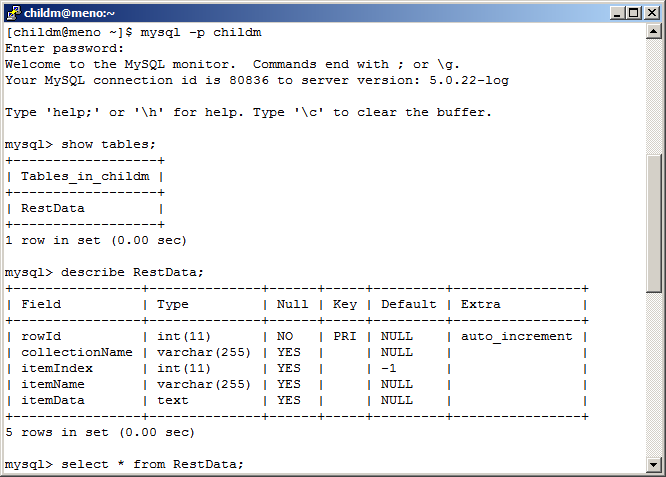
# PHP and MySQL

## EXERCISE 1: MySQL

MySQL is a database server. That means it sits and listens for connections. When someone connects it checks they have a user account and checks their credentials and then logs them in. After that it accepts commands in the SQL language and executes them, sending back data and response codes to the user. It is possible to communicate directly with a MySQL server using a command line console. Doing that looks like this:



Working with the database in this way is not much use for anything other than low level maintenance of your own account. For practical applications we want to be able to display data from the database as part of a website or the output of a mobile app. PHP is ideally suited to this.

### PDO versus MySQLi

PHP has two different libraries of instructions to use to access MySQL (in fact there is a third one too which is the old fashioned way of doing it and no longer recommended). This means that you may see very different code in different examples and tutorials on the web. All code used in this module uses the PDO library. The advantage of the PDO library is that it can actually talk to any database server, not just MySQL servers (for example Oracle, PostGRES, MariaDB etc. all of which are competitors to MySQL). So learning PDO enables you to write code for any database, while learning the MySQLi library is limited to MySQL databases alone. That said, the principles are all the same and having learnt one the other is easy to pick up.

As mentioned above, using the database involves connecting to it with a username and password, and once connected, sending it SQL commands and reading the results back. To connect to the database server from a PHP script you need to know the following:

1. The name of the database server.
2. Your username.
3. Your password.
4. The name of the database within your account that you want to use.

You have a MySQL account set up on the Daydream server. It only allows you to have a single database (containing as many tables as you want), and that database has the same name as your username. The things you need to connect to the Daydream MySQL are as follows:

1. The name of the database server is **bcimsql.lsbu.ac.uk**.
2. Your username is the **same as your windows username** for the University.
3. Your password is **your student number**.
4. The name of the database is the **same as your username**.

Because *every* PHP script which you write that needs to connect to MySQL needs this same information, the best way to do it is to put it in a function in one file, and then include that file at the top of the scripts you write which want to use the database.

The following is the PHP code for a connect function. Copy and paste it into NotePad++ and save it with the name **dbutils.php**. This should be placed in the **lib** folder you already have which contains the lightbox folder. In the first three lines of the function, type your username and password into the empty quotes. The database hostname is already there.

|  |
| --- |
| <?php  function connect() {  $username = '';  $password = '';  $mysqlhost = 'bcimsql.lsbu.ac.uk';  $dbname = $username;  $pdo = new PDO('mysql:host='.$mysqlhost.';dbname='.$dbname.';charset=utf8', $username, $password);  if ( $pdo) {  // make errors throw exceptions  $pdo->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);  return $pdo;  } else {  die("Could not create PDO connection.");  }  }  ?> |

Now copy the following into NotePad++ and save as a file called **dbtest.php**. Do not save this one in the lib folder.

|  |
| --- |
| <?php  require\_once( 'lib/dbutils.php');  ?>  <!DOCTYPE html>  <html>  <head>  <title>DB Test</title>  </head>  <body>  <?php  $pdo = connect();  if ( $pdo) {  print "<h1>Successfully connected!</h1>\n";  } else {  print "<h1>Did not connect.</h1>\n";  }  ?>  </body>  </html> |

All this script does is include the **dbutils.php** file with the **require\_once** command, and then attempt to connect by calling the **connect()** function. You should get either error messages because you have not properly included the **dbtest.php** file, or not entered the correct username and password details into it or you should get the “Successfully connected!” message.

