G64DBS EXERCISE 4: PHP, MYSQL AND HTML

INTRODUCTION

During this exercise we will cover how to use PHP to produce dynamic web pages based on our database. SQL is great for declarative queries using a DBMS, but for outputting useable, formatted documents, it falls short. Instead of trying to adapt SQL to improve the output, we can use PHP to retrieve our database results, and convert them into good looking HTML. You might wonder why we need to learn HTML and PHP in a module on databases. Both HTML and PHP are extremely useful skills to learn, and learning them here will also help us understand how databases fit into websites. Almost all big websites feature a back-end database that runs behind the scenes, feeding information to servers that generate dynamic HTML documents. This exercise will be a brief introduction into this.

For this exercise, we're going to extend our database tables from Exercise 3 a bit. We're going to add a "Rating" column to the CD that represents review scores these CDs might receive. We'll also add a "Year" column that represents the release date for the album. Finally, we'll add a "Nationality" column to the Artist table. If you wish to, you can make all these additions for yourself. However, to make things easier I've created a setupex4.sql file that will do it all for you if you'd prefer. In either case, your tables should look like this:

```
mysql> SELECT * FROM Artist;
+-------+---------------------------------+----------------+
| artID | artName                         | artNationality |
+-------+---------------------------------+----------------+
|     1 | Muse                            | British        |
|     2 | Mr. Scruff                      | British        |
|     3 | DeadMau5                        | Canadian       |
|     4 | Mark Ronson                     | British        |
|     5 | Mark Ronson & The Business Intl | British        |
|     6 | Animal Collective               | American       |
|     7 | Kings of Leon                   | American       |
|     8 | Maroon 5                        | American       |
+-------+---------------------------------+----------------+
8 rows in set (0.00 sec)

mysql> SELECT * FROM CD;
+------+-------+-----------------------------+---------+---------------+----------+---------+
| cdID | artID | cdTitle                     | cdPrice | cdGenre       | cdRating | cdYear |
+------+-------+-----------------------------+---------+---------------+----------+---------+
|    1 |     1 | Black Holes and Revelations |    9.99 | Rock          |       78 |   2006 |
|    2 |     1 | The Resistance              |   11.99 | Rock          |       90 |   2009 |
|    3 |     2 | Ninja Tuna                  |    9.99 | Electronica   |       55 |   2008 |
|    4 |     3 | For Lack of a Better Name   |    9.99 | Electro House |       38 |   2009 |
|    5 |     4 | Version                     |   11.99 | Rock          |       77 |   2007 |
|    6 |     5 | Record Collection           |   12.99 | Pop           |       22 |   2010 |
|    7 |     6 | Merriweather Post Pavilion  |   12.99 | Electronica   |       82 |   2009 |
|    8 |     7 | Only By The Night           |    9.99 | Rock          |       67 |   2008 |
|    9 |     7 | Come Around Sundown         |   12.99 | Rock          |       31 |   2010 |
|   10 |     8 | Hands All Over              |   11.99 | Pop           |       64 |   2010 |
+------+-------+-----------------------------+---------+---------------+----------+---------+
10 rows in set (0.00 sec)
```

SETTING UP PHP

Before you can begin writing PHP scripts you need to create a public_html folder inside your home directory. Begin by starting exceed, and make sure you're in your home directory. When you log onto avon.cs.nott.ac.uk, you should see the following command line:

```
username@avon:~>
```

The `:~>` tells us you're in your home directory, which is represented as a `~` in linux. If you see other directories, you can change to your home directory like this:

```
username@avon:~/solaris/Private> cd ~
username@avon:~>
```

As you can see, we were in the directory `~/solaris/Private`, and after we use the command `'cd ~'`, we move back into our home directory.

Now you are in your home directory, we need to create the necessary php and other files ready for your web page. To make this process easier, we will download a compressed website from elsewhere, and extract it into your home directory. The steps we will follow are shown below:

1. Ensure you are logged into avon from exceed, and you are in your home directory.
2. Download the required file from the school servers using the following command:

   ```
   wget http://cs.nott.ac.uk/~mpp/files/exercise4.tar.gz
   ```

   This command downloads the file straight to your home directory. This file contains all the files we need to start a website with PHP.

