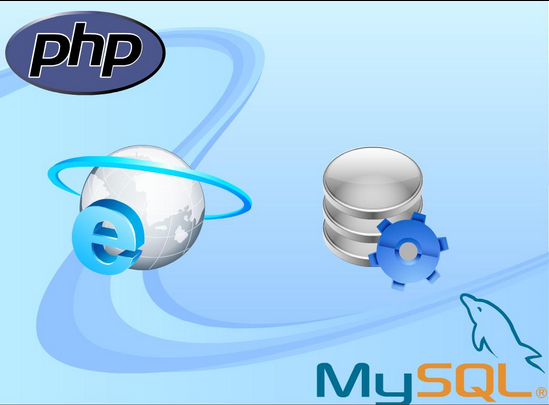
PHP and MySQL

Creating an Interactive Multi-User Learning Environment

Stewart Blakeway

Liverpool Hope University

1/1/2014



Written by Stewart Blakeway

Edited by Anna Kirpichnikova

**Table of Contents**

[Introduction 1](#_Toc717503556)

[Understating the structure of this document. 1](#_Toc2026630317)

[Your workspace 1](#_Toc94547453)

[Consistency 2](#_Toc194024812)

[PHP 4](#_Toc2006809220)

[Include 4](#_Toc477316747)

[Test and Debug 5](#_Toc1783639477)

[Database Design 5](#_Toc354075008)

[Creating a database 5](#_Toc1600793886)

[Creating a table and fields 8](#_Toc223178681)

[Creating a form 10](#_Toc882951248)

[Flow Control 10](#_Toc521195923)

[Associate Arrays 13](#_Toc270940880)

[isset() 14](#_Toc990744934)

[Retrieving Data from the database 15](#_Toc2097441553)

[Authentication and Access Control 19](#_Toc1575620879)

[Functions 19](#_Toc86884050)

[The PHP echo function and HTML forms 20](#_Toc1185981607)

[Session Variables 26](#_Toc364019542)

[Logging Out 27](#_Toc1124147419)

[Dynamic Pages 29](#_Toc1583577167)

[Tutor Pages 30](#_Toc2018372083)

[Access Control 30](#_Toc1443081333)

[Tutors Options 33](#_Toc246312814)

[Tutor Add Course 35](#_Toc1255614571)

[Student Enrol for Course 38](#_Toc523010703)

[Showing and authorising waiting students 45](#_Toc1118891317)

[Enter student registration details manually 59](#_Toc240452917)

# Introduction

This document will guide you through the process of creating an Interactive Multi-User Learning Environment. The system is controlled by a website as the front-end with a database serving the back-end. In order to follow these instructions you will need to have the following in place:

1. A web server (Uniform Server, XAMMP or alternative).
2. Sublime Text 3 (or your favourite text editor). Not a word processor like Microsoft Word. If you can apply formatting such as bold or italics you are not using a text editor.
3. A simple homepage with CSS (A template can be downloaded from course website).

# Understating the structure of this document.

Most of this document explains key fundamentals of the PHP scripting language. Although you may be tempted to implement when explanations are given, it is best to wait until instructed to do so. All actions are indicated by numbers in incrementing order starting at 1. Unless you see a number it is best to read and attempt to understand the explanations.

When we refer to a PHP function we apply different formatting. All functions that are discussed are formatted using the courier new font. For example: myFunction()

When we refer to folder or file names, we apply the same formatting but apply the colour blue: myFilename.php

# Your workspace

Managing your workspace is important; you need to know where your files are and the structure your website will take.

1. Go ahead and copy the folder aceTraining which contains a sample template website to the folder that serves the web pages on the web server.

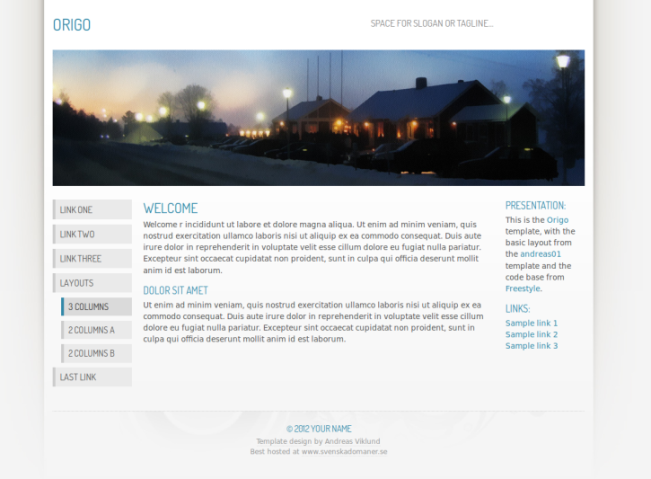
This is usually a folder called www (if using Uniform Server) or htdocs (if using XAMMP).

