COMP519 Practical 16
PHP (5)

Introduction

- This worksheet contains further exercises that are intended to familiarise you with PHP Programming. In particular, we will consider the use of the PHP Data Objects (PDO) extension for accessing databases that is independent of the specific DBMS that is used. This is often preferable over DBMS specific approaches, like the MySQLi extension to access MySQL databases, as it improves the portability of your code.

  While you work through the exercises below compare your results with those of your fellow students and ask for help and comments if required.
- It is assumed that you have completed the exercises in the previous COMP519 practical and that you have an account with the departmental MySQL DBMS and that you have created a `meetings` table on it.
- You might proceed more quickly if you cut-and-paste code from this PDF file. Note that a cut-and-paste operation may introduce extra spaces into your code. It is important that those are removed and that your code exactly matches that shown in this worksheet.
- The exercises and instructions in this worksheet assume that you use the Department's Linux systems to experiment with PHP.
- To keep things simple, we will just use a text editor, a terminal, and a web browser. You can use whatever text editor and web browser you are most familiar or comfortable with.

Exercises

1. Let us try to connect to our MySQL database using PHP.

   a. Open a text editor, enter the following HTML markup and PHP code, save it to a file named `php16A.php` in `$HOME/public_html/`.

```
<!DOCTYPE html>
<html lang='en-GB'>
  <head>
    <title>PHP16 A</title>
  </head>
  <body>
    <h1>PHP and Databases</h1>
<?php
$db_hostname = "studdb.csc.liv.ac.uk";
$db_database = "<user>";
$db_username = "<user>";
$db_password = "<password>";
$db_charset = "utf8mb4";

$dsn = "mysql:host=$db_hostname;dbname=$db_database;charset=$db_charset";
$opt = array(
```

```
    PDO::ATTR_ERRMODE          => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    PDO::ATTR_EMULATE_PREPARES => false
);
try {
   $pdo = new PDO($dsn,$db_username,$db_password,$opt);

   // Code for 3c here
   // Code for 3d here
   // Code for 4a here
   echo "<h2>Data in meeting table (While loop)</h2>\n";
   $stmt = $pdo->query("select * from meetings");
   echo "Rows retrieved: ".$stmt->rowcount()."<br><br>\n";
   while ($row = $stmt->fetch()) {
      echo "Slot: ",$row["slot"], "<br>\n";
      echo "Name: ",$row["name"], "<br>\n";
      echo "Email: ",$row["email"],"<br><br>\n";
   }
   echo "<h2>Data in meeting table (Foreach loop)</h2>\n";
   $stmt = $pdo->query("select * from meetings");
   foreach($stmt as $row) {
      echo "Slot: ",$row["slot"], "<br>\n";
      echo "Name: ",$row["name"], "<br>\n";
      echo "Email: ",$row["email"],"<br><br>\n";
   }
   $pdo = NULL;
} catch (PDOException $e) {
   exit("PDO Error: ".$e->getMessage()."<br>");
}
?>
  </body>
</html>
```

Replace both occurrences of *<user>* with your MySQL username and replace *<password>* with the password you have chosen for your MySQL account.

b. Use the documentation at http://php.net/manual/en/pdo.connections.php to understand what the various PDO-functions in the code do.

c. Make sure that nobody but you has read access for the file by using

```
chmod u+r,og-rwx $HOME/public_html/php16A.php
```

(Note: No space after the comma!) You should only have to do so once. File permissions should not change while you continue to edit the file.

d. Execute the PHP script in the terminal using the command

```
php $HOME/public_html/php16A.php
```

Check that there are no syntax error and that the script produces the output

```
<!DOCTYPE html>
<html lang='en-GB'>
  <head>
```

```
    <title>PHP 16A</title>
  </head>
  <body>
    <h1>PHP and Databases</h1>
<h2>Data in meeting table (While loop)</h2>
Rows retrieved: 3<br><br>
Slot: 1<br>
Name: Michael North<br>
Email: M.North@student.liverpool.ac.uk<br><br>
Slot: 5<br>
Name: Jody Land<br>
Email: J.Land@student.liverpool.ac.uk<br><br>
Slot: 7<br>
Name: Trish Shelby<br>
Email: T.Shelby@student.liverpool.ac.uk<br><br>
<h2>Data in meeting table (Foreach loop)</h2>
Rows retrieved: 3<br><br>
Slot: 1<br>
Name: Michael North<br>
Email: M.North@student.liverpool.ac.uk<br><br>
Slot: 5<br>
Name: Jody Land<br>
Email: J.Land@student.liverpool.ac.uk<br><br>
Slot: 7<br>
Name: Trish Shelby<br>
Email: T.Shelby@student.liverpool.ac.uk<br><br>
  </body>
</html>
```

