Database Connectivity
Dynamic Web Development [DWDDCO701]

Thomas D. Devine
http://www.noucamp.org
thomas.devine@lyit.ie

September 30, 2008

Contents

1 Document Information **5**

2 Connecting to a MySQL Database **5**
2.1 Opening and Using a Database Connection 5
2.2 Essential Functions for Accessing MySQL with PHP 7

3 SQL Lesson **8**
3.1 `SELECT` 8
3.1.1 `ORDER BY` and `LIMIT` 9
3.2 `INSERT` 9
3.3 `UPDATE` 10
3.4 `DELETE` 10

List of Figures

Listings

1 Connecting to a database using PHP 5

1 Document Information

This document has been produced primarily from the books *Learning PHP* [1] and *Web Database Applications with PHP & MySQL* [2]. There is no requirement for you to purchase these or any other book. These notes will be sufficient for any assignments and examination assessment for this module.

2 Connecting to a MySQL Database

In this section, we by showing the PHP scripting techniques to query the database tier and render HTML in a client-tier web browser. We present the basics of connecting to and querying the `footballDB` database using a simple query. The focus of this section is the DBMS interaction, not the presentation.

2.1 Opening and Using a Database Connection

In PHP a set of library functions are provided for executing SQL statements, as well as for managing result sets returned from queries. We overview these functions here and show how they can be combined to access the **MySQL DBMS**. Connecting to and querying a MySQL DBMS with PHP is a five-step process. Listing 1 shows a script – `listClubs.php` – that connects to the MySQL DBMS, uses the `footballDB` database, issues a query to select all the records from the clubs table, and reports the results as preformatted HTML text. The example illustrates six of the key functions for connecting to and querying a MySQL database with PHP. Each function is pre-fixed with the string `mysql_`. We explain the function of this script in detail in this section.

Listing 1: Connecting to a database using PHP

```
1  <html>
2  <head>
3    <title>Football Clubs</title>
4  </head>
5  <body>
6
7  <?php
8    $connection = mysql_connect("localhost");
9    mysql_select_db("footballDB", $connection);
10
11   $result = mysql_query ("SELECT * FROM clubs", $connection);
12
13   while ($row = mysql_fetch_row($result))
14   {
15     print "<li>";
16     for ($i=0; $i<mysql_num_fields($result); $i++)
17       print $row[$i] . " ";
18     print "</li>";
19   }
```

```
20
21   mysql_close($connection);
22 ?>
23
24 </body>
25 </html>
```

The five steps of querying a database are numbered in the comments in Listing 1, and they are as follows.

1. **Connect to the DBMS and use a database**. Open a connection to the MySQL DBMS using `mysql_connect()`. There is a single parameter – the hostname of the DBMS server to use. Once you connect, you can select a database to use through the connection with the `mysql_select_db()` function. In this example, we select the `footballDB` database. Let's assume here that MySQL is installed on the same server as the scripting engine and therefore, we can use localhost as the hostname. The function `mysql_connect()` returns a connection *handle*. A handle is a value that can be used to access the information associated with the connection. As discussed in Step 2, running a query also returns a handle that can access results.

2. **Run the query**. Let's run the query on the `footballDB` database using `mysql_query()`. The function takes two parameters: the SQL query itself and the DBMS connection to use. The connection parameter is the value returned from the connection in the first step. The function `mysql_query()` returns a result set handle resource.

3. **Retrieve a row of results**. The function `mysql_fetch_row()` retrieves one row of the result set, taking only the result set handle from the second step as the parameter. Each row is stored in an array `$row`, and the attribute values in the array are extracted in Step 4. A `while` loop is used to retrieve rows until there are no more rows to fetch. The function `mysql_fetch_row()` returns false when no more data is available.

4. **Process the attribute values**. For each retrieved row, a `for` loop is used to print with an `print` statement each of the attributes in the current row. `mysql_num_fields()` is used to return the number of attributes in the row; that is, the number of elements in the array. For the `clubs` table, there are two attributes in each row – `clubname` and `ground`.

   The function `mysql_num_fields()` takes as a parameter the result handle from Step 2 and, in this example, returns 2 each time it is called. The data itself is stored as elements of the array `$row` returned in Step 3. The element `$row[0]` is the value of the first attribute (the clubname) and `$row[1]` is the value of the second attribute (the ground).

