

PHP and MySQL

(An introduction only!)

G51DBS Database Systems

Jason Atkin

Last lecture

- "[Every] non-key [attribute] must provide a fact about **the key, the whole key, and nothing but the key**"
[Bill Kent, as quoted on Wikipedia ;)]
- 1NF: **A key exists**, cell contents atomic
- 2NF: Non-key attributes depend on the **whole** key
- 3NF: Non-key attributes depend on **nothing but** the key
- BCNF: **All attributes** depend on nothing but the key

This and next lecture

- PHP
 - Variables
 - Arrays
 - IF...ELSE statements
 - Loops
 - Connecting to MySQL
 - Formatting select results
- 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
 - It 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
 - Often 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

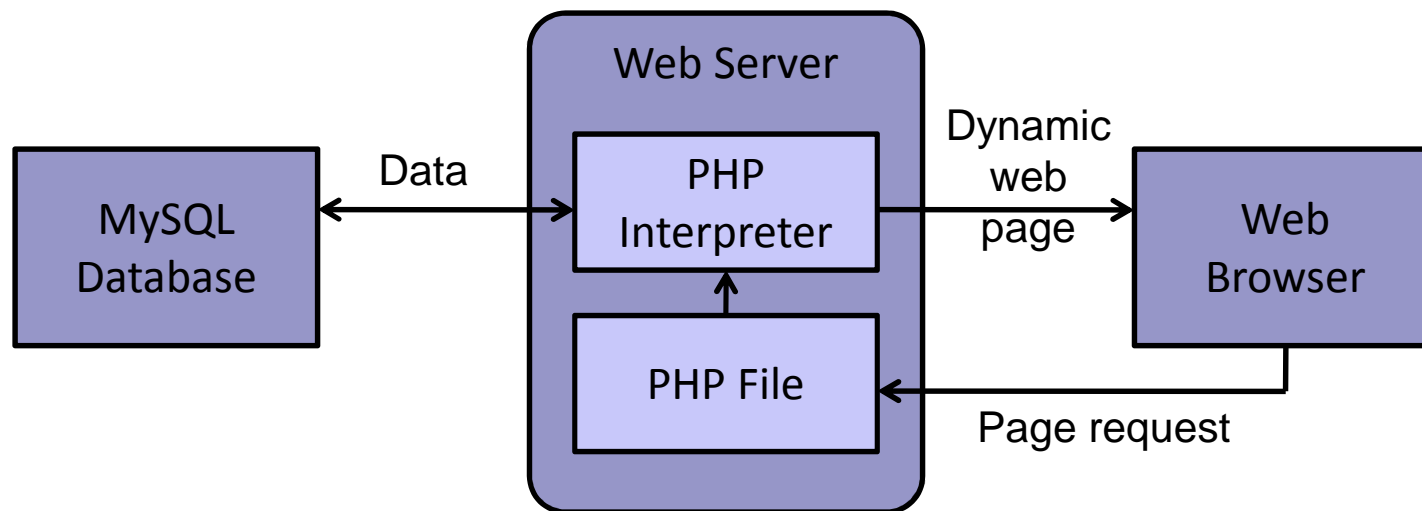
ODBC

- 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
 - We will use a very small subset of the functionality
 - This is a module on databases, not web pages
 - Easy to combine this with your web knowledge

How does PHP work?



Running PHP in the School

- Steps required to get PHP to run on the School computers:
 - All files must be text files, with .php extension, E.g. index.php
 - All files must be in H:/public_html/ (or a sub-directory)
 - You must have read permission on these files. 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 from outside the university for security reasons (must appear to be on campus!)

