



## PHP – MySQL Architecture

- PHP and MySQL interact in the following way:
  - PHP opens up a connection to MySQL
  - PHP executes a SQL query via the connection
  - The result set is stored in a variable
  - The result set can then be manipulated to access the data

### Establishing a MySQL Connection

- To open up a connection to a MySQL Server we use the following syntax:

```
$link = mysql_connect (host, user, password)
```

Where **\$link** is the variable which holds the link to the MySQL database. **host** is the name or IP address of the MySQL host. **user** is the MySQL username to be used in accessing the database. **password** is the password for the MySQL username used.

- Once a link has been opened, we are now ready to start executing SQL queries.
- It is a good idea to first set the active database being queried as follows :

```
mysql_select_db (db, link)
```

Where **db** is the name of the database to be queried and **link** is the variable containing the database link (e.g.: \$link).



## Executing SQL Queries

- SQL queries are executed by calling the `mysql_query()` function with the following syntax:

```
mysql_query (query)
```

- The query is then executed on the MySQL server.

e.g.:

```
mysql_query ("INSERT INTO `users` (`username`, `password`) VALUES ('me', 'm3");
mysql_query ("DELETE FROM `users` WHERE `username` LIKE 'm%");
mysql_query ("UPDATE `users`
              SET `password` = 'm33'
              WHERE `username` = 'me');
```

## Retrieving Data

- To retrieve data using a SELECT query we do so by catching the result of the `mysql_query()` function and performing various operations on it.

```
$result = mysql_query (query);
```

- For example :
- ```
$result = mysql_query ("SELECT * FROM `users`");
```
- We could then fetch the results row by row in the following way:

```
$result = mysql_query ("SELECT * FROM `users`");
while ($row = mysql_fetch_row($result)) {
    $username = $row[0];
    $password = $row[1];
    print "Username is $username and password is $password";
}
```

In the above code, we used the `mysql_fetch_row()` function to fetch a row of data from the results of the SELECT query, and then move the pointer to the next row. Thus, the next time we call `mysql_fetch_row()`, we will get the new row and move the pointer onto the next row yet again.

The `mysql_fetch_row()` function is passed the result set ( e.g.: `$result` ) and returns a single-dimensional array containing the data from the current row.

## Native MySQL Functions

- [mysql\\_affected\\_rows](#) — Get number of affected rows in previous MySQL operation
- [mysql\\_change\\_user](#) — Change logged in user of the active connection
- [mysql\\_client\\_encoding](#) — Returns the name of the character set
- [mysql\\_close](#) — Close MySQL connection
- [mysql\\_connect](#) — Open a connection to a MySQL Server
- [mysql\\_create\\_db](#) — Create a MySQL database
- [mysql\\_data\\_seek](#) — Move internal result pointer
- [mysql\\_db\\_name](#) — Get result data
- [mysql\\_db\\_query](#) — Send a MySQL query
- [mysql\\_drop\\_db](#) — Drop (delete) a MySQL database
- [mysql\\_errno](#) — Returns the numerical value of the error message from previous MySQL operation
- [mysql\\_error](#) — Returns the text of the error message from previous MySQL operation
- [mysql\\_escape\\_string](#) — Escapes a string for use in a mysql\_query
- [mysql\\_fetch\\_array](#) — Fetch a result row as an associative array, a numeric array, or both
- [mysql\\_fetch\\_assoc](#) — Fetch a result row as an associative array
- [mysql\\_fetch\\_field](#) — Get column information from a result and return as an object
- [mysql\\_fetch\\_lengths](#) — Get the length of each output in a result
- [mysql\\_fetch\\_object](#) — Fetch a result row as an object
- [mysql\\_fetch\\_row](#) — Get a result row as an enumerated array
- [mysql\\_field\\_flags](#) — Get the flags associated with the specified field in a result
- [mysql\\_field\\_len](#) — Returns the length of the specified field
- [mysql\\_field\\_name](#) — Get the name of the specified field in a result
- [mysql\\_field\\_seek](#) — Set result pointer to a specified field offset
- [mysql\\_field\\_table](#) — Get name of the table the specified field is in
- [mysql\\_field\\_type](#) — Get the type of the specified field in a result
- [mysql\\_free\\_result](#) — Free result memory
- [mysql\\_get\\_client\\_info](#) — Get MySQL client info
- [mysql\\_get\\_host\\_info](#) — Get MySQL host info
- [mysql\\_get\\_proto\\_info](#) — Get MySQL protocol info
- [mysql\\_get\\_server\\_info](#) — Get MySQL server info
- [mysql\\_info](#) — Get information about the most recent query
- [mysql\\_insert\\_id](#) — Get the ID generated from the previous INSERT operation
- [mysql\\_list\\_dbs](#) — List databases available on a MySQL server
- [mysql\\_list\\_fields](#) — List MySQL table fields
- [mysql\\_list\\_processes](#) — List MySQL processes
- [mysql\\_list\\_tables](#) — List tables in a MySQL database
- [mysql\\_num\\_fields](#) — Get number of fields in result
- [mysql\\_num\\_rows](#) — Get number of rows in result
- [mysql\\_pconnect](#) — Open a persistent connection to a MySQL server
- [mysql\\_ping](#) — Ping a server connection or reconnect if there is no connection
- [mysql\\_query](#) — Send a MySQL query
- [mysql\\_real\\_escape\\_string](#) — Escapes special characters in a string for use in a SQL statement
- [mysql\\_result](#) — Get result data
- [mysql\\_select\\_db](#) — Select a MySQL database
- [mysql\\_set\\_charset](#) — Sets the client character set
- [mysql\\_stat](#) — Get current system status
- [mysql\\_tablename](#) — Get table name of field
- [mysql\\_thread\\_id](#) — Return the current thread ID
- [mysql\\_unbuffered\\_query](#) — Send an SQL query to MySQL, without fetching and buffering the result rows

## Error Handling

- MySQL errors that occur can be retrieved by using the **mysql\_error()** function as follows :

```
print "An error has occurred : " . mysql_error();
```

- This could be called after calling the **mysql\_query()** or **mysql\_connect()** functions.



## LESSON 6 – PHP : OOP

### Outline

- Introduction to OOP
- How objects operate within PHP
- Creating a class
- Constructors
- Debugging methods

## Introduction to OOP

- a programming paradigm that uses "objects" and their interactions to design applications and computer programs. It is based on several techniques, including encapsulation, modularity, polymorphism, and inheritance.
- A **class** is a collection of variables and functions working with these variables. Variables are defined by *var* and functions by *function*.

## OOP in PHP

- Since PHP version 5, the inclusion of the Zend2 engine has made a huge difference to the way PHP operates.
- From PHP 5 onwards, we now have excellent support for Object Orientation in PHP.

## Creating a Class

- Classes are created as follows :

```
<?php

class class_name {
    var $var_name;

    function __construct() {
        // some code
    }

    function function_name() {
        // some code
    }
}

?>
```

## Constructors

- In PHP4, the constructor is merely a function declared within a class of the same name as the class. ie.: **function class\_name() {}**
- In PHP5, the constructor is a special function named : function **\_\_construct() {}**

## Debugging Methods

- `var_dump(variable)`
  - By calling the **`var_dump()`** function and passing an object to it, the complete tree structure of that object, along with its contents, will be dumped onto the web browser (or terminal if you are using it in a command line application).
  
- `die()`
  - By printing a string to the browser followed by the `die()` function, one can check to see if that particular piece of code is being executed.



## Ralfe Poisson

**[ralfepoisson@gmail.com](mailto:ralfepoisson@gmail.com)**

Please email me with any queries or questions about the course content.

---