Programming the Internet—Lecture Notes

Week 6: PHP and MySQL: Database Connectivity

Introduction to MySQL

This week we are going to look at how we can retrieve information from a database. The database package we are going to use is MySQL (2012). It is an example of a relational Database Management System (DBMS). A relation is a table, and this indicates that data is stored in rows and columns in one or more tables. Columns represent fields in the database (e.g. name, address, phone number) and rows represent records (e.g. information about one particular person).

For the purposes of this module, the database is just a tool to help us illustrate how PHP can connect with, store and retrieve information. We shall not be going into detail about how relational databases are constructed. In our examples and in your assessments a single table containing a small amount of data will be sufficient for us to fulfil the learning outcomes.

MySQL is an open source product and is widely used. It can be downloaded to a desktop computer along with PHP and the Apache Web server by installing a free software package such as XAMPP (Apache Friends, 2012). MySQL is available on the Laureate server, and each project group will have a database created for them. You will then be able to create a table and insert records into it by connecting remotely to MySQL using some free software called MySQL Workbench. Instructions on how to download, set up and connect this software are included in the Week 2 Learning Resources.

The examples in these Lecture Notes will use a small database called *mobiles*, which is hosted on the Laureate server. Here is the data structure of the *devices* table within that database.

	#	Column	Type	Collation	Attributes	Null	Default
☐	1	refNo	int(4)		UNSIGNED ZEROFILL	No	None
☐	2	name	varchar(20)	utf8_unicode_ci		No	None
☐	3	deviceType	varchar(10)	utf8_unicode_ci		No	None
☐	4	colour	varchar(10)	utf8_unicode_ci		No	None
☐	5	stockLevel	int(4)			No	None
☐	6	salesThisMonth	int(4)			No	None
☐	7	customerRating	int(1)			No	None

Figure 1: The *devices* table structure

The *int* values are obviously integers and the number in brackets specifies the maximum size (how many digits they can have). The *varchar* type is used for

text, and the number in brackets shows the maximum number of characters allowed. Under *Attributes*, the reference to *UNSIGNED ZEROFILL* means that the reference number (refNo) will be a positive value (or zero) and leading zeros will be inserted to pad the content up to the maximum of 4 digits. Thus, 2 becomes 0002. The refNo field is the primary key, which means the value in this field cannot be duplicated. The *No* value under *Null* means, in effect, that an entry is required in each field. Here are the contents of the database:

Mobiles in Stock

refNo	name	deviceType	colour	stockLevel	salesThisMonth	customerRating
0001	iPhone	phone	black	25	6	4
0002	Android	phone	silver	40	15	4
0003	Blackberry	phone	black	20	6	4
0004	Nokia	phone	pink	10	4	3
0005	Windows Phone	phone	red	10	2	3
0006	iPad	tablet	white	30	15	5
0007	Blackberry Playbook	tablet	black	10	2	3
0008	Kindle Fire	tablet	black	30	12	4
0009	Samsung Galaxy	tablet	gold	20	10	4
0010	Google Nexus	tablet	black	30	20	5

Figure 2: The *device*s table contents

We shall have to live with the disappointment of knowing that each mobile device is only available in one colour, and that only a no brand Android phone is in stock. It does, however, simplify our task. In the above table, it is expected that customer ratings will be on a scale of 1 to 5.

Those familiar with Microsoft Access will be accustomed to thinking of a database as a file. MySQL databases are not files; instead, just think of them as logical 'entities' that are available within the MySQL software. Fortunately, this does not mean that databases cannot be moved from one copy of MySQL to another (e.g. between one on the Laureate server and another on your computer at home). MySQL has an Export and Import function.