   The script prints each row on a line, separating each attribute with a semi colon character. Steps 3 and 4 are then repeated.

5. **Close the DBMS connection** using `mysql_close()`, with the connection to be closed as the parameter.

2.2 Essential Functions for Accessing MySQL with PHP

`resource mysql_connect([string host], [string username], [string password])` Establishes a connection to the MySQL DBMS. The function returns a connection resource handle on success that can be used to access databases through subsequent commands. Returns false on failure.

The command has three optional parameters. When the DBMS runs on the same machine as the PHP scripting engine and the web server the first parameter can be localhost.

In Listing 1, the function call:-
`$connection = mysql_connect("localhost")`

connects to the MySQL DBMS on the local machine with the no username and no password. If the connection is successful, the returned result is a connection resource handle that should be stored in a variable for use as a parameter to other MySQL functions.

This function needs to be called only once in a script, assuming you don't close the connection.

`int mysql_select_db(string database, [resource connection])` Uses the specified `database` on a connection. In Listing 1, the database `footballDB` is used on the connection returned from `mysql_connect()`:-

```
mysql_select_db("footballDB", $connection)
```

`resource mysql_query(string SQL_command, [resource connection])` Runs the SQL statement `SQL_command`. In practice, the second argument isn't optional and should be a connection handle returned from a call to `mysql_connect()`. The function `mysql_query()` returns a resource – a result handle that can fetch the result set.

In Listing 1, the function call:-

```
$result = mysql_query("SELECT * FROM clubs", $connection)
```

runs the SQL query `SELECT * FROM clubs` through the previously established DBMS connection resource `$connection`. The return value is assigned to `$result`, a result resource handle that is used as a parameter to `mysql_fetch_row()` to retrieve the data.

`array mysql_fetch_row(resource result_set)` Fetches the result set data one row at a time by using as a parameter the result handle result set that was returned from an earlier `mysql_query()` function call. The results are returned as an array, and the elements of the array can then be processed with a loop statement. The function returns false when no more rows are available. In Listing 1, a `while` loop repeatedly calls the function and fetches rows into the array variable `$row` until there are no more rows available:-

```
while ($row = mysql_fetch_row($result))
```

```
{
  . . .
}
```

`int mysql_num_fields(resource result_set)` Returns the number of attributes associated with a result set handle `result_set`. The result set handle is returned from a prior call to `mysql_query()`. This function is used in Listing 1 to determine how many elements to process with the `for` loop that prints the value of each attribute.

`int mysql_close([resource connection])` Closes a MySQL connection that was opened with `mysql_connect()`. The connection parameter is optional. If it is omitted, the most recently opened connection is closed.

This function doesn't really need to be called to close a connection opened with `mysql_connect()`, because all connections are closed when a script terminates. But, it is considered good programming practice to do so.

3 SQL Lesson

3.1 SELECT

The `SELECT` command retrieves data from the database. The syntax of `SELECT` is:-

```
SELECT column1[, column2, column3, ...] FROM tablename
```

The `SELECT` query below retrieves the *clubname* and *ground* columns for all the rows in the *Clubs* table:-

```
SELECT clubname, ground FROM clubs
```

As a shortcut, you can use `*` (wildcard) instead of a list of columns. This retrieves all columns from the table. The `SELECT` below retrieves everything from the *Clubs* table:-

```
SELECT * FROM clubs
```

To restrict a `SELECT` statement so that it matches only certain rows, add a `WHERE` clause to it. Only rows that meet the tests listed in the `WHERE` clause are returned by the `SELECT` statement. The `WHERE` clause goes after the table name, as shown below:-

```
SELECT column1[, column2, column3, ...] FROM tablename
WHERE where_clause
```

The where_clause part of the query is a logical expression that describes which rows you want to retrieve. Below is a `SELECT` query with `WHERE` clauses:-

```
; Players valued more than 500000
SELECT playername FROM players WHERE value > 5000000
```

3.1.1 ORDER BY and LIMIT

Rows in a table don't have any inherent order. A database server doesn't have to return rows from a SELECT query in any particular pattern. To force a certain order on the returned rows, add an ORDER BY clause to your SELECT. The query below returns all the rows in the players table ordered by value, lowest to highest:-

```
SELECT playername,value FROM players ORDER BY value
```

To order from highest to lowest value, add DESC after the column that the results are ordered by:-

```
SELECT playername,value FROM players ORDER BY value DESC
```