The connect function returns an object representing the connection to the database server, which this script assigns to **$pdo**. We can use the **$pdo** variable to send commands to the database and retrieve results.

Because we don’t have any tables at the moment, there is not much we can do. Add the following functions to the **dbutils.php** file you already have (the one in lib ***NOT*** dbtest.php), after the existing connect function.

|  |
| --- |
| function tableExists( $pdo, $table) {  $sql = "SHOW TABLES LIKE '$table'";  return ( $pdo->query( $sql)->rowCount() > 0);  }  function createDemoTable( $pdo) {  if ( !tableExists( $pdo, 'demotable')) {  $sql = "CREATE TABLE IF NOT EXISTS `demotable` (  `idnumber` int(11) NOT NULL auto\_increment,  `firstname` varchar(255) NOT NULL,  `lastname` varchar(255) NOT NULL,  `age` int(11) NOT NULL,  PRIMARY KEY (`idnumber`)  )";  $pdo->exec( $sql);  $sql = "INSERT INTO `demotable`  (`firstname`, `lastname`, `age`)  VALUES  ('John','Smith',23),  ('Sue','Davis',13),  ('Peter','Young',45),  ('Alice','Brown',56),  ('Frank','Gray',22),  ('Betty','Redwood',27),  ('Jane','Corner',29),  ('Bob','Church',19),  ('Mary','Collins',67),  ('Herbert','River',42),  ('Lucy','Hardy',13),  ('Charles','Winter',81),  ('Samantha','Weather',37),  ('Karl','Saunders',34),  ('Angela','Brown',35)";  $pdo->exec( $sql);  }  }  function dumpTable( $pdo, $table) {  $sql = "SELECT \* FROM `".$table."`";  $stmt = $pdo->query( $sql);  // get all rows in an array  $results = $stmt->fetchAll(PDO::FETCH\_ASSOC);  print\_r( $results);  } |

This gives us three functions we can call from other scripts whenever required. Both these functions require an active connection to a database server to be given to them when called. The first, tableExists( $pdo, $table) is a useful function to determine whether a database table already exists. You give it a connection object and the name of a table you want to test for. It creates an SQL command in the variable $sql which asks the database to list all tables of that name. It then executes that SQL command by giving it to the *query method* the connection object has. This looks like this: $pdo->query($sql). This returns a result object, and the result object has a method rowCount() which tells how many rows were returned in the result – that is, how many tables of the given name were found. The tableExists function returns TRUE if any were found and FALSE if zero were found.

The second function creates a demo table and populates it with data. The table it creates is called demotable but before it does anything it calls tableExists( $pdo, ‘demotable’) to see if the table already exists. If it does, the function does nothing and returns. This is so that we can call this function over and over again and not keep adding the same data to the table over and over again. Assuming the table does not exist the function does the following:

1. Creates an SQL command to create a table and assigns it the variable $sql.
2. Executes this SQL command by giving the variable $sql to the exec method of the connection object which looks like this: $pdo->exec($sql).
3. Creates an SQL command to insert a whole load of rows into that table and assigns it to the variable $sql.
4. Executes this second SQL command by giving the variable $sql to the exec method of the connection object which looks like this: $pdo->exec($sql).

These SQL commands are executed using the *exec* method instead of the *query* method because they do not return a set of data from the database but just a success or failure code.

The third function, dumpTable simply queries the database for all the data in a given table and dumps it out in the raw format in which PHP receives it. In this case an SQL command is created and executed using the *query* method of the pdo object, and the result is assigned to $stmt. The result object provides access to all the rows of the table we have just selected. There are various ways to read them but this code simply uses the *fetchAll* method to put them into a PHP array, and then the **print\_r** function to print out the whole thing in one go.

Add the following lines at the bottom of your **dbtest.php** script.

createDemoTable( $pdo);

print "<pre>\n";

dumpTable( $pdo, 'demotable');

print "</pre>\n";

Make sure the updated versions of both files are on the server and view **dbtest.php** in a browser.

You should see the contents of the table displayed as a long list of array entries. It should look a bit like this:

Successfully connected!