The simplest way of exporting the tables within a database is to select the Export command and then ask the database software to create a .sql file. This will contain Structured Query Language (SQL) commands to recreate a table structure and insert records on another copy of MySQL (using the Import command). When you are importing, create an empty database in MySQL (we shall give you details of how to do this elsewhere in Blackboard) and then select the Import command within that empty database. This will allow you to point at the .sql file you have exported. In the Code Examples zip file for Week 6, there is a .sql file which will enable you to recreate the *devices* table shown above on a version of MySQL running on your computer.

This week we will not cover SQL in detail, but you can learn more about it in the articles listed in the References section of these Lecture Notes (W3 Schools, 2012a; Tizag, 2008; IT Business Edge, 2012). Instead, this week we shall look at and introduce a few queries into our sample programs that show how SQL can be used to query relational databases. Remember that SQL can be used on most databases, and a little knowledge will go a long way.

Figure 2 above is actually a screenshot of the output from a PHP script and an SQL query. Looking at a program that lists the entire contents of a database will therefore be a good place to start. To demonstrate some of the things we can do with PHP in connecting to a database, take a look at this example hosted on the Laureate server: http://laureatestudentserver.com/faculty1/main-inc.html. Use this page to see how the examples in this week's Lecture Notes function. Here is what it looks like:

Figure 3: The *mobiles* database main access page

That's enough by way of explanation. Let's get straight to the PHP code in Figure 4 below that lists the database's contents when we click on the 'List all the devices' link on the Mobiles Database main page above (Figure 3). Don't forget to read the code comments. They explain what is going on.

```
<!DOCTYPE html>
<html>
<head>
<title> List of all the mobile devices </title>
<meta charset="utf-8" />
<link href="allDevices.css" type="text/css" rel="stylesheet" />
</head>
<body>
<p>
<h2>List of Mobile Devices in the Database</h2>
</p>
<?php
$dsn =
'mysql:host=laureatestudentserver.com;dbname=laureate_mobiles';
/* The data source name consists of the name of the server hosting
MySQL plus the name of the database */
$username = 'laureate_xxxx';
$password = 'yyyyy';
/* A database, username and password will be allocated to each
project group */
```

```
$dbc = new PDO($dsn, $username, $password);
/* Create a new database connection with the data source name,
username and password */
/* PDO = PHP Database Objects. We are creating a new instance of the
connection object */
$query = 'SELECT * FROM devices ORDER BY refNo';
/* The SQL. This is the command that will extract data from the
database */
/* It says "Select everything (*) from the table called devices and
place it in (ascending) reference number order" */
$results = $dbc->query($query);
/* Executes the SQL query stored in the variable $query, and saves
all the data returned in $results */
$rows = $results->rowCount();
/* Establishes how many rows of data have been extracted */
?>
<table>
<caption>Mobiles in Stock</caption>
<thead>
<tr>
<th>refNo</th>
<th>name</th>
<th>deviceType</th>
<th>colour</th>
<th>stockLevel</th>
<th>salesThisMonth</th>
<th>customerRating</th>
</tr>
</thead>
<?php
if ($rows == 0)
/* If no rows of data (no records) have been found */
{
echo("<p> There are currently no records in the devices table</p>");
echo("<p><a href='main.html'>Continue</a></p>");
}
else {
foreach ($results as $devices) :
/* $results contains every row of data (every record) */
/* Each record is being temporarily stored in the variable $devices
while the foreach loop works its way through every record in $results
*/
?>
<tr>
<td class = "left">
<?php echo $devices['refNo']; ?>
</td>
<td class = "left">
<?php echo $devices['name']; ?>
</td>
<td class = "left">
<?php echo $devices['deviceType']; ?>
</td>
<td class = "left">
<?php echo $devices['colour']; ?>
</td>
<td class = "right">
<?php echo $devices['stockLevel']; ?>
</td>
<td class = "right">
```

```
<?php echo $devices['salesThisMonth']; ?>
</td>
<td class = "right">
<?php echo $devices['customerRating']; ?>
</td>
</tr>
<?php endforeach;
}
$dbc = null;
/* Close the database connection */
?>
</table>
<?php echo("<p><a href='main.html'>Continue</a></p>"); ?>
</body>
</html>
```