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
- Commands end with a semicolon, like SQL, C, Java, C++, ...

```
<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 imperative programming languages use variables as a means of storing values using names.
- For example, to create a numeric variable, 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 unhelpful to have a lot of similar variables:

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

- For even a few variables, this will become messy.
- It would be nice to be able to group them

Arrays

- Arrays act like normal variables, but hold much more data, as in Java/C/C++ etc
- Arrays are lists, and individual elements are accessed by the [] operator
- Arrays are usually accessed by a number that represents the position of the variable we want

```
<?php
```

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

```
    echo $var1[3];
```

```
?>
```

Arrays

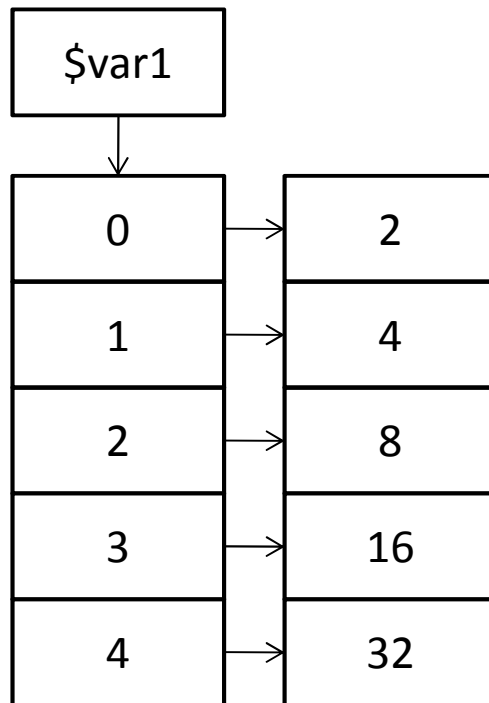
- Arrays can also be created and accessed by a keyword:

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

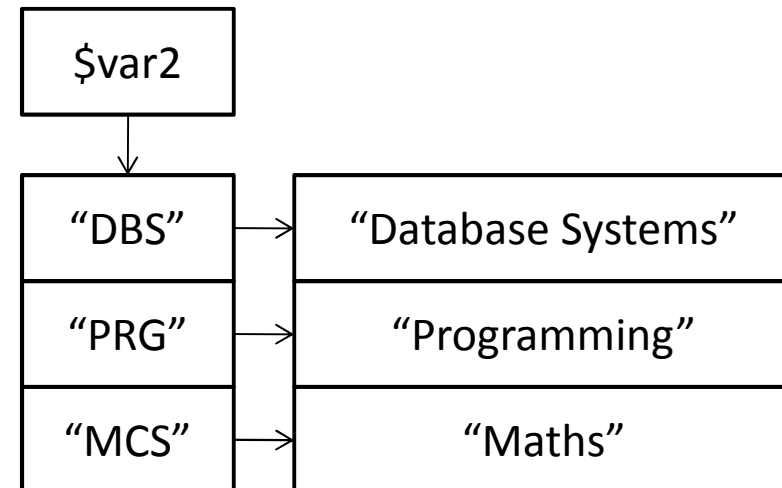
- Arrays are important because MySQL will give us an array of data when we write a query

Array Examples

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



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



While and Do...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 are structured like this:

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

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

Conditions and booleans

- When converting to boolean, the following values are considered **FALSE**:
 - the boolean **FALSE** itself
 - the integer 0 (zero)
 - the float 0.0 (zero)
 - the empty string, and the string "0"
 - an array with zero elements
 - an object with zero member variables (PHP 4 only)
 - the special type NULL (including unset variables)
 - SimpleXML objects created from empty tags
- Every other value is considered **TRUE**, including:
 - Result sets, non-empty arrays, non-zero numbers
- Source:
<http://php.net/manual/en/language.types.boolean.php>

Functions

- 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)`
- If you wish to reuse code, you can put it in your own function to access it later, see later

Connecting to the database and making your first PHP page

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** function:

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

- Output the message then terminate the script

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

- Sometimes you may want to do the same thing in multiple files
 - E.g. connect to the database
- Also, keeping our password in plain text inside our PHP document isn't very secure
 - What if something goes wrong and people can access the php code rather than the output?
 - Or you accidentally make the file readable
 - By keeping it in a different file, two things need to go wrong
- In PHP you can include code from other files for reuse later
- In this case, we can separate out our connection code for reuse and 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 returned rows

Using SELECT Results

- To use the values in `$result`, we can use the following command to get the next row of the result, as an array:

```
$row = mysql_fetch_array($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
```

- Example:

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

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>
```

