PHP and MySQL

Database Systems

Michael Pound

This Lecture

- PHP
  - Variables
  - Arrays
  - IF...ELSE statements
  - Loops
  - Connecting to MySQL
- Further reading
  - W3Schools online tutorials at http://www.w3schools.com/php/

The Limitations of SQL

- SQL is not a general purpose language
  - It is designed to create, modify and query databases
  - It is non-procedural, so doesn't contain normal programming constructs
  - Cannot handle platform-specific challenges such as formatting output

Extending SQL

- Some DBMSs add programming structures such as variables and loops to SQL.
  - Very specific to a DBMS
  - Essentially a new language that includes SQL
  - Not hugely flexible
- Connect to SQL from another language
  - Access SQL to run the relevant queries
  - All other work can be done using procedural code

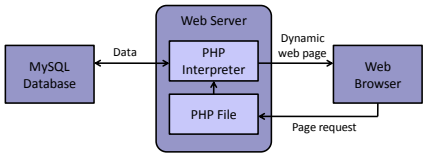
ODBC

- Connections to databases from programs are often handled using Open DB Connectivity
  - Provides a standard interface for communication with a DBMS
  - Can run queries, updates etc.
  - Results of queries can be used inside the program code

PHP

- PHP is a free, server-side scripting language
  - Often embedded into web pages to produce dynamic content
  - Can connect to most modern DBMSs, and those implementing ODBC.
  - Contains specialised functions for connecting to MySQL

How does PHP work?



Running PHP in the School

- Here are the steps required to get PHP to run on the School computers.
  - All files must be text files with a .php extension. E.g. index.php
  - All files must be in, or in a sub-directory of, H:/public_html/
  - You must have execute rights on these files. To do this you can use `chmod 600 file.php` from the command line
  - You can then run the files in an internet browser by going to http://avon.cs.nott.ac.uk/~username/file.php
  - For more info, visit http://support.cs.nott.ac.uk/help/docs/webpages/lphp/
- Note: You cannot run these php scripts outside the university for security reasons

PHP Basics

- PHP is procedural code that can be embedded into html documents inside php tags. Like this:

```
<html>
<body>

<?php
  // This is a comment
  // Some php code goes in here
?>

</body>
</html>
```

PHP Basics

- You can have any number of php blocks, separated by html. All php blocks will be connected when the file is run
- Code you write in an earlier block can be seen by code you write in later blocks. This will be important later
- Anything outside a php block is HTML text

Outputting Text

- Inside a PHP block, you can output text using the echo command. Like SQL and C, commands end with a semicolon:

```
<html>
<body>

<?php
  echo "This will be output as text!";
?>

</body>
</html>
```

Outputting HTML

- Remember, you're working in an HTML document, so anything you output will be read by the browser as HTML:

```
<html>

<?php
  echo "<head>";
  echo "<title>Title of the Page</title>";
  echo "</head>";
?>

<body>
</body>
</html>
```

Variables

- All programming languages use variables as a means to store values using names. For example, to create a number, called "num1" that has a value of 5:

```
$num1 = 5;
```

- PHP is a weakly typed language, which means you don't need to specify that num1 is of type "integer", because it works it out.

Variables

- Variables in PHP act much like in C, but remember to always use $

```
<?php
  $var1 = 5;
  $var1 = $var1 + 10;
?>
```

Then later:

```
<?php
  echo $var1;
?>
```

Strings

- Strings are lists of characters
  - Similar to varchar(n) in SQL
  - Can be declared using 'single' or "double" quotes
  - Can be appended together using '.'

```
<?php
  $var1 = "Hello";
  $var2 = "everybody";
  echo $var1 . " " . $var2;
?>
```

Arrays

- Sometimes it is more helpful to store variables in lists rather than as individual names. For example:

```
<?php
  $var1 = 2;
  $var2 = 4;
  $var3 = 8;
  $var4 = 16;
  $var5 = 32;
  echo $var1 . ", " . $var2 . ", " .
    $var3 . ", " . $var4 . ", " . $var5;
?>
```

Arrays

- For even a few variables, this will become messy. An alternative is to store them in a list structure, called an Array
- Arrays act like normal variables, but hold much more data
- Arrays are lists, and individual elements are accessed by the [] operator

```
<?php
  $var1 = array(2,4,8,16,32);
  echo $var1[3];
?>
```

Arrays

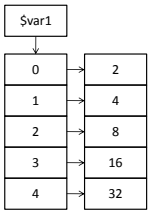
- Arrays are usually accessed by a number that represents the position of the variable we want
- Arrays can also be created and accessed by a keyword:

```
<?php
  $courses = array("DBS"=>"Database
    Systems", "PRG"=>"Programming");
  echo $courses["DBS"];
?>
```

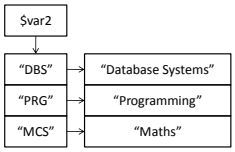
- Arrays are important because MySQL will give us an array of data when we write a query.