3.2 INSERT

The INSERT command adds a row to a database table. The command below shows the syntax of INSERT:-

```
INSERT INTO table (column1[, column2, column3, ...])
VALUES (value1[, value2, value3, ...])
```

The INSERT query below adds a new player to the *Players* table:-

```
INSERT INTO players (playername, club, value)
VALUES ('Samuel Etto', 'Barcelona',20000000)
```

String values such as *Samuel Etto* have to have single quotes around them when used in an SQL query. The number of columns enumerated in the parentheses before VALUES must match the number of values in the parentheses after VALUES. To insert a row that contains values only for some columns, just specify those columns and their corresponding values:-

```
INSERT INTO players (playername, club)
VALUES ('Lionel Messi', 'Barcelona')
```

As a shortcut, you can eliminate the column list when you're inserting values for all columns:-

```
INSERT INTO players
VALUES ('Ronaldinho', 'Barcelona',15000000)
```

3.3 UPDATE

The UPDATE command changes data already in a table. The syntax of UPDATE is:-

```
UPDATE tablename SET column1=value1[, column2=value2, ...]
[WHERE where_clause]
```

The value that a column is changed to can be a string or number. For example:-

```
UPDATE players SET value = 5000000
```

The UPDATE command above will change the value of every player to 5000000. To just change some rows with an UPDATE query, add a WHERE clause. This is a logical expression that describes which rows you want to change. The changes in the UPDATE query then happen only in rows that match the WHERE clause. For example:-

```
UPDATE player SET value=value*2
WHERE club = 'Chelsea';
```

The command above will increase the value of *Chelsea* players by 100%.

3.4 DELETE

The DELETE command removes rows from a table. The syntax of DELETE is:-

```
DELETE FROM tablename [WHERE where_clause]
```

Without a WHERE clause, DELETE removes all the rows from the table. For example:-

```
DELETE FROM players
```

would remove all players.

With a WHERE clause, DELETE removes the rows that match the WHERE clause. For example:-

```
DELETE FROM players WHERE club = 'Chelsea'
```

References

[1] David Sklar. *Learning PHP*. O'Reiily, 2004.

[2] Hugh Williams and David Lane. *Web Database Applications with PHP & MySQL*. O'Reiily, 2002.

If the connection is successful, the returned result is a connection resource handle that should be stored in a variable for use as a parameter to other MySQL functions. This function needs to be called only once in a script, assuming you donâ€™t close the connection. int mysql select db(string database, [resource connection]) Uses the specified database on a connection. In Listing 1, the database footballDB is used on the connection returned from mysql connect():mysql select db(â€footballDBâ€, $connection) resource mysql query(string SOL command, [resource connection]) Runs the SQL statement SQL ...Â [2] Hugh Williams and David Lane. Oâ€™Reiily, 2002. Web Database Applications with PHP & MySQL. 11. Related documents. 23. Database Systems, 8th Edition 23 Internet Databases â€¢ Web database connectivity allows new innovative services that: â€“ Permit rapid response by bringing new services and products to market quickly â€“ Increase customer satisfaction through creation of Web-based support services â€“ Yield fast and effective information dissemination through universal access. Open Database Connectivity. 585-780-701 Issue 1.0 May 2002. COMPAS ID 89705. Â© 2002, Avaya Inc. All Rights Reserved.Â This web site includes telephone numbers for escalation within the United States. For escalation telephone numbers outside the United States, select Global Escalation List. Providing Telecommunications Security.Â A relational database in which CMS stores much of its data, including administration and historical data. The database is an Informix Standard Engine (SE) for CMS loads that are R3V8 or earlier, and Informix Dynamic Server (IDS) for CMS R3V9. ODBC. Open Database Connectivity. Exam 701: DevOps Tools Engineer. Exam Objectives Version: Version 1.0. Exam Code: 701-100. About Objective Weights: Each objective is assigned a weighting value. The weights indicate the relative importance of each objective on the exam.Â 701.1 Modern Software Development (weight: 6). Weight: 6. Description: Candidates should be able to design software solutions suitable for modern runtime environments. Candidates should understand how services handle data persistence, sessions, status information, transactions, concurrency, security, performance, availability, scaling, load balancing, messaging, monitoring and APIs. Furthermore, candidates should understand the implications of agile and DevOps on software development. Key Knowledge Areas