3. The file is a tar archive, that has also been zipped. When you extract it, it will create the complete website you need to get started, with all the correct file permissions. Use the following command to do this:

   tar -pxzvf exercise4.tar.gz

   This command will extract all the files we need. tar is the program we run, -pxzvf are the optional flags.

   - -p: Instructs the program to preserve all our file permissions. This is extremely important
   - -x: Instructs the program to extract all the files in the archive
   - -z: Instructs the program to unzip all the files as well, because they have been compressed

- -v: Instructs the program to operate in verbose mode, which increases the information it shows us about the process
- -f: Tells the program we intend to supply a file name for the archive we are extracting, in this case it's exercise4.tar.gz

4. Now we have downloaded and extracted the file, we should be able to view our test website. If you open your web browser and go to http://avon.cs.nott.ac.uk/~username/exercise4/index.php you should see a message telling you your website has been setup successfully!

The entire process above should look like this on your Exceed terminal (highlighted in bold are the commands you use):

```
user@avon:~> wget http://www.cs.nott.ac.uk/~mpp/files/exercise4.tar.gz
--2010-10-23 22:30:55--  http://www.cs.nott.ac.uk/~mpp/files/exercise4.tar.gz
Resolving www.cs.nott.ac.uk... 128.243.21.19, 128.243.20.9
Connecting to www.cs.nott.ac.uk|128.243.21.19|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3466 (3.4K) [application/x-gzip]
Saving to: `exercise4.tar.gz'

100%[======================================>] 3,466       --.-K/s   in 0s

2010-10-23 22:30:55 (288 MB/s) - `exercise4.tar.gz' saved [3466/3466]

user@avon:~> tar -pxzvf exercise4.tar.gz
public_html/
public_html/exercise4/dbconnect.php
public_html/exercise4/styles.css
public_html/exercise4/images/
public_html/exercise4/images/blend.png
public_html/exercise4/images/flags/
public_html/exercise4/images/flags/flag1.gif
public_html/exercise4/images/flags/flag2.gif
public_html/exercise4/images/flags/flag3.gif
public_html/exercise4/index.php
public_html/exercise4/functions.php
user@avon:~>
```

EDITING THE WEB PAGE

If everything has worked correctly, you should now have a working website. Next you will want to open up your favourite text editor (I recommend Notepad++ which is installed on all the lab machines). Once you have an editor running you need to open the following files:

- `H:/public_html/exercise4/index.php`
- `H:/public_html/exercise4/dbconnect.php`

There are other files in the directory, such as *functions.php* and *styles.css*. We won't be editing these in this exercise. We will be using them, so feel free to open them and have a look at how they work. *functions.php* is a file containing some useful functions. *styles.css* is a

cascading style sheet, this tells your browser how to render your website. For example, properties like background colour and font size are stored in this file.

Index.php is the main page of the website. Currently it only contains the bare minimum required for a webpage. That is, a pair of <html> tags, around a pair of <body> tags, and finally the text you saw in your browser. First we'll add a header to the HTML document so that it reads the style sheet, then we'll begin adding some content.

HTML HEADERS

The HTML <head> tag goes between the <html> tags, but before the <body> tags. We will add a link to the styles.css file, as well as a title for our webpage. Begin by creating a pair of <head></head> tags before the <body> start tag. If you do this correctly, your index.php file should look like this:

```
<html>
<head></head>
<body>
Well done, you've successfully created your web page!
</body>
</html>
```

Remember, all tags in HTML need to open and close. You can tell which tags are close tags because they contain a '/' character. Between your head tags, add a <title></title> pair, that also contain the text you would like for your web page title. For example:

```
<title>G64DBS Lab</title>
```

Next, add the following line inside the <head> tags too:

```
<link rel="stylesheet" type="text/css" href="styles.css" />
```

This tells the browser that your HTML gets it's formatting information from *styles.css*. If you've done this correctly, your website text will be centered, and your *index.php* will look like this:

```
<html>

  <head>
    <title> G64DBS Lab </title>
    <link rel="stylesheet" type="text/css" href="styles.css" />
  </head>

  <body>

    Well done, you've successfully created your web page!

  </body>
</html>
```

It's useful to indent each pair of tags that appear inside others, much like you might with SQL or PHP code. It makes it easier to read, and spot any errors.