Figure 4: allDevices.php

There is more than one method of using PHP to extract information from a database. Many examples use the PHP *mysqli* extension (*i* is for improved) or the older *mysql* extension. This module uses PDO (PHP Data Objects). There is inbuilt support for PDO from PHP 5.1 onwards (PHP Group, 2012, Boronczyk, 2011). The PHP mysqli extension can only be used with the MySQL database. The advantage of using the object-oriented PDO approach is that its techniques can be applied with only minor changes to PHP programs accessing other major databases such as Oracle, DB2, Microsoft SQL Server and PostgreSQL. It therefore provides students with a more transferable skill.

In the example, there is also a CSS file which helps with the formatting of the HTML table we saw in Figure 2, which is the table that is displayed when you select ‘List all the devices’ on the Mobiles Database main page (Figure 3):

```
table {
border: 2px solid blue;
}

caption {
font-weight: bold;
}

th {
font-weight: bold;
text-align: center;
border: 1px solid blue;
}

td {
border: 1px solid blue;
}

td.left {
text-align: left;
}

td.right {
```

```
text-align: right;
}
```

Figure 5: allDevices.css

Here we have defined two classes for the `<td>` HTML tag. This will allow us to show text as left-aligned and numbers as right-aligned within different table cells.

Database Connections

Before we go on to look at another example of how we can access a database using a PHP script, let us pause for a moment and look at the way we are making a connection. Consider this code:

```
$dsn =
'mysql:host=laureatestudentserver.com;dbname=laureate_mobiles';
/* The data source name consists of the name of the server hosting
MySQL plus the name of the database */
$username = 'laureate_xxxx';
$password = 'yyyyy';
/* A database, username and password will be allocated to each
project group */
$dbc = new PDO($dsn, $username, $password);
```

This is the code that makes the initial connection to the database before we go on to use SQL in order to query it in some way. It involves passing a username and password to the MySQL server. Although users will normally see only the output from the PHP script and will not be able to read the source code, in a commercial environment it would be considered less than ideal to include sensitive information like database usernames and passwords in files that are potentially visible to hackers over the Internet.

One solution would be to place this connection code in a PHP *include* file which would be located in a directory that has not been made publicly visible by being nominated as some Web server's home directory or one of its sub-directories. Suppose that we move this sensitive code to a file called *connection.php* . In our main file, instead of the code above, we might then find ourselves using something like this:

```
include("../../connection.php");
```

The word *include* tells the PHP script that it should paste (include) the contents of the file *connection.php* at this point in the script when *allDevices.php* is executed. The two pairs of dots tell us that the file *connection.php* is two directory levels above the current directory on the machine that is hosting the Web server.

Remember that only the Web server's home directory and its sub-directories are publicly visible on the Internet. Anything above that level in the directory structure on the computer hosting the Web server will not be publicly visible. Let us assume that *connection.php* has been located in a directory at a level

above the Web server's home directory and is, thus, in a relatively safe place. The PHP script has privileged access to the file to be included but hackers should not.

Now whether we can place the connection details in such a safe place depends on the permissions we have on the server. If we control the server, then it should be possible to access and utilise a file on a directory that is not publicly visible. However, you will not have such permissions on the Laureate Web server and so the slightly less secure method of providing connection details within your main script will be necessary when you are developing PHP programs for your Group Project pages.

Your Hand-in Assignment this week involves writing some additional code to access our *mobiles* database. In the Code Examples files for this week, you will see that there are xxxx and yyyy where the username and password should be in the connection code. This is to prevent you from having administrative login information which would allow you greater control over the database. Although we are happy to let you access our database using PHP scripts, we don't really want to give you the username and password that would give you greater powers over that database. Don't worry, you will get a username and password for your own group database, and you can play with it to your heart's content. We don't necessarily want you to do the same with the *mobiles* database.