e. Open a web browser and access the url

`https://student.csc.liv.ac.uk/~<user>/php16A.php`

where *<user>* should be replaced by your University (MWS) username.

Make sure that the web page you are shown corresponds to the HTML code you have seen in Exercise 1d.

f. It would be nice if the database data would be presented in the form of a HTML table with three columns called 'Slot', 'Name' and 'Email'. Change the code of your PHP script so that two such tables are produced. Also, modify the query so that entries in the tables will be ordered by slot number.

2. We now want to add some interactivity to our web application.

a. Add the following code to `php16A.php` just before the statement `$pdo = NULL`.

```
echo "
<form name='form1' method='post'>
  <select name='select' onChange='document.form1.submit()'>
    <option value='None'>Select a name</option>";
// Add further options here
echo "
  </select>
```

```
</form>";
foreach ($_REQUEST as $key => $value)
  echo "$key => $value<br>\n";
```

b. Save the modified file, check that your code is syntactically correct by executing the script in a terminal, then refresh the URL

`https://student.csc.liv.ac.uk/~<user>/php16A.php`

in your web browser. You should now see a rudimentary drop-down menu at the bottom of the page.

c. At the point indicated by the comment "`Add further options here`" in Exercise 2a, add PHP code that generates additional options for the drop-down menu, one for each entry in the `meetings` database table. The value attribute for each option should be the email address stored in the database while the label should be the name, for example:

```
<option value='T.Shelby@student.liverpool.ac.uk'>Trish Shelby</option>
```

As in Exercise 1f, the required data should be retrieved from the database.

d. Once you have successfully completed Exercise 2c, observe what happens if you select one of the options in the pop-up menu. You should see additional text at the bottom of the web page, for example

```
select => M.North@student.liverpool.ac.uk
```

if you have selected the name `Michael North` among the options.

Make sure that you understand where this text comes from and how it comes about.

e. Modify your script so that the output you see in Exercise 2c is no longer produced by the script, but instead the script adds

```
You can contact Michael North via the e-mail address
M.North@student.liverpool.ac.uk
```

at the bottom of the page. Make sure that on the first visit of the URL no text is shown. Hint: The PDO function `fetch` described at `http://php.net/manual/en/pdostatement.fetch.php` will be useful, as you only retrieve one row from the database.

3. The next task is to add a facility that allows us to insert new data into the database via our web page.

a. Add the following code to `php16A.php` just before the statement `$pdo = NULL`.

```
echo "
<form name='form2' method='post'>
 Slot: <input type='number' name='slot' min='1' max='100'><br>
 Name: <input type='text' name='name' size='100'><br>
 Email: <input type='text' name='email' size='100'><br>
<input type='submit' name='insert' value='Insert into DB'>
<input type='submit' name='delete' value='Delete from DB'>
<input type='submit' name='query' value='Query DB'>
</form>";
```

b. Save the modified file, check that your code is syntactically correct by executing the script in a terminal, then refresh the URL for the script. You should now see a form that allows you to enter a slot number, name and e-mail address.

c. Add code to your PHP script after the comment "`Code for 3c here`" and before the comment "`Code for 3d here`" that does the following: If a user supplies a non-empty slot number, name and e-mail address using the form introduced in Exercise 3a and clicks on the 'Insert into DB' button, then your code should insert those values into the database. If the insertion is successful, a success message should be shown. If the insertion fails (which will be the case if the slot number already exists in the database), then a failure message should be shown that includes the error message you get from MySQL.