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>";
```

Next lecture

**A few more PHP comments
and examples**

Coursework

- Coursework 1 is out (and has been for a while)
 - Your tutorial with your personal tutor should help you to do it
 - **I have gained permission to delay the submission date to 21st March for this reason – submit early if you can**
 - Muhammed Ahmed's ER-diagram tool should help
 - Use the labs to ask questions! (Not many people!)
- Coursework 2 will come out soon
 - Deadline 2nd April – not long after CW1 now!
 - Select statements and PHP (today's lectures)

See http://www.cs.nott.ac.uk/~jaa/dbscw2014_demo/dbscw2014.php

PHP and MySQL

(Part 2 – more PHP things)

G51DBS Database Systems

Jason Atkin

This lecture

- PHP
 - IF...ELSE statements
 - Other loops
 - Formatting select results
- Examples
- Further reading
 - W3Schools online tutorials at <http://www.w3schools.com/php/>

Control statements

IF...ELSE

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

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

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

Conditions and booleans

- When converting to boolean, the following values are considered **FALSE**:
 - the boolean **FALSE** itself
 - the integer 0 (zero)
 - the float 0.0 (zero)
 - the empty string, and the string "0"
 - an array with zero elements
 - an object with zero member variables (PHP 4 only)
 - the special type NULL (including unset variables)
 - SimpleXML objects created from empty tags
- Every other value is considered **TRUE**, including:
 - Result sets, non-empty arrays, non-zero numbers
- Source:
<http://php.net/manual/en/language.types.boolean.php>

For Loops

- For loops are structured like this:

```
for (  
    initialisation ;  
    condition ;  
    increment )  
{  
    // Code a  
}
```

- Initialisation will happen
- Code a will be **run repeatedly** until the condition is false
- After each execution**, the increment operation will be executed

```
for (  
    $i = 1 ;  
    $i <= 10 ;  
    $i++ )  
{  
    echo $i;  
}
```

Foreach Loops

- Foreach loops are not in C, but they are in Java, C++, C#, Objective C, Haskell etc.
- Only used for iterating arrays

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

- Foreach loops can also obtain keys for associative arrays

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

- foreach loops exist mainly for convenience

Defining functions

Defining your own functions

- Functions are defined like this:

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

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

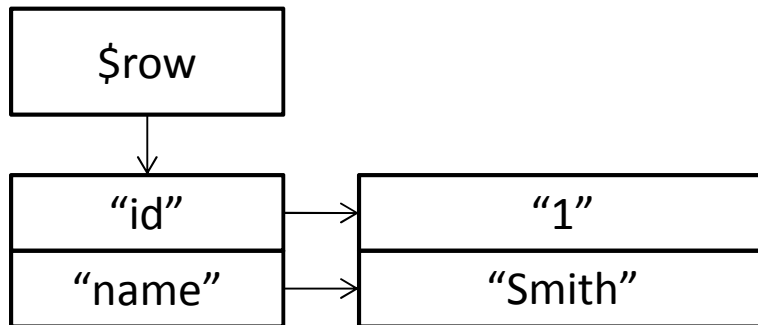
```
function
  factorial($val)
{
  return $val *
    factorial($val-1);
}
```

Important: During the coursework, write all functions in the coursework files which you will submit (you are unlikely to need functions though)

Associative and numeric arrays

Associative mysql_fetch

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



```
$result = mysql_query( "SELECT id, name FROM mytable" );
```

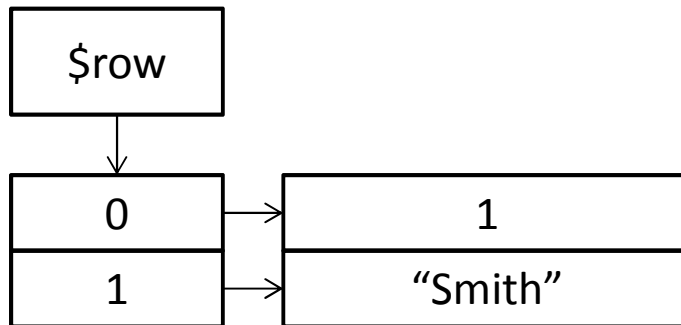
```
while ( $row = mysql_fetch_array( $result, MYSQL_ASSOC ) )  
{  
    echo "ID:" . $row["id"] . "Name:" . $row["name"];  
}
```