So how have we coped with this? We have, of course, used an include file to hide the details from you. Instead, you should utilise an *include* statement to connect to the database pointing at the information in the *include* file *connection.php*. When you are doing your Hand-in Assignment this week, you will have to use an *include* as well. Where you see the following code in the Code Examples files we have given you (and this time we have stripped out the comments):

```
$dsn =
'mysql:host=laureatestudentserver.com;dbname=laureate_mobiles';
$username = 'laureate_xxxx';
$password = 'yyyyy';
$dbc = new PDO($dsn, $username, $password);
```

We want you to substitute this code:

```
include("../../connection.php");
```

You will just have to trust us when we say that the right username and password are in that file called *connection.php* .

Just to underline that, we want you to do different things in different situations:

1. In the Week 6 Assignment, use the *include* statement instead of the long version of the connection code so that we can keep our *mobiles* database a little more secure.

2. In the Group Project work (due in Week 7), don't use an include (because you don't have the permissions to put an include file in a safe place). Use the long version of the connection code with the username and password we provided for your own database.

Viewing a Single Record in MySQL

Let's look at another program that accesses a MySQL database. The second link on the Mobiles Database main page (Figure 3) allows the user to view a single record in the database. This requires some user input, so there are both HTML and a PHP files. Here is the HTML code:

```
<!DOCTYPE html>
<html>
<head>
<title>Single Device Selection</title>
<meta charset="utf-8" />
</head>
<body>
<h2>Select a Single Device</h2>
<form id="selectForm" action="singleDevice.php" method="POST">
<p>Device Reference Number (RefNo) <input type="text" name="refNo"
size = "4" maxlength = "4"></p>
<p><input type=submit value="Submit"></p>
</form>
</body>
</html>
```

Figure 7: singleDevice.html

Here we are simply inviting the user to input the reference number of the record he or she wants to view. Figure 2 should remind you of those reference numbers that are available. Comments have been included in the following PHP code only where something new has been introduced.

```
<!DOCTYPE html>
<html>
<head>
<title> Searching for a Single Record </title>
<meta charset="utf-8" />
<link href="allDevices.css" type="text/css" rel="stylesheet" />
</head>
<body>

<p>
<h2>Device Search Result</h2>
</p>

<?php
$refNo = $_POST['refNo'];
$dsn =
'mysql:host=laureatestudentserver.com;dbname=laureate_mobiles';
$username = 'laureate_xxxx';
$password = 'yyyyy';
$dbc = new PDO($dsn, $username, $password);

$query = "SELECT * FROM devices WHERE refNo ='" . $refNo . "'";
```

```
/* The SQL says "Select everything from the devices table where the
reference number equals that which the user typed into the previous
HTML form. As the refNo is unique, there will be at most one record
*/
$results = $dbc->query($query);
$rows = $results->rowCount(); ?>


<?php
if ($rows == 0) {
echo("<p> There is no device with that reference number in the
devices table");
}
else {
$selectedDevice = $results->fetch();
/* Fetch a record from the $results variable and store it in
$selectedDevice. There will be a maximum of one record */
?>
<table border>
<caption>The Device You Chose </caption>
<thead>
<tr>
<th>refNo</th>
<th>name</th>
<th>deviceType</th>
<th>colour</th>
<th>stockLevel</th>
<th>salesThisMonth</th>
<th>customerRating</th>
</tr>
</thead>
<tr>
<td class = "left">
<?php echo $selectedDevice['refNo']; ?>
</td>
<td class = "left">
<?php echo $selectedDevice['name']; ?>
</td>
<td class = "left">
<?php echo $selectedDevice['deviceType']; ?>
</td>
<td class = "left">
<?php echo $selectedDevice['colour']; ?>
</td>
<td class = "right">
<?php echo $selectedDevice['stockLevel']; ?>
</td>
<td class = "right">
<?php echo $selectedDevice['salesThisMonth']; ?>
</td>
<td class = "right">
<?php echo $selectedDevice['customerRating']; ?>
</td>
</tr>
<?php
} // End of else
$dbc = null;
?>
</table>
<?php
```

```
echo("<p><a href='main.html'>Continue</a></p>");
?>
</body>
</html>
```