CREATING A WEBPAGE

By this point you have a working php file, that currently only contains HTML text. Remember that a PHP document outputs HTML, so in this case we have simply written the HTML directly into the file. A PHP script will usually contain a combination of directly typed HTML, and HTML that is output by PHP code.

Next we will focus on creating some content for the web page. From now onwards all PHP code blocks and HTML text will be written between these two body tags. That is:

```
<body>
      All web page content will go between these two tags in the
      index.php file.
</body>
```

Begin by deleting any text that appears between the two body tags. Then write in some text of your own. Save your file, and visit the page in your browser. When you refresh the browser, you should see that the text has been updated to reflect your changes. From now on, each time you add some content you might like to revisit the page in your browser, to make sure everything is working.

Clear any text that is in your body region, and begin by adding a page Heading. This can be achieved by adding some text into the body, between two <h1></h1> tags. Below this, add your username between <h2></h2> tags. Your <body> region should look like this:

```
<body>
      <h1>Database Systems</h1>
      <h2>Username: user</h2>

</body>
```

CREATING AN HTML TABLE

Since a lot of the output of SQL is in the form of tables, it's helpful if we understand the HTML structure we need to create tables on the web page. An HTML table is created in the following way:

- <table></table> tags around the outside of the table element
- Inside the table tags, <tr></tr> tags for each Table Row that is required.
- Inside the tr tags, <td></td> tags for each table data cell needed.
- Inside each td tag, optional text that will appear.

For example the following code will create a 2x2 table:

```
<table>
      <tr>
            <td>Cell 1</td>
            <td>Cell 2</td>
      </tr>
      <tr>
            <td>Cell 3</td>
            <td>Cell 4</td>
      </tr>
</table>
```

Exercise: Create a table on your webpage that contains 4 rows and 3 columns per row. Put any text you like inside each cell.

You'll notice that there are no borders at the top and bottom of some of the cells. This is because the styles.css file has been created to format the table in this way. To make out table look better, we should tell the browser that the top row of the table is the "tablehead". By doing this, the browser will look inside the *styles.css* file for the *tablehead* class. It will read the formatting information contained within, and make the top row of the table look better. To add the *tablehead* class to the first row in the table, we add an attribute class="tablehead" to our first <tr> tag. Like below:

```
<table>
      <tr class="tablehead">
            <td>Cell 1</td>
            <td>Cell 2</td>
      </tr>

      ...

</table>
```

Only make this change to the first row in your table, otherwise the reformatting will be applied to every row.

Now that you know how to make a table, we should attempt to create one as if we were working with the database. At first we will create one by hand, then we will connect to the database and retrieve information using code.

Exercise: Create a new table using HTML that looks like the one below (Obtained using `SELECT artName, cdTitle, cdPrice, cdGenre FROM Artist NATURAL JOIN CD WHERE artName LIKE 'M%'`):

```
+-------------------------------+----------------------------+---------+-------------+
| artName                       | cdTitle                    | cdPrice | cdGenre     |
+-------------------------------+----------------------------+---------+-------------+
| Mark Ronson                   | Version                    |   11.99 | Rock        |
| Mark Ronson & The Business Intl | Record Collection        |   12.99 | Pop         |
| Maroon 5                      | Hands All Over             |   11.99 | Pop         |
| Mr. Scruff                    | Ninja Tuna                 |    9.99 | Electronica |
| Muse                          | Black Holes and Revelations |    9.99 | Rock        |
| Muse                          | The Resistance             |   11.99 | Rock        |
+-------------------------------+----------------------------+---------+-------------+
```

It's important to become familiar with the HTML Table element, so that we can produce it using PHP code later in the exercise.