```
mysql_free_result($result);
```

Note: memory is freed anyway at end of script, so usually unnecessary

Numeric mysql_fetch

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



```
$result = mysql_query( "SELECT id, name FROM mytable" );
```

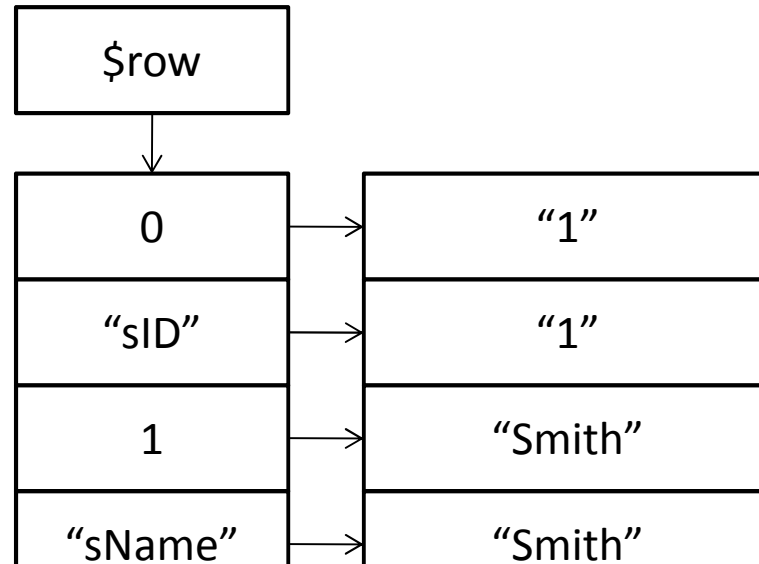
```
while ( $row = mysql_fetch_array( $result, MYSQL_NUM ) )  
{  
    echo "ID:" . $row[0] . "Name:" . $row[1];  
}
```

```
mysql_free_result($result);
```

Both

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

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



Other notes

- You can retrieve the array of keys for an associative array using the `array_keys()` function and then use the key to look up value
- If you set the key to be a numeric value, it will act as if the value was at that index in the array, allowing you to fake an array as both numeric and associative (see previous slide)

```
$arr2 = array( 6=>1, 5=>"Test", 4=>6, 3=>"Bye",  
              2=>3, 1=>4, 0=>5 );
```

```
$keys2 = array_keys( $arr2 );
```

```
for ( $i=0; $i < count($arr2) ; $i++)  
{  
    echo "Key " . $i . " is " . $keys2[$i] . " ";  
    echo "Value " . $i . " is " . $arr2[$i] . "<br>";  
}
```

Key 0 is 6	Value 0 is 5
Key 1 is 5	Value 1 is 4
Key 2 is 4	Value 2 is 3
Key 3 is 3	Value 3 is Bye
Key 4 is 2	Value 4 is 6
Key 5 is 1	Value 5 is Test
Key 6 is 0	Value 6 is 1

Integrating with forms

Simple HTML form code

- HTML form code, overview:

```
<form action= "index.php" method="get">
```

```
Username: <input type="text" name="user" />
```

```
<input type="submit" name="mysubmit"  
      value="Submit" />
```

```
</form>
```

Simple HTML form code

- HTML form code, overview:

Create a form

Action: what it does, e.g. open a page

<form **action="index.php"**

method="get">

How does it send params? "get" or "post".

Text - appears

Input field – type in it

Username: <input type="text" name="user" />

Submit button

**<input type="submit" name="mysubmit"
value="Submit" />**

Text to appear in the button.
Server would get "mysubmit=Submit"

</form>

\$_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'

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

\$_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
- **\$_REQUEST**
 - Holds all GET **and** POST values

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

Example 1

<http://severn.cs.nott.ac.uk/~jaa/dbs/htmlform.php>

<http://www.cs.nott.ac.uk/~jaa/dbs/htmlform.txt>

Demo 1: Set name