Figure 8: singleDevice.php

It is worth commenting on the *if* statement in the code above. As programmers we want to return a simple message to the user if there is no device in the database with the reference number typed into the HTML form. If there is a record with that reference number (the *else* condition is true), we want to display the details in an HTML table with just one row of data.

Within the *else* part of this *if* statement, we see a mixture of PHP and HTML. Previously, we have stated that HTML code is just automatically sent to the user. This is an exception. PHP is clever enough to realise that HTML code within a PHP *if* statement should only be sent to the user when the relevant condition is found to be true. Thus, all those HTML `<table>` tags will only be sent where the PHP *else* condition is met. This applies to any HTML within the braces (curly brackets) after the word *else* in the PHP code.

Updating a Record

We are now ready to look at a more complex example which involves updating (amending) a record. In this example, the user will input a reference number,. Our first PHP script, *updateDevice1.php*, will check to see if such a record exists. If it does, the program will display the current contents of that record in editable HTML text boxes using this code:

```
value="<?php echo($colour); ?>">
```

The user will amend the relevant fields and submit them to the Web server. A second PHP script, *updateDevice2.php*, will then update the database and report the outcome. Here are the example files:

```
<!DOCTYPE html>
<html>
<head>
<title>Update a Single Device</title>
<meta charset="utf-8" />
</head>
<body>
<h2>Device Update Form</h2>
<form id="updateForm" action="updateDevice1.php" method="POST">
<p>Reference Number <input type="text" name="refNo" size="4"
maxlength = "4"></p>
<p><input type=submit value="Check that the Device Exists"></p>
</form>
</body>
</html>
```

Figure 9: updateDevice.html

```
<?php
$refNo = $_POST['refNo'];
$dsn =
'mysql:host=laureatestudentserver.com;dbname=laureate_mobiles';
$username = 'laureate_xxxx';
$password = 'yyyyy';
$dbc = new PDO($dsn, $username, $password);

$query = "SELECT * FROM devices WHERE refNo = '$refNo'";

$results = $dbc->query($query);
$rows = $results->rowCount();

if ($rows == 0) {
      echo("<p> There is no device with the reference number " .
$refNo . " in the devices table</p>");
      echo("<p><a href='main.html'>Continue</a>");
   }
else {
$deviceToUpdate = $results->fetch();
$name = $deviceToUpdate['name'];
$deviceType = $deviceToUpdate['deviceType'];
$colour = $deviceToUpdate['colour'];
$stockLevel = $deviceToUpdate['stockLevel'];
$salesThisMonth = $deviceToUpdate['salesThisMonth'];
$customerRating = $deviceToUpdate['customerRating'];
?>

<!DOCTYPE html>
<html>
<head>
<title> Update a Single Device's Details  </title>
<meta charset="utf-8" />
</head>
<body>

<p>
<h2>Device Update Form</h2>
</p>

<form name="updateForm" action="updateDevice2.php"method="POST">

<p>Reference Number <?php echo ($refNo) ?></p>

Name <input type="text" name="name" size="20 "maxlength = "20"
value="<?php echo($name); ?>">
Device Type <input type="text" name="deviceType" size="10" maxlength
= "10" value="<?php echo($deviceType); ?>">
Colour <input type="text" name="colour" size="10" maxlength = "10"
value="<?php echo($colour); ?>">
Stock Level <input type="text" name="stockLevel" size="4" maxlength =
"4" value="<?php echo($stockLevel); ?>">
Sales This Month <input type="text" name="salesThisMonth" size="4"
maxlength = "4" value="<?php echo($salesThisMonth); ?>">
Customer Rating <input type="text" name="customerRating" size="1"
maxlength = "1" value="<?php echo($customerRating); ?>">
<input type="hidden" name="refNo" value="<?php echo($refNo); ?>">
<input type=submit value="Submit Update">
</form>
</body>
```

```
</html>
<?php
}
$dbc = null;
?>
```