STARTING WITH PHP

Now we're familiar with the main elements of HTML we need, we can begin using PHP to access the database. For all remaining work in this exercise, you can either delete what you have previous done, between <body> and </body> or you can add onto the end. It depends on whether you'd like to keep all your previous HTML answers intact. Remember, everything must go inside your body tags.

To begin writing a php script, we must open and close a php block. Like this:

```
<body>
      <h1>Database Systems<h1>
      <h2>Username<h2>
      … possibly some previous tables you've created here …
<?php

?>
</body>
```

It's important to remember that when you're writing inside php tags, you're writing php code. Outside of php tags, you're writing HTML text. Begin by using the *echo* command to output some text using PHP. Like this:

```
<?php
      echo "This is some text!";
?>
```

Exercise: Use multiple echo commands to output 5 lines of text. The text can be anything you like. Go to your browser and see if they appear. You might notice that they will all appear on the same line in your webpage.

One problem with HTML is that most of the time it will ignore information that isn't text, including new lines. We can tell HTML to put a new line in, by using `<br/>`. The `br` tag doesn't need anything between them, so we can use a single, self-closed tag. So to output text on separate lines, echo some pre tags like this:

```
echo "some text";
echo "<br/>";
echo "some text on a new line";
```

Exercise: Use the code above to separate your echo commands onto new lines in your web page.

PHP VARIABLES

Much like other programming languages, PHP uses variables to store information throughout a script. A variable is declared like this:

```
$variablename = value;
```

Variables in PHP can be anything you like, their type is inferred from the value you pass to it. You can then use variables in functions, in arithmetic, and output them using echo. For example:

```
$var1 = 5;
$var2 = 15;
$var1 = $var1 + 5;
echo $var1 * 20 + $var2;
```

Exercise: Create a variable called $variableone that equals 10. Next multiply this variable by 45, then add 50. Finally, multiply this variable by 3.14 and use echo to display the result on your webpage.

If you wrote the code correctly, your webpage should show the number 1570.

Next we will create some strings. Strings are much like any other variable in PHP, and are equivalent to VARCHAR in SQL. You create them in the same way, and can append strings together using the concatenate operator '.'.

```
$var1 = "String text 1";
$var2 = $var1 . " and some more text";
echo $var2 . " then even more text";
```

Exercise: Create 3 string variables containing any text you like, then use echo to output all strings on a single line. You may wish to add " " space strings between your variables to output spaces too.

IF STATEMENTS

IF statements are used to run specific segments of code, depending on some conditional statement. They are structured like this:

```
$varx = 15;
if($varx < 5)
{
      echo "IF Statement Run 1";
}
else if ($varx > 10)
{
      echo "ELSE IF Statement Run 2";
}
else
{
      echo "ELSE Statement Run 3";
}
```

The conditional part of if statements can contain boolean variables, or use variables and conditional operators as above.

Exercise: Copy the above IF statements into your code. Then change the value of $varx to ensure that statement 1 runs. Check in your browser to test if this has worked. Next, change the value of $varx two more times to have the PHP script output statement 2 and statement 3.

WHILE LOOPS

While loops are useful if you'd like to run the same section of code repeatedly. For example, we will use one to output each row of our database result later. The general structure of a while loop is as follows:

```
while (condition)
{
      // Some code here
}
```

Let's imagine we want to output a list of numbers from 1 to 20 onto the webpage. First we need to create a variable to hold the current number, then inside the loop we output the number, and then add one to it. The while condition must be true while the number is less than 21. Like this:

```
$number = 1;
while ($number < 21)
{
      echo $number . " ";
      $number = $number + 1;
}
```

Exercise: Write a while loop in your PHP code to output the numbers 10 to 30. Then try to change the loop to output numbers 20 to 1, in reverse order! Hint: Try starting the $number higher, and decreasing each loop.

CONNECTING TO MYSQL

By now you should understand all you need to start using database output on your web page. We will now look at how to connect to MySQL, and query the database. In general, the process of connecting to MySQL can be summarized as follows:

1. Use function mysql_connect() to connect to the server with your username and password
2. Check connection has been successful
3. Use the function mysql_select_db to select your database on mysql.
4. Check selecting a database has been successful.