```
<?php
if ( !isset( $_GET['name'] ) )
{
    echo "No name has been given. Please choose one.";
    $message = "Please enter your name:";
}
else
{
    echo "Hello " . $_GET['name'];
    $message = "You can try another name now:";
}
echo '<form action=' . $_SERVER['PHP_SELF'] . ' method="get">';
echo $message;
echo '<input type="text" name="name" />';
echo '<input type = "submit" value="GO" />';
echo '</form>';
?>
```

The generated code

Hello Jason

```
<form action="/~jaa/dbs/htmlform.php"  
method="get">
```

You can try another name now:

```
<input name="name" type="text">  
<input value="GO" type="submit">  
  
</form>
```

Example 2

<http://severn.cs.nott.ac.uk/~jaa/dbs/dbdemo.php>

<http://www.cs.nott.ac.uk/~jaa/dbs/dbdemo.txt>

Demo 2: An outline of using a database

```
<head>
<title>Test Display</title>
<link rel="stylesheet" type="text/css" href="styles.css" />
</head>
<html>
<body>

<?php
    include_once('dbconnect.php');

    Include information here for updates, and/or selects
?>

    Include any pure HTML elements
</body>
</html>
```

Outline of the file

Start HTML code
Embedded PHP code
Connect to database
Code for SQL queries
End the PHP code
Any HTML forms?
End HTML

Create/drop tables

```
if ( isset( $_GET['create'] ) )
{
    echo "<H2>CREATE TABLE</H2>";

    mysql_query(
"CREATE TABLE test ( ID INT PRIMARY KEY, MyInt INT, MyString VARCHAR(255) )" );

    mysql_query(
"INSERT INTO test (ID,MyInt,MyString) VALUES (1,1,2)" );
}
```

Create
database
tables

Populate
database
tables

```
if ( isset( $_GET['drop'] ) )
{
    echo "<H2>DROP TABLE</H2>";
    mysql_query( "DROP TABLE test");
}
```

Drop the
database
tables

Update the database tables

```
if ( isset( $_GET['ival'] ) )
{
    echo "<H2>SET MyInt</H2>";
    $value = mysql_real_escape_string( $_GET['ival'] );
    $query = "UPDATE test SET MyInt = " . $value;
    mysql_query($query);
}
```

Code to update a table, building the update statement using the parameter passed in

← Set the MyInt column according to the ival parameter

```
if ( isset( $_GET['strval'] ) )
{
    echo "<H2>SET MyString</H2>";
    $value = mysql_real_escape_string( $_GET['strval'] );
    $query = "UPDATE test SET MyString = '" . $value . "'";
    mysql_query($query);
}
```

← Set the MyString column according to the strval parameter

Output database contents

```
echo "<H2>Current table contents</H2>";
$query = "SELECT ID, MyInt, MyString FROM test";
$result = mysql_query($query);
echo "<table> <tr class=\"tablehead\">";
echo "<td>ID</td>";
echo "<td>MyInt</td>";
echo "<td>MyString</td>";
echo "</tr>";
while ($row = mysql_fetch_array($result))
{
    echo "<tr>";
    echo "<td>" . $row['ID'] . "</td>";
    echo "<td>" . $row['MyInt'] . "</td>";
    echo "<td>" . $row['MyString'] . "</td>";
    echo "</tr>";
}
echo "</table>";
```

Add some form fields

?>

End PHP

<H2>Enter your values here:</H2>


<form action= "test.php" method="get">

Int value: <input type="text" name="ival" />

<input type="submit" name="ButtonPressed" value="SetMyInt" />

</form>

First form
Input an int



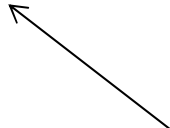
<form action= "test.php" method="get">

String value: <input type="text" name="strval" />

<input type="submit" name="ButtonPressed" value="SetMyString" />

</form>

Second form
Input a string



</body>

</html>

Next Lecture

- General Database Security
- Privileges
 - Granting
 - Revoking
- Views
- SQL Injection Attacks
- Further Reading
 - The Manga Guide to Databases, Chapter 5
 - Database Systems, Chapter 20