Figure 10: updateDevice1.php

Note that in the above example we have used a hidden field like this:

```
<input type="hidden" name="refNo" value="<?php echo($refNo); ?>">
```

We do not want to let the user change the reference number, as that is the primary key for accessing the database. We therefore retrieve the reference number from the HTML form in *updateDevice.html* and store it in the PHP variable, *$refNo*. We then insert that variable in a hidden field which, as its name suggests, is not displayed on screen. However, the contents of the hidden field are returned to *updateDevice2.php* in the same way as other HTML form fields. This value is then used by *updateDevice2.php* in the SQL statement to identify the record which needs to be updated.

```
<?php
$refNo = $_POST['refNo'];
$name = $_POST['name'];
$deviceType = $_POST['deviceType'];
$colour = $_POST['colour'];
$stockLevel = $_POST['stockLevel'];
$salesThisMonth = $_POST['salesThisMonth'];
$customerRating = $_POST['customerRating'];
/* The above code retrieves data from the HTML form contained in
updateDevice1.php */
$dsn =
'mysql:host=laureatestudentserver.com;dbname=laureate_mobiles';
$username = 'laureate_xxxx';
$password = 'yyyyy';
$dbc = new PDO($dsn, $username, $password);

$sqlQuery = "UPDATE devices SET name = '$name', deviceType =
'$deviceType', colour = '$colour', stockLevel = $stockLevel,
salesThisMonth = $salesThisMonth, customerRating = $customerRating
WHERE refNo = '$refNo'";
/* The SQL says "Update the devices table by setting each field to
the contents of the HTML form in updateDevice1.php". Only where the
user changed something will any difference be seen */
$update_result = $dbc->exec($sqlQuery);
/* Execute the SQL query and store the number of rows that were
changed in $update_result */
$dbc = null;

if($update_result != 0)
/* If at least one row in the database was changed */
{
echo("<h3>The device record was updated successfully</h3>");
}
else
{
echo("<h3>The update was unsuccessful</h3>");
```

```
}
echo("<p><a href='main.html'>Continue</a></p>");
?>
```

Figure 11: updateDevice2.php

PHP and Text Files

Sometimes we shall not need anything as sophisticated as a database to store data and a simple text file will suffice. PHP has a range of built-in functions for reading from and writing to text files (W3 Schools, 2012b). Firstly we are likely to want to open a text file:

```
$file = fopen ("names_file.txt", "a+");
```

The built in PHP function, *fopen()*, does the work for us. It accepts two parameters. The first is the name of the file we want to open. PHP will, in this case, expect the text file, *names_file.txt,* to be in the same directory as the PHP program. This parameter can also be a variable name such as *$fileName*, if we have previously given it a value like this:

```
$fileName = "names_file.txt";
```

The second parameter specifies the mode we want to use in opening the file on the Web server. In this example, we have used a very safe parameter, *a+*, which means we can both read from the file and append it (add data to the end of the file). If the file does not exist, it will be created.

Note that other parameters that permit writing to a file, such as *w* and *w+*, will start writing at the beginning of the file, and existing data will be overwritten and lost. Simply opening a file in *w* mode will erase its previous contents, so use it cautiously. That is why *a+* is a safe option. Sometimes programmers prefer to perform the read and write operations separately. In such cases it is possible to open a file with the parameter, *r*, which is read only, close it and then open it again in the mode *a*, which is append only. When the programmer specifically wants to create a new file and be alerted by an error message if a file of that name already exists, the mode *x* should be used for write only or *x+* for read and write.