Before you start the web server you should ensure that you have switched the PHP configuration file from production to development. If you do not know how to switch to development mode for your version of web server you should investigate how to switch by consulting the documentation that came with your web server. Alternatively use a search engine for your server version. If you do not switch to development the web browser will not inform you of any syntax errors and simply present you with a blank page, essentially it will mean that you will have to debug your code without having the slightest clue where or why it went wrong.

1. Switch you PHP configuration file to development mode.
2. You should start the web server if you have not already done so.

Once the server has started load your favourite web browser and type localhost in the URL, if the web server is running correctly you will see the index page for the web server.

1. Change the URL of your favourite web browser to: http://localhost/aceTraining/ You should see:



We start our website by altering the index page. Open the index file in your favourite text editor. If you use Sublime Text 3 (available for Windows, Mac and Linux) the default code colouring will match the examples in this document.

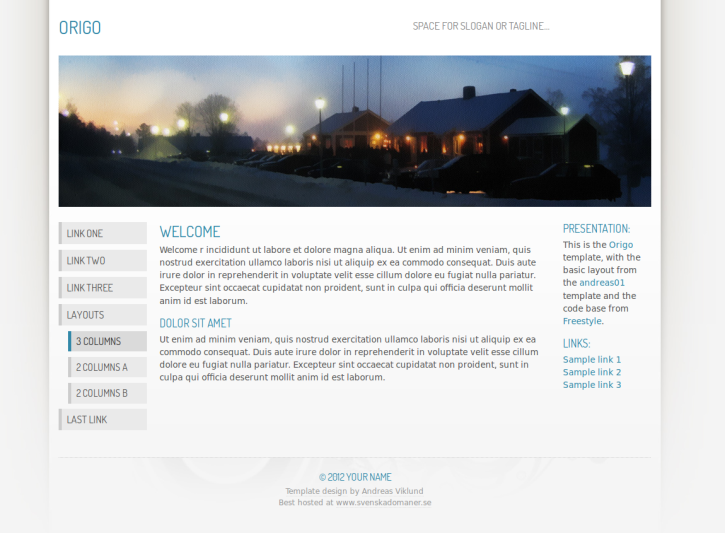
1. Open index.html from the aceTraining folder in your favourite text editor.

# Consistency

Consistency of a website is important because it places less of a cognitive load on the user and is one of the main design goals. By consistency we mean that most of the major elements are common for most of the pages on the website. These elements might be the: navigation menus, footer and other common elements. Normally the only content that changes is the main section.

In order to ensure consistency PHP provides a function called include(). This allows you to break down your web page into separate files and include them into the main document. For example consider the template provided, the areas that are surrounded by blue rounded rectangles will be consistent across all pages.

**main content area**



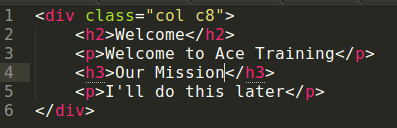
Rather than duplicate code across the entire site we will take these sections and put them into separate files.

1. Open the page named index.html in a text editor and save as index.php (this way we preserve our index.html file should something go wrong and we have to start again).
2. Select lines 1-30 and cut (some of the lines by be blank, include these anyway).
3. Create a new text document and save as: header.php, paste the lines of code into this file and save.
4. Go back to the index.php file and select lines 1-18 and cut.
5. Create a new text document and save as: leftNav.php, paste the lines of code into this file and save.
6. Go back to the index.php file and select lines 11-20 and cut.
7. Create a new text document and save as: rightNav.php, paste the lines of code into this file and save.
8. Go back to the index.php file and select lines 13-23 and cut.
9. Create a new text document and save as: footer.php, paste the lines of code into this file and save.
10. Save index.php.

Now you have a very minimalistic file that contains one div, a heading 2, a paragraph, a heading 3 and finally another paragraph.