Array Examples

```
$var1 = array(
  2,4,8,16,32);
```




```
$var2 = array(
  "DBS"=>"Database Systems",
  "PRG"=>"Programming",
  "MCS"=> "Maths");
```



IF...ELSE

- Sometimes we might want to choose what code to run depending on our variables:

```
if (condition)
{
  // Code to run if condition is true;
}
```

IF...ELSE

- Conditions can be boolean variables, or other expressions.
- Conditions will include a single IF, any number of ELSE IFs, and then an optional ELSE
- Conditional operators are similar to those in MySQL.
- E.g. <, >, ==, !=, <>, etc.

```
$var1 = true;
$var2 = 15;
if ($var1)
{
  // Code
}
else if ($var2 < 5)
{
  // Code
}
```

Loops

- Sometimes we need to run similar code multiple times
- We can use a loop to run the same code repeatedly
- Four types of loops (We can get away with only while loops for this course)
  - WHILE
  - DO…WHILE
  - FOR
  - FOREACH

While Loops

- While loops are structured like this:

```
while (condition)
{
  // Code a
}
```

- Code a will be run repeatedly until that condition is false

Do...While Loops

- Do...While loops are structured like this:

```
do
{
  // Code a
} while (condition);
```

- Code a will be run once, then repeatedly until that condition is false

For Loops

- Do…While loops are structured like this:

```
for (initialisation; condition;
  increment)
{

}
```

- Code a will be run once, then repeatedly until that condition is false

Foreach Loops

- Foreach loops are not in C, but they are in Java, C++ C#, Objective C, Haskell etc.
  - Sometimes the foreach will still use the FOR keyword
  - Only used for iterating arrays

```
foreach ($array as $value)
{
  // Do something with
  //  each $value
}
```

Foreach Loops

- Foreach loops can also obtain keys for associative arrays

```
foreach ($array
  as $key => $value)
{
  // Do something with
  //  each $key, $value pair
}
```

- Foreach loops exist mainly for convenience

Functions

- If you wish to reuse code, you can put it in a function to access it later.
- There are numerous PHP functions you will find useful, e.g.
  - `count($array);`
  - `mysql_close($connection);`
  - `print_r($array);`
  - `mysql_real_escape_string($s, $c)`

Functions

- Functions are defined like this:

```
function <name> (<parameters>)
{
  // Do something
  // return;
  // or return <value>;
}
```

Functions

- An example of a function. Notice that you need not specify parameter or return types:

```
function factorial($val)
{
  return $val
    * factorial ($val - 1);
}
```

- ***Important***: During the coursework, write all functions in your main index.php file

$_GET and $_POST

- GET and POST are PHP global associative array variables that hold information passed to the PHP script
- $_GET
  - Holds information passed to the page via the URL
  - E.g. http://www.something.com/index.php?v=12
- $_POST
  - Holds information passed to the page via POST
  - Post variables are usually sent upon HTML form submissions

$_GET

- HTML Get variables are passed in the URL, after a ? and separated by &
- For example:

  http://www...com/index.php?age=19&name=Tim
- Inside our PHP script, we can access the values using $_GET['varname']

```
echo $_GET['name'] . " is " .
   $_GET['age'] . " years old";
```

$_POST

- HTML Post variables are passed separately, usually during a form submission
- For example:

```
<form action="index.php" method="post">
 Name: <input type="text" name="fname" />
 Age: <input type="text" name="age" />
 <input type="submit" />
</form>
```

- When the form is submitted, the page index.php will contain values in $_POST for 'fname' and 'age'

GET or POST

- While they essentially do the same thing, GET and POST are quite different:
  - Because GET parameters are passed in the URL, you can bookmark a script along with parameters
  - GET variables are easy to hack, and raise security issues
  - POST variables are generally unseen, so more secure
  - POST variables can be any size, GET variables should be short, e.g. 2000 chars max
  - Avoid GET when the values will be used to change the server information, e.g. Update a database

Connecting to MySQL

- PHP includes various functions for communicating with a MySQL server

```
mysql_connect('server', 'username', 'password');
```

  - Connects to the database and returns a connection resource
  - Host will usually be 'mysql.cs.nott.ac.uk' or 'localhost' if you're running at home

```
mysql_select_db('username', connection resource);
```

  - Will change the server to required database
  - Will return a boolean stating whether this action was successful

Connecting to MySQL

- In both the previous commands, if anything goes wrong, we should stop processing the PHP file
- You can terminate a PHP script using the `die` keyword:

```
die ("A problem has occurred!");
```

Connecting to MySQL

```
<?php
  $conn = mysql_connect('mysql.cs.nott.ac.uk',
    'username', 'password');
  if(!$conn)
  {
    die ("Error connecting to MySQL: " .
      mysql_error();
  }
  $db_select_success = mysql_select_db('username',
    $conn);
  if(!db_select_success)
  {
    die ("Error selecting database: " . mysql_error());
  }
?>
```

Includes

- Keeping our password in plain text inside our PHP document isn't very secure
- In PHP you can include code from other files for reuse later
- In this case, we can separate out our connection code for security. It also makes our code more concise.