In our example, when the file has been opened, the file can from that point onwards be referred to as *$file* because of the assignment statement we have employed in using the *fopen()* function. The variable *$file* is an example of a file 'handle'. Thus, when we have finished working on our file (or want to open it in a different mode), we shall write:

```
fclose($file);
```

Often, we shall want to search through a text file, line by line. This can be achieved as follows:

```
$file = fopen("names_file.txt", "r");
while(!feof($file))
  {
  echo fgets($file). "<br />";
  }
fclose($file);
```

This piece of code opens our file in read only mode (*r*) and then says that while we have not reached the end of the file (detected by *feof*), we should read a line of text (*fgets*) and output it to the user as HTML (using *echo*). A `<br />` tag is used to print each line in the file on a new line in the user's browser. It is good programming practice to always close files when we have finished with them, although, if we forget, PHP will do this for us when a program ends.

Writing to a File

Writing to a file can be performed using the built in PHP function, *fwrite()*, like this:

```
$file = fopen("names_file.txt", "a");
fwrite($file, $name . "\r\n");
fclose ($file);
```

The function *fwrite()* accepts two parameters. The first is the file handle that indicates the file to which we want to write, and the second is the data we want to write, usually stored in a variable. In this example, we shall assume that the variable *$name* contains data that has been returned from a textbox in an HTML form, and we want to store this information on the server. Note that in this example we have opened the file in append mode so as not to overwrite any data already in the file.

The data to be written here is *$name . "\r\n"*. This code will write the contents of the variable *$name* to the file and then insert a carriage return (\r) and line feed (\n) to ensure the next write operation takes place on a new line in the text file. Because different systems use different methods of taking a new line, both \r and \n are needed to ensure cross-system compatibility. Escape characters such as \n must be enclosed in quotation marks. They are effectively part of the text that is being written, so our second parameter might have been:

```
"Pauline Jones\r\n"
```

The symbol \t represents a horizontal tab. If a backslash needs to be written to a file, it must be represented in an *fwrite()* statement as two backslashes (\\). Similarly if double quotes need to be written to a file they must be represented as \" . We say that those characters have to be 'escaped'.

Remember that \r\n will only allow us to take a new line in writing to a text file. If we are sending HTML back to a browser using *echo*, the browser will simply reproduce the characters "\r\n" and will not take a new line. Output in the

browser will only appear on a new line in response to HTML tags such a *
*.

A full list of things that can be done with files will be found at W3 Schools (2010a).

To conclude this section on text files, here is a simple working example. The code is included in the Code Examples zip file in this week's Learning Resources. The user is asked to type the name of a colour in an HTML form text box. The PHP script launched on the Web server searches a text file and sends back a message confirming whether that colour is listed in the file. Here is the HTML:

```
<!DOCTYPE html>
<html>
<head>
<title>Is the Colour in the File ?</title>
<meta charset="utf-8" />
</head>
<body>
<h1> Guess Whether the Colour is in the File </h1>

<form method = "post" action = "colours.php">
<fieldset>
<legend> The file contains 8 of the 16 standard colour names defined
by the W3C. </legend>
<label for="colour">Please type the name of a colour.</label><br />
<input name = "colour" id = "colour" type = "text" size = "10"
maxlength = "15" /><br />
</fieldset>
<p>
<input type = "submit" name = "submit" value = "Submit" />
<input type = "reset" value = "Clear" />
</p>
</form>
</body>
</html>
```