1. Replace the text within the first paragraph to: Welcome to Ace Training. The text for heading 3 to: Our Mission and the paragraph text to: I’ll do this later!

Your index.php text file should now look like:



Don’t worry if your text is different in colours in your text editor, different editors use different colour schemes.

## PHP

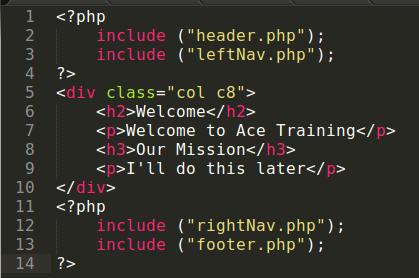
PHP scripts work within HTML documents (providing the document is saved with a php file extension). Most HTML tags have an open tag and a close tag, for example, <h2> and </h2>. In order to use PHP within the HTML document we need to define where the PHP starts and where the PHP finishes, we do this by using <?php to define the start of the PHP script and ?> to define the end of the script, this allows the web browser to differentiate between HTML and PHP.

In order to use a PHP function such as include() we have to enclose the code within PHP tags.

### Include

Include is a PHP function and uses this signature: void include(String); void tells us that a value is not returned. Include is the name of the function. String between the parenthesis is the data type of the parameter to be passed. The parameter is the location and name of the file that you wish to include into the main document, for example: **include(“menu.php”);** would include a file called menu.php into the main document.

1. Alter the code of index.php to match:



Our main content is now lines 5-10. Anything that is to be displayed in the **main content area** will be between <div class=”col c8”> and </div>.

### Test and Debug

Debugging as you progress is extremely important and can save a huge amount of time. Test now to ensure that your include functions are working correctly. The appearance of the web page should not have changed with the exception of the text in the main content area. If the main content area has not updated you may have to move (or rename) index.html as this may be loading by default rather than the desired index.php.

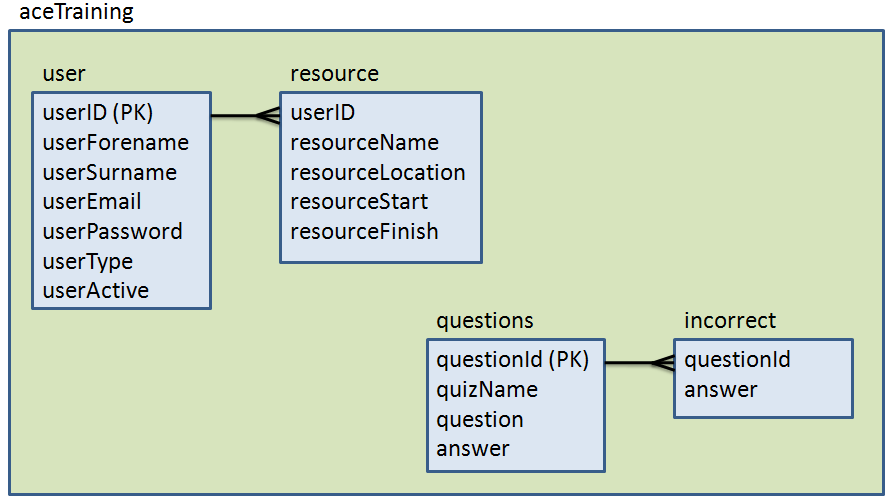
You can find more information on testing towards the end of this document, for now we are simply checking our scripts work.

1. Open index.php in your favourite web browser to ensure the includes are working.

# Database Design

As part of your assessment you will need to demonstrate that you have undergone the design process for your data storage requirements and your data flows. At a minimum you should have conceptual diagrams, logical diagrams, normalised tables, identification of data to be stored (data types, attributes, limitations) and identification of update and delete rules (i.e. restrict or cascade).

For the purpose of these exercises we will assume that the design process has been undertaken (very poorly) and the outcome is the following database structure:



## Creating a database

This section will show you how to construct a script that will create a database. During the development phase you may find that you will want to start afresh with the database several times. Creating a script that contains the database name and the structure of the tables allows you to modify the structure at the touch of a button as many times as you want.