To make things easier, we've put all the necessary code into *dbconnect.php*, which you should have open. Have a look through, and make sure you understand what it is doing. To use this file, we need to include it into our index.php. This will run this code at the specified point, even if it's in a different file. This means we can alter *dbconnect.php* once, then leave it alone and simply include it every time we need to connect to a database.

Exercise: Look inside dbconnect.php and see how it works. Change the strings for server, username and password to 'mysql.cs.nott.ac.uk', your username and password. You will also need to add your username on the mysql_select_db line too. On the school servers, your database is named the same as your username.

Exercise: Connect to mysql by including *dbconnect.php* inside your *index.php* using either include_once() or require_once() like this:

```
include_once('dbconnect.php');
```

If you run index.php now, you should notice no change. If the connection *hasn't* been successful, you will see error messages. No error messages means no problem!

At this point, your *index.php* file should look a little like this:

```
<html>
	<head>
		<title>G64DBS Lab</title>
		<link rel="stylesheet" type="text/css" href="styles.css" />
	</head>
	<body>
		<h1>Database Systems<h1>
		<h2>Username<h2>
		… possibly some previous tables you've created here …
	<?php

		// Possibly some previous php exercise answers in here.
		include_once('dbconnect.php');

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

QUERYING MYSQL FROM PHP

Now we've connected to PHP, we can send off a query to the database. You can send any command (E.g. INSERT, CREATE, UPDATE) but for these exercises we're going to focus on SQL SELECT. The general process for querying the database is as follows:

1. Create a string variable to hold the text of our select query. E.g. `"SELECT * FROM CD"`
2. Use the `mysql_query()` function to send the above string to the database. This will return a result, which we will store in a variable
3. Create a while loop to retrieve every row in the output, using the `mysql_fetch_array()` function.
4. Inside the loop, we can output the result as `$row['Name of Column']`

The code below shows you how to achieve these steps for a very simple query. Line numbers are shown and an explanation of each line is also provided:

```
 1  include_once('dbconnect.php');
 2
 3  $query = "SELECT artName FROM Artist";
 4  $result = mysql_query($query);
 5
 6  while ($row = mysql_fetch_array($result))
 7  {
 8      echo $row['artName'];
 9      echo "<br/>";
10  }
```

Explanation of each line:

1	Includes dbconnect.php and connects to MySQL.
2	-
3	Creates a string that holds our query.
4	Sends the query to MySQL and returns the result.
5	-
6	While loop. Obtains the next row of our result. If there are no rows left, $row will be false and the while loop will stop.
7	-
8	Output the current 'artName' in our row to the web page.
9	Output a new line tag. So that all artists don't appear on the same line.
10	-

Remember, all SQL SELECT queries you might wish to make can be achieved by adapting the code above.

Exercise: Use the code above, suitably adapted, to output a list of cdTitles from the CD table to the web page.

Exercise: Further adapt the code to output a list of "cdTitle, cdGenre" pairs, separated by a comma. If you do it correctly, the output should look like this:

Black Holes and Revelations, Rock
The Resistance, Rock
Ninja Tuna, Electronica
For Lack of a Better Name, Electro House
Version, Rock
Record Collection, Pop
Merriweather Post Pavilion, Electronica
Only By The Night, Rock
Come Around Sundown, Rock
Hands All Over, Pop

Congratulations! You've now successfully created a dynamic web page that reads information from a database. While this website is not particularly functional or good looking, almost all websites are built around the principles that you've just learned.

More Advanced PHP and MySQL

Before we can start drawing large sets of results from our database, it would be helpful to develop a while loop that would output HTML tables, rather than simply values separated by commas. Remember that PHP echo writes out HTML text, so we can output the <table> tags and create a table. A very simple example is shown below, and uses the `class="tablehead"` that we learned about earlier.

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

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

This code works in a similar way to the previous examples, but now outputs table tags where necessary. Before and after the while loop, <table> tags tell the browser there is a table element. We also output a single table row (<tr>) before the while loop that contains our column names. Remember all data that goes into the table row must be in a cell, in a set of <td> tags. Each loop we output a new table row, then a single column with our artist name inside. We also close every tag we open, as HTML requires.

Note: We have to escape " characters when they appear inside strings. We do this with a backslash, like this `\"`. This makes sure that PHP knows we want " to be part of our string, rather than the beginning or end of a string. This is very important!

Exercise: Adapt your cdTitle, cdGenre query from the previous exercise to output information into a table instead. Each table row will need two sets of <td> tags to hold cdTitle and cdGenre. Once you are done, your web page table should look like this:

cdTitle	cdGenre
Black Holes and Revelations	Rock
The Resistance	Rock
Ninja Tuna	Electronica
For Lack of a Better Name	Electro House
Version	Rock
Record Collection	Pop
Merriweather Post Pavilion	Electronica
Only By The Night	Rock
Come Around Sundown	Rock
Hands All Over	Pop

Exercise: Now it gets a little more difficult! As above, use a while loop in PHP to output an HTML table to hold the information produced by this query "`SELECT artName, cdTitle, cdGenre, cdRating, cdYear FROM Artist NATURAL JOIN CD`". The table on your web page should look like this:

artName	cdTitle	cdGenre	cdRating	cdYear
Animal Collective	Merriweather Post Pavilion	Electronica	82	2009
DeadMau5	For Lack of a Better Name	Electro House	38	2009
Kings of Leon	Only By The Night	Rock	67	2008
Kings of Leon	Come Around Sundown	Rock	31	2010
Mark Ronson	Version	Rock	77	2007
Mark Ronson & The Business Intl	Record Collection	Pop	22	2010
Maroon 5	Hands All Over	Pop	64	2010
Mr. Scruff	Ninja Tuna	Electronica	55	2008
Muse	Black Holes and Revelations	Rock	78	2006
Muse	The Resistance	Rock	90	2009

Exercise: Write PHP to output a table to provide a list of artist names along with the average rating of their albums, and their nationality. You will have to design an SQL SELECT statement for this as well. The web page output should look something like this:

artName	Average Rating	artNationality
Animal Collective	82.0000	American
DeadMau5	38.0000	Canadian
Kings of Leon	49.0000	American
Mark Ronson	77.0000	British
Mark Ronson & The Business Intl	22.0000	British
Maroon 5	64.0000	American
Mr. Scruff	55.0000	British
Muse	84.0000	British

Exercise: You might have noticed that our Average rating has been calculated to a very large degree of accuracy, when in fact all we need is integer precision. Use the `round()` function supplied in PHP to remove these zeros. All you need to do is use `round($row['Average Rating'])` or similar, depending on your column alias. Once you're done, you should see this:

artName	Average Rating	artNationality
Animal Collective	82	American
DeadMau5	38	Canadian
Kings of Leon	49	American
Mark Ronson	77	British
Mark Ronson & The Business Intl	22	British
Maroon 5	64	American
Mr. Scruff	55	British
Muse	84	British

ADVANCED HTML FORMATTING WITH PHP

The following exercises are designed to demonstrate the power of connecting to MySQL using a programming language, rather than having all results output to simple text terminals. We're going to start by colouring the output text for the ratings column, then we're going to replace the nationality information with flag images!

Let's start by colouring the ratings column text, depending on what rating the artist has received. To begin with, we'll only handle very low ratings. You can colour text in the HTML table by applying a class to the <td> tag. The classes, like with "tablehead" from earlier, have been defined in our *styles.css* file. You can use the classes like this:

```
<table>
<tr>
<td>Cell with regular text</td>
<td class="greatrating">Great: Green</td>
<td class="goodrating">Good: Yellow</td>
<td class="okrating">Average: Orange</td>
<td class="poorrating">Poor: Red</td>
</tr>
...
</table>
```

Like common music rating websites, we're going to start by colouring low ratings with red text. Somewhere inside your while loop you should have some code that looks similar to the following (don't worry if your aliases are different or you have split this code over multiple lines and echo statements):