Hints:

- First check whether the user has clicked the 'Insert into DB' button. If so, check whether the user has provided a slot number, name and e-mail address. If one of the pieces of information is missing, generate an error message and do not proceed to insert the incomplete information that was provided by the user.
- A naive solution will retrieve the three values entered by the user, construct an SQL query as a string containing those values and execute that query, using code like

```
$query = "insert into meetings (slot,name,email) values(
 {$_REQUEST['slot']},\"{$_REQUEST['name']}\",\"{$_REQUEST['email']}\")";
$success = pdo->query($query);
```

  This solution is vulnerable to SQL injection and should therefore by avoided. See http://php.net/manual/en/security.database.sql-injection.php for additional information.
- A better solution involves the use of a *prepared statement* and *parameter binding*. Prepared statements are a kind of compiled template for SQL statements that includes parameters/placeholders that will later be filled by values. Prepared statements offer two major benefits: (i) the SQL statements only need to be parsed (or prepared) once, but can be executed multiple times with the same or different values for the parameters, (ii) neither the parameters nor the values that are bound to them need to be quoted; this is handled automatically and in such a way that no SQL injection will occur.

  Using a prepared statement, the insertion of slot number, name and email address into our database may look as follows:

```
$stmt = $pdo->prepare(
             "insert into meetings (slot,name,email) values(?,?,?)");
$success = $stmt->execute(
       array($_REQUEST['slot'],$_REQUEST['name'],$_REQUEST['email']));
```

  Here, using `$pdo->prepare()`, we first create a prepared statement with three placeholders, indicated by `?`, that we will later bind to values. The binding of placeholders is done using `$pdo->execute()` that then also executes the query. The function takes as arguments an array with the values that should be bound to the placeholders. The function will return a boolean value indicating whether the execution has been successful.

  Instead of `?` it is possible to use *named placeholders*. The names of the placeholders must then all appear as keys in the array that is used to provide values for the placeholders:

```
$stmt = $pdo->prepare(
   "insert into meetings (slot,name,email) values(:slot,:name,:email)");
$success = $stmt->execute(
        array("name"  => $_REQUEST['name'],"slot" => $_REQUEST['slot'],
```

```
                "email" => $_REQUEST['email']));
```

The advantage of named placeholders is obviously that fewer errors with the order of values are made.
The manual pages for the relevant functions are
– http://php.net/manual/en/pdo.prepare.php,
– http://php.net/manual/en/pdostatement.bindparam.php, and
– http://php.net/manual/en/pdostatement.execute.php

d. Add code to your PHP script after the comment "`Code for 3d here`" and before the comment "`Code for 4a here`" that does the following: If a user supplies a non-empty slot number using the form introduced in Exercise 3a and clicks on the 'Delete from DB' button, then your code should attempt to delete any entry with a matching slot number from the database. If the deletion successfully removes an entry from the database, then a success message should be shown. If the deletion fails to remove anything from the database, then a failure message should be shown. If the database operation fails for any other reason, then a failure message should be shown that includes the error message you get from MySQL.

Hint: For database operations like 'update' and 'delete' `$pdo->execute()` will return `TRUE` even if no database entry was updated or deleted. To determine whether the operation was truely successful, one has to check whether the number of affected rows is greater than zero (or equal to the expected number of affected rows). `$stmt->rowcount()` returns the number of affected rows.

4. Finally, we add a facility that allows us to query the database using regular expressions.

a. Add code to your PHP script after the comment "`Code for 4a here`" that does the following: If a user supplies a regular expression in the name field of the form introduced in Exercise 3a and clicks the 'Query DB' button, then your code should retrieve and display all entries in the `meetings` table where the value in the name field matches that regular expression.

Hints:

- First check whether the user has clicked the 'Query DB' button. If so, check whether the user has entered something into the name field. If not, generate an error message and do not proceed to query the database.
- For information on regular expression matching in MySQL see http://dev.mysql.com/doc/refman/8.0/en/pattern-matching.html
- Just as in Exercise 3c, a prepared statement is the safest way to query the database. To prepare a statement for the query, to bind variables to the placeholder in the statement and to execute the statement, proceed as in Exercise 3c.
- Once the prepared statement has been executed, we can use a `foreach`-loop to access and print out each row that was retrieved as in Exercise 1a.
- It would be nice if the information retrieved from the database would be presented as HTML table, just as in Exercise 1f. Develop code for that.