We will use the index.php file for the example website. This file will be used as a template for most of our pages by saving it as the file that we are working on, in this case save the file now as: createDatabase.php

* + Create a connection to the SQL server
  + Create a SQL
  + Execute the SQL

Three simple steps will create a database. Firstly, we must connect to the SQL server, and the following code will do this:

$conn = mysqli\_connect ("localhost","root","root");

We are passing three parameters, the first is the location of the database server (localhost), the second is the username to access the server (root), and the third is the password for that username (root). Be advised that different server versions will have different passwords for the root user. If password root does not work try root123 or leave it blank. If this fails consult the documentation that came with the server that you are using. These notes are based on Uniform Server (Nano V5.0) and root is the default password.

The next step is to create the SQL statement.

$sql = "CREATE DATABASE aceTraining";

The final step is to execute the SQL statement. There are several ways of doing this, we could simply:

mysqli\_query($conn, $sql);

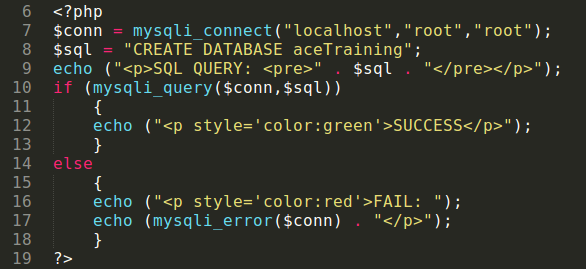
However, this will not tell us if something goes wrong, we could take this one step further and use the die() function and the mysqli\_error() function:

mysqli\_query($conn, $sql) **or die(mysqli\_error($conn));**

Again this should work but if something should go wrong an error message will be outputted to the browser. The error message is useful for debugging but what might not be as useful is that the server stops parsing the PHP file. If for example the database already existed you would get an error and the script would stop dead in its tracks.

It is often useful to output the SQL to the browser for development purposes, it is much easier to spot errors in the SQL without the other code surrounding the SQL. In this example we will create the SQL, output the SQL then execute the SQL.

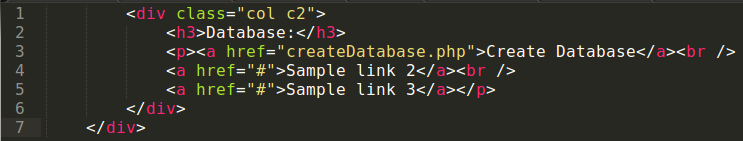
1. If you have not already done so save index.php as createDatabase.php
2. Replace lines 6-9 with the following code:



Essentially, when the function mysqli\_query() is executed it will return a value. If the value returned is anything but 1 (indicating true), then an error has occurred. The return value can be used in a conditional statement to display informative messages to the user, rather than error codes that the user probably will not understand.

Next we will update our links so that we can click a hyper-link to load this page and create our database.

1. Open the file called rightNav.php
2. Remove the heading 3 Presentation and the paragraph directly underneath.
3. Replace the word Links: with Database.
4. Change Sample link 1 to Create Database
5. Replace the first # with createDatabase.php. Your code should look like this:



1. Save your text document and go back to the web browser.
2. Refresh or load index.php.
3. You should now see a link Create Database on the right hand side. Click this link.

You should see the text **SUCCESS**. Refresh the page, you should now see **FAIL** with an error message stating why it failed.

## Creating a table and fields

We have to consider various factors when creating a table, for example are we going to create a field that is a primary key? Do we want an index number that automatically increments? What type of data will the individual fields store?

We will create a table based on the structure defined earlier. You will notice that we have a userID field. This is our primary key and should automatically increment each time a user is created. The other fields will store text, but how many characters? We need to consider carefully how many characters each field will store at maximum. For example, what is the maximum length of a surname? According to a quick Google search it’s 21 letters and is MacGhilleseatheanaich, however, we shall play it safe and set the length to 50 characters.

* + Create a connection to the SQL server
  + Create a SQL
  + Execute the SQL

Now that the database exist we will create a table. To create a table we refer to the same three steps as before. We need to include an additional parameter for the mysqli\_connect() function to indicate which database we are using.

1. Save createDatabase.php as createUserTable.php.
2. Line 7 contains the function that creates the connection string to the database, alter this line so that it reads:   
   $conn = mysqli\_connect("localhost","root","root","aceTraining");

We are not creating a database (we have done this) we are creating a table, therefore you will need to replace the SQL statement to reflect this.