Figure 12: colours.html

Here is the PHP file that processes the information sent from the HTML form:

```
<!DOCTYPE html>
<html>
<head>
<title>Is the Colour in the File ? </title>
<meta charset="utf-8" />
</head>

<h1>Guessing the Colour</h1>
<?php
$colour = $_POST["colour"];
$filecolour = "";
$colourfound = false;
$colour = trim($colour); // Strip out spaces, carriage returns, line
feeds, tabs from the beginning and end of the string
```

```
$colour = strtoupper($colour);  //Name of colour chosen by the user
is converted to upper case
$file = fopen ("colours.txt", "r");

while(!feof($file))
{
$filecolour = fgets($file);
$filecolour = trim($filecolour); // Strip out spaces, carriage
returns, line feeds, tabs from the beginning and end of the string
$filecolour = strtoupper($filecolour);  // Name of colour in the file
is converted to upper case
if ($colour == $filecolour)   // Comparison is now not case sensitive
{
$colourfound = true;
}  // End of if statement
}  // End of while loop
if ($colourfound)
{
echo ("The colour " . $colour . " has been found in the file.");
}
else
{
echo ("The colour " . $colour . " is not in the file. Please guess
again.");
} // End of if statement
echo ("<p><a href='colours.html'>Continue</a></p>");
fclose($file);
?>
</body>
</html>
```

Figure 13: colours.php

A text file which contains the list of eight colours, *colours.txt*, is also in the Code Examples zip file. There are a couple of programming techniques worth noting. Firstly the use of the PHP *trim()* function to strip spaces, carriage returns, line feeds and tabs from the beginning and end of a string. It is worth using this function regularly because PHP is likely to read a carriage return which follows a word in a text file as being part of the string that is being compared to another string. We can therefore often get unexpected results when we believe two strings are the same.

Secondly, PHP does have a *strcasecmp()* function that ignores the case of the letters in a string (whether they are upper or lower case) when deciding whether two strings are equal to each other. This is useful where the case is not important and we are not confident of the user typing input in the case that we were expecting. We have achieved the same case insensitive comparison of strings in a slightly different way here, because it is a technique that will be of use in other programming languages you may come across.

In our example, we have converted both the strings we are going to compare (from the input and from the file) into upper case (capital letters) before we make the comparison. We achieve this with the lines of code:

```
$colour = strtoupper($colour);
```

```
$filecolour = strtoupper($filecolour);
```

We can then say:

```
if ($colour == $filecolour)
```

References

Apache Friends (2012) *XAMPP* [Online]. Available from http://www.apachefriends.org/en/xampp.html (Accessed: 17 September 2012).

Boronczyk, T. (2011) *Migrate from the MySQL extension to PDO* [Online]. Available from http://phpmaster.com/migrate-from-the-mysql-extension-to-pdo/ (Accessed: 17 September 2012).

IT Business Edge (2012) *What is SQL?* [Online]. Available from http://www.sqlcourse.com/intro.html (Accessed: 17 September 2012).

Murach, J. & Harris, R. (2010) *Murach's PHP and MySQL*. Fresno, California: Mike Murach & Associates.

MySQL (2012) *MySQL: the world's most popular open source database* [Online]. Available from http://www.mysql.com/ (Accessed: 17 September 2012).

PHP Group (2012) *PHP data objects* [Online]. Available from http://php.net/manual/en/book.pdo.php (Accessed: 17 September 2012).

Tizag.com (2008) *SQL fundamentals* [Online]. Available from http://www.tizag.com/sqlTutorial/ (Accessed: 17 September 2012).

Tizag.com (2008) *PHP tutorial: for each* [Online]. Available from http://www.tizag.com/phpT/foreach.php (Accessed: 17 September 2012).

Valade, J. (2010) *PHP and MySQL for dummies.* Hoboken, NJ: Wiley Publishing.

W3 Schools (2012a) *Introduction to SQL* [Online]. Available from http://www.w3schools.com/sql/sql_intro.asp (Accessed: 17 September 2010).

W3 Schools (2012b) *PHP filesystem functions* [Online]. Available from http://www.w3schools.com/php/php_ref_filesystem.asp (Accessed: 17 September 2012).

W3 Schools (2012c) *PHP error handling* [Online]. Available from http://www.w3schools.com/php/php_error.asp (Accessed: 17 September 2012).