Includes

- There are 4 commands that can include files:

`include(file.php)`
  - Includes all code from file.php at this location in the current php script

`include_once(file.php)`
  - As above, but only once. If you include_once a second time, nothing will happen

`require(file.php)` / `require_once(file.php)`
  - As above, but if any errors occur in the included file, the php scripting will stop immediately

Includes

mainfile.php

```
<html>
<head>
  <title>Title</title>
</head>

<?php

require_once('dbconnect.php');
// Some code that uses our
// database connection goes
// here

?>

</body
</html>
```

dbconnect.php

```
$conn =
  mysql_connect('mysql.cs.nott.ac.uk',
    'username', 'password');

  if(!$conn)
  {
    die ("Error connecting to MySQL: " .
      mysql_error();
  }

  $db_select_success =
    mysql_select_db('username', $conn);

  if(!db_select_success)
  {
    die ("Error selecting database: " .
      mysql_error());
  }
```

Using a MySQL Connection

- All SQL commands are sent to the server using the following functions:

`mysql_query("SQL Statement", $conn);`
  - Sends the SQL statement to the database at the given connection

`mysql_query("SQL Statement");`
  - Sends the SQL statement to the database you most recently connected to using `mysql_connect();`

Example Query

- You can use any SQL command via the `mysql_query()` function. For example:

```
$query = "CREATE TABLE Artist(
  artID INT NOT NULL AUTO_INCREMENT,
  artName VARCHAR(255) NOT NULL,
  CONSTRAINT pk_art PRIMARY KEY (artID))";

$success = mysql_query($query);
// success will be true if the table was
// created
```

SELECT in PHP

- For SELECT, SHOW and DESCRIBE commands, `mysql_query()` will return a set of results:

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

- $result will now hold all our returned rows

Using SELECT Results

- To use the values in $result, we can use the following command:

```
$row = mysql_fetch_array($result);
```

  - $row will be an array containing all the data from one row of our result set
  - Each time we use the above statement the next row will be returned
  - When no rows are left, $row will be `false`

Using SELECT Results

- Because `mysql_fetch_array()` will return false when no rows remain, we can use a while loop to make things easier:

```
while ($row = mysql_fetch_array($result))
{
  // Use the data in $row
}
// We reach this point when we have used
// every row
```

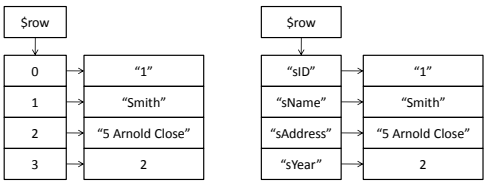
Using SELECT Results

- Once we have each row individually, we can use the data like any regular array:

```
while ($row = mysql_fetch_array($result))
{
  echo "Artist ID: " . $row['artID'];
  echo "Artist Name: " . $row['artName'];
}
```

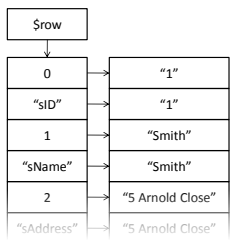
Associative or Numeric

```
$row = mysql_fetch_array(   $row = mysql_fetch_array(
  $result, MYSQL_NUM);        $result, MYSQL_ASSOC);
```



Both

```
$row = mysql_fetch_array(   $row = mysql_fetch_array(
  $result);                   $result, MYSQL_BOTH);
```



HTML Tables

- Sometimes it might be useful to output our results into an HTML Table. A table takes the following form:

```
<table>
 <tr>
  <td>Row 1 Col 1</td>
  <td>Row 1 Col 2</td>
 </tr>
 <tr>
  <td>Row 2 Col 1</td>
  <td>Row 2 Col 2</td>
 </tr>
</table>
```

HTML Tables

```
<table>
 <tr>
  <td>Row 1 Col 1</td>
  <td>Row 1 Col 2</td>
 </tr>
 <tr>
  <td>Row 2 Col 1</td>
  <td>Row 2 Col 2</td>
 </tr>
</table>
```

Row 1 Col 1	Row 1 Col 2
Row 2 Col 1	Row 2 Col 2

Creating a table in PHP

- Creating a table in php is simply a case of using ECHO to output the necessary tags.

```
echo "<table>";
while ($row =
           mysql_fetch_array($result))
{
  echo "<tr>";
  echo "<td>" . $row['artID'] . "</td>";
  echo "<td>" . $row['artName'] . "</td>";
  echo "</tr>";
}
Echo "</table>";
```

Setting up PHP at home

- To set up PHP and MySQL at home, you need:
  - A web server e.g. Apache
  - MySQL server
  - PHP 5.3
- XAMPP contains all of the above and some other useful things. All are installed at the same time, and set up for you.

http://www.apachefriends.org/en/xampp.html