

Web Tech II – Class 10: Getting started with MySQL

J. Adrian Herbez – purplestatic@yahoo.com

Databases and tables

In order to build large, dynamic websites, it's necessary to come up with a structured way to store and retrieve your data. By and large, that's accomplished via the use of RDBMS (Relational DataBase Management Systems). The most popular such RDBMS (since it's both robust and free) is MySQL. The "SQL" in MySQL stands for "Structured Query Language", which is just a fancy way of saying that there is a specific way that you have to structure (build) your queries (questions) to the system in order to have them work out for you.

Databases are made up of tables, and its tables that hold the actual data. You can think of tables as actual files, and databases as folders that hold them. You probably won't need more than a single database for any given application (or perhaps even a collection of web apps), but you will need several tables. You probably need fewer databases (and more tables) than you think you do.

Creating databases

The specifics of how you go about creating databases on your system varies depending on your access to the server machines. If you are using a hosting service (such as GoDaddy or similar), you will likely need to use their interface to create the database. Since all of the actual data is stored in tables, there's not much to creating a database. All you really need to do (usually) is give the database a name and set a password for it. Note that this password should *NOT* be the same as the one you use to log in to your site itself- never share that with anyone.

If you have direct access to the server, you will likely use a secure telnet application (such as SSH) to connect to the server. Once there, you can start up MySQL with:

```
mysql -u root
```

You'll be prompted for a password (this part is just like logging into your site to upload files), and then you run queries from the command line. Creating a database from the command line would look something like the following:

```
CREATE DATABASE myDB;
```

...which would create a database named "myDB".

Using PHPMyAdmin

While companies will often expect you to run MySQL via command-line tools, services like GoDaddy and similar provide an easier way to get started with MySQL- PHPMyAdmin. You'll want to learn the command-line tools eventually, but in the meantime, PHPMyAdmin is a great way to get started.

Adding tables

A database doesn't do you any good until you add tables, since its the tables that actually store your data. A database without tables is like a file cabinet without files. A table is defined by a set of rows, with each row representing an object (like a book, a registered user, or a blog post). Each row is broken down into various columns that define the type of data the table stores.

So, if we wanted to have a users table, we might want to store the following:

```
user name  
user password (this should be hashed for storage)  
email address
```

The key component to getting things to work for you is ensuring that each row in each table is unique. If that's not the case, and there are multiple rows with exactly the same information across all columns/fields, then MySQL will get confused and won't be able to do its job.

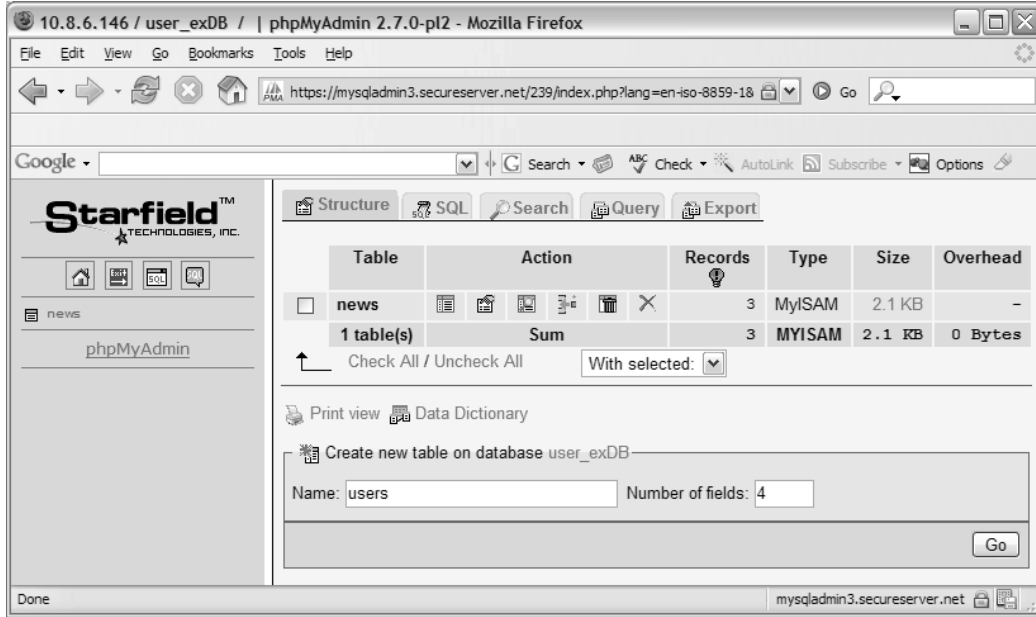
To create tables, you can use the following syntax with command-line MySQL:

```
CREATE TABLE table_name (create_definition,...)
```

...where *create_definition* is a list fo the names and types of columns you want to add to the table, along with various options.