Array

(

[0] => Array

(

[idnumber] => 1

[firstname] => John

[lastname] => Smith

[age] => 23

)

[1] => Array

(

[idnumber] => 2

[firstname] => Sue

[lastname] => Davis

[age] => 13

)

[2] => Array

(

[idnumber] => 3

[firstname] => Peter

[lastname] => Young

[age] => 45

)

What you are seeing is an array with numbered indexes, where each numbered element is also an array. Each numbered array is one row from the database table. The arrays representing the rows do not have numbered elements, but named elements, and the names correspond to the columns in the table. Refer to the createTable function to see the columns of the table and the values that were inserted.

Now that we have access to this data, let’s try and display it a bit nicer. Add the following function to **dbutils.php**.

|  |
| --- |
| function htmlTable( $pdo, $table) {  $sql = "DESCRIBE `".$table."`";  $stmt = $pdo->query( $sql);  print "<table>";  print "<tr>";  foreach( $stmt as $v) {  print "<th>".$v['Field']."</th>";  }  print "</tr>";    $sql = "SELECT \* FROM `".$table."`";  // specify only an associative array to be returned  $stmt = $pdo->query( $sql, PDO::FETCH\_ASSOC);  foreach( $stmt as $row) {  print "<tr>";  foreach( $row as $v) {  print "<td>".$v."</td>";  }  print "</tr>";  }  print "</table>";  } |

This performs two queries on the database and uses them to print out an HTML table. The first is “DESCRIBE table” which returns a result row for each column in the table. The rows give the names of the columns and other information about them. This is used to print out the headings of the HTML table. Then a “SELECT \* FROM table” query is carried out to get all the data from the table. This is the same data as we printed out before as an unformatted array, but this function iterates through each row and then iterates through each value in each row, printing them out between HTML <tr> and <td> tags.

In **dbtest.php**, add a call to this method giving it the name ‘demotable’. Since HTML tables look plain by default, add this set of CSS rules to the head of **dbtest.php** too. The end result should look like the inset picture.

|  |
| --- |
| <style>  **Result:**  table {  border-collapse: collapse;  }  tr:nth-of-type(odd) {  background: #ffff99;  }  tr:nth-of-type(even) {  background: #dddd88;  }  th {  background: #ffffff;  border-bottom: solid 2px black;  text-align: left;  padding: 4px 16px;  }  td {  padding: 4px 16px;  }  </style> |

Connect to the meno server and transfer copies of **dbutils.php** and **dbtest.php** to your meno server in a folder called dba. Don’t forget **dbutils.php** needs to be a folder called **lib**. Correct the connections details in the **connect()** function in dbutils.php and make sure the web page works as expected. You will need to change the MySQL password to whatever you received in your account creation email, and the mysql host to the string “localhost”.

Use PUTTY to log in to a terminal session to meno, and then log in to your MySQL account on the command line by typing **mysql -p *yourusername***followed by your meno MySQL password.

Use the command **show tables;** to list out the tables you have, and then **describe demotable;** to describe the demonstration table. Take a screenshot of just the terminal window with these commands showing and put it in your report. Now type a select statement to list out the contents of demotable, similar to what is in the screenshot at the beginning of this document.

Examine the PHP code of the function ***htmlTable*** in **dbutils.php** and try and make sense out of how it uses the two queries you have just typed by hand in order to get the headings of the table and then the values of the rows. Try and explain this in your report.

|  |
| --- |
| **EXERCISE 1: MYSQL** |
| ***Required in your report (basic exercise).*** |
| * The PHP code for the connect() function set up to connect to your account. * A clickable link to dbtest.php in your daydream account. * A screenshot of using the MySQL command prompt on Meno. * A clickable link to dbtest.php in your meno account. * An explanation of how the table description shown in your screenshot matches up to the PHP code in htmlTable. * Reflection on the tasks and all activities involved. |
| ***Extended Tasks*** |
| * Experiment with inserting rows to the table at the command line prompt and seeing the results on the web page. * Try UPDATE and DELETE queries at the command line too. * Try creating another simple table and adding some rows to it at the command line and add statements to your script to display this other table too. * Try the query **SELECT \* FROM demotable ORDER BY lastname, firstname** to see how the data can be ordered. * Modify the htmlTable function to take a third parameter called $orderby and use it to create a query like the one above, so that the calling code can specify the ordering to use on the table. |
| *Original additional work:*  *Investigate and experiment with any related subject matter that interests you.* |