```
echo "<td>" . round(row['Average Rating']) . "</td>";
```

We still want to output the average rating for each artist, but we want to colour it depending on what it's value is. First, we will use a variable as a temporary storage for the Average Rating, so we can find out the value. Like this:

```
$roundedrating = round($row['Average Rating']);
echo "<td>" . $roundedrating . "</td>";
```

This will make no difference to what our page looks like yet, because we have simply read a variable and output it again. Next we will write an if statement to determine if we want to recolour the text, and if we do, we can use the td class:

```
$roundedrating = round($row['Average Rating']);

if ($roundedrating < 40)
{
      echo "<td class='poorrating'>" . $roundedrating . "</td>";}
else
{
      echo "<td>" . $roundedrating . "</td>";
}
```

This code should colour our text red, if the rating is poor, otherwise it will remain black. If you've done this correctly, you should see the following table on your web page, low ratings have been coloured red:

artName	Average Rating	artNationality
Animal Collective	82	American
DeadMau5	38	Canadian
Kings of Leon	49	American
Mark Ronson	77	British
Mark Ronson & The Business Intl	22	British
Maroon 5	64	American
Mr. Scruff	55	British
Muse	84	British

Exercise: Extend your code using ELSE IF statements to colour the remaining ratings using the rules below:

Average Rating	Colour	Class Name
0 – 39	Red	'poorrating'
40 – 59	Orange	'okrating'
60 – 79	Yellow	'goodrating'
80 – 100	Green	'greatrating'

If you do this correctly, your output should look like this:

artName	Average Rating	artNationality
Animal Collective	82	American
DeadMau5	38	Canadian
Kings of Leon	49	American
Mark Ronson	77	British
Mark Ronson & The Business Intl	22	British
Maroon 5	64	American
Mr. Scruff	55	British
Muse	84	British

Next will replace the string showing the artists' nationalities by images of their flag. Most of the work has already been done for us. In your `public_html/images/flags` folder there are 3 flag images that you can use. Inside *functions.php* there is a function called get_flag() that will convert a string nationality into an HTML image tag.

Exercise: Use `include_once` to include *functions.php* in your code. Do not put this line inside your while loop, put it at the top of your php near the *dbconnect.php* include. Next, use the `get_flag()` function and pass it the nationality for each row, it will return a flag image tag, that you can echo as normal. If you do this right, your final table should look like this (I've renamed the column titles to make them look neater):

Artist Name	Average Rating	Artist Nationality
Animal Collective	82	
DeadMau5	38	
Kings of Leon	49	
Mark Ronson	77	
Mark Ronson & The Business Intl	22	
Maroon 5	64	
Mr. Scruff	55	
Muse	84	

Last of all, we'll use some CSS formatting to make the web page look a lot nicer! All the code is written for us in *functions.php*, and we've already included it so we just need to make use of the functions. We're going to use two functions, `start_div_region($title)` and `end_div_region()`. If you like, open functions.php and have a look at these. They simply output some more HTML tags, that are formatted using *styles.css*.

You can use these functions as many times as you like, to put each of your tables into a nice web page region, with a border and title. Before you use `echo "<table>";` you must call `start_div_region("Name of Table")` with any name you like. After the while loop and closing table tag, you can use `end_div_region()`. Like this:

```
start_div_region("Table showing Artist Names and CD Information");

... The rest of your table code and while loop ...

end_div_region();
```

Exercise: Use the two functions described above to place your table inside a border. Make sure that *all* table HTML code is inside between the two functions. If all goes well, your final web page will look like this:

Database Systems

Username: user

Table showing Artist Names and CD Information

Artist Name	Average Rating	Artist Nationality
Animal Collective	82	
DeadMau5	38	
Kings of Leon	49	
Mark Ronson	77	
Mark Ronson & The Business Intl	22	
Maroon 5	64	
Mr. Scruff	55	
Muse	84	

You may still have other tables and output from previous exercises, don't worry about this.