMyAdmin in December 1998, when he made the first multi-language version. He has been actively involved since May 2001, as a developer and a project administrator.

He has worked since 1980 at Coll�ge de Sherbrooke, Qu�bec, Canada, as an application programmer and network manager. He has also been teaching networking, security, Linux servers, and PHP/MySQL application development.

Marc Delisle has recently published with Packt Publishing the first book on phpMyAdmin: **Mastering phpMyAdmin for effective MySQL Management**.