Adding tables with PHPMyAdmin is fairly similar, though a bit easier to work with. You start by specifying how many columns you

want in the table. Remember that the number of columns means the number of distinct forms of data you want to store, not the number of entries. In the above example, we would need to have 4 columns- one each for username, password, and email, and one more to hold a unique identifier value to ensure that each row is unique. It's common practice to add a unique integer (whole number) ID column to tables, as that guarantees that each one is completely unique.



Once you've specified the name of the table and the number of fields, you'll need to tell PHPMyAdmin exactly what kind of data each of the fields/columns should hold.

Data in MySQL comes in lots of different types, which can be pretty daunting at first. It gets easier, though if you bear in mind that all of the types resolve to one of three basic types:

- numeric
- text / BLOB
- date / time

Using PHP with MySQL

PHP makes it very straightforward to both store and retrieve data into / out of a database. Whether you're storing or retrieving data, the basic process breaks down into three steps:

- 1) connect to the server with a specified username and password (NOT the same username/password you use to FTP)
- 2) select a database on the system (it's likely you'll only have one, but it is possible to have more)
- 3) run the appropriate MySQL command to either insert or retrieve data.

In order to connect to the database, you'll need to use the following command in your PHP script:

```
$con = mysql_connect( server, username, password );
```

Note that the `mysql_connect` command is set to fill up a variable with its results. That's useful, since it allows you to detect if something goes wrong with the connection. It's very common to follow up the above code with something like:

```
if (!$con)
{
    die("Could not connect to database".mysql_error());
}
```

If anything went wrong, the value returned by `mysql_connect` will be zero (false). By testing for ! (not) `$con`, we're effectively saying "if anything went wrong with the connection". If something did go wrong, the `die` command can be used to stop the rest of the script from running, and to display a meaningful error message to the user.

Once you've connected to the server, you'll need to select a database, which is very simple- just use `mysql_select_db`, as in:

```
mysql_select_db("users", $con);
```

...note the inclusion of the `$con` variable- if everything went as expected, the `mysql_connect` command should have returned a "link identifier" that can be used to run queries on the server.

Once you've both connected to the server and selected a database, you're ready to start inserting data and making queries.

Inserting data

To insert data into the database, you'll need to first have the right data. For the most part, you'll most likely be getting the data from web forms, but you can certainly get data from many other sources as well. Let's say that we have a new user, and we have their information stored in the following variables:

```
$username;  
$password;  
$email;
```

In order to add them to the database, we would need to use the INSERT statement, as in:

```
INSERT INTO table_name (username, pswd, email) VALUES ($username, $password, $email);
```

While the above command is exactly what we need to do, it isn't valid PHP- it's MySQL syntax. In order to run it from a PHP script, we'll need to wrap it up in a string variable and use the `mysql_query` command. Putting that together looks like the following:

```
// create a string command to insert the data  
$myQuery = "INSERT INTO table_name (username, pswd, email) VALUES ('$username', '$password', '$email')";  
  
// run the query in order to insert the values into the table  
mysql_query($myQuery, $con);
```

Retrieving data

Retrieving data is fairly similar to inserting it, but you use the SELECT statement, rather than INSERT. Also, you have to point the output of the query at a variable so that you can make use of the returned data. There are a lot of different ways to go about getting data out of the database, and it is the flexibility of the SELECT statement that makes MySQL so incredibly useful. A really simple query to get all of the users from a database might look like the following:

```
SELECT * FROM news
```

...where the * is a wildcard character meaning "everything". Once again, the MySQL query must be stored into a string and run by using the `mysql_query` command, as in:

```
$query = "SELECT * FROM news";  
$result = mysql_query($query);
```

Once you have the `$result`, you'll need to extract the data and do something with it, such as display it to the user. PHP makes that easy with the inclusion of the `mysql_fetch_array` command. The `mysql_fetch_array` command can be used to give you each row of the result as an associative array, with the index of each piece of data being equal to the name of that field. If we use the command as part of a while loop, it becomes easy to loop through and display all of the data.

Putting all of that together yields something like the following:

```
$result = mysql_query("SELECT * FROM news ");  
  
while ($row = mysql_fetch_array($result))  
{  
    echo "<div class='title'>";  
    echo $row['title'];  
    echo "</div><div class='news'>";  
    echo $row['text'];  
    echo "</div>";  
}
```