1. Line 8 contains the SQL statement, replace the text between the double quotation marks to read:

CREATE TABLE user (

userID INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,

userForename VARCHAR(50) NOT NULL,

userSurname VARCHAR(50) NOT NULL,

userEmail VARCHAR(50) NOT NULL,

userPassword VARCHAR(50) NOT NULL,

userType VARCHAR(13) NOT NULL,

userActive BOOLEAN NOT NULL

)

The other steps are identical to the previous code.

1. Save the createUserTable.php file.

Your text file should look like:



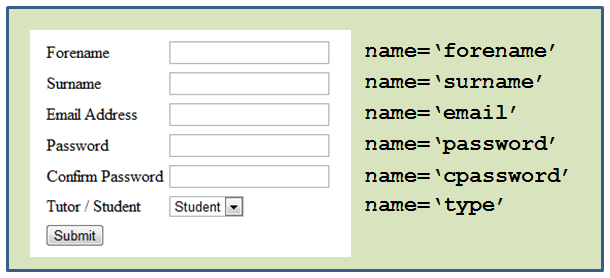
Next we need to create a link to the file that you have just created.

1. Open rightNav.php
2. Change Sample Link 2 to Create User Table
3. Change the # to createUserTable.php
4. Save, refresh/reload index.php and click the link to create the user table

With any luck you should see a message saying **SUCCESS**, if you refresh the page again the create table will fail because the table already exists. A suitable error will be displayed to reflect this.

# Creating a form

We now have a database with a table, we could continue and create our other tables but we will try and populate our existing table with data first. The usual way to get user input is by creating a form (although there are other methods like reading from a file).



The name attributes are the labels for the text boxes. The labels are used to access the data from the form elements. Once the form has been submitted to a PHP page we can access the users input by using the variable $\_POST[‘forename’]. You replace forename with the named element that you wish to access, for example, if you wanted to access confirm password you would replace forename with cpassword.

### Flow Control

Functions allow the re-usability of code. However in this example they are to be used for flow control, normally code is executed sequentially. The interpreter will read the file from top to bottom. Starting at line 1. There are situations where we will want to skip some sections of code entirely or execute alternative sections of code. The if/else conditional statement that was coded in the previous section is an example of this. Let’s look at one way of getting user input using a form:

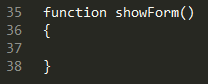
* If the user has not had the opportunity to enter data into the form then show the form
* Else, take the data that the user entered into the form and put it into the database.

Like before we will do this in steps, we will create two functions, one to get the data and one to process the data (i.e. put it into the database). Once we have created the functions we will apply some flow control that will allow us to call the required function at the appropriate time.

1. Save the createUserTable.php as register.php

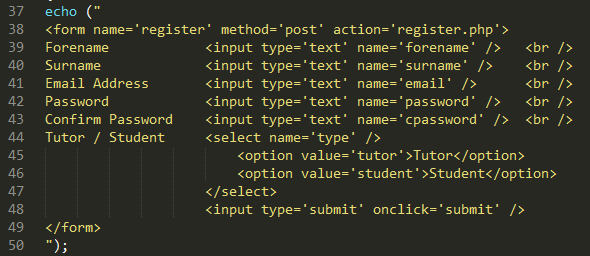
One of the beauties of PHP functions is that they do not need to be included in the main section of the document but can be called from these sections. Unlike some scripting or programming languages functions can be declared (created) after the function has been called. Therefore your function can be placed at the end of the document to keep the main section clean. This helps with the readability of your code. In our examples we will place our functions at the very end of the document.

1. Create a new line (line 35) and type the following code:

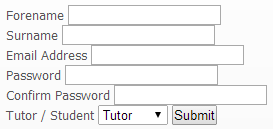


We are going to create a standard HTML form within this function. In order to use HTML with PHP we have to enclose the HTML in the PHP echo() function. The echo() function allows us to output text to the web browser, with the additional support of allowing us to mark-up this text using any HTML tag, for example we could display the text in header 1 mark-up using the <h1> </h1> tag. The fact that we can use any HTML tag means we can output a form.

1. Type the following code within the function that you have just created.



This results in a form that looks like:



This form could be better formatted using CSS or by placing the form elements in a table. You could investigate this further by visiting w3schools.com and reading about tables and CSS. If the content of the web pages is well presented you will gain more marks in your assessment. However, for the purpose of this tutorial we continue on – do not test just yet, it will not work because we haven’t called the function.

There is a slight problem with this solution, and that is we have created a PHP function outside of the PHP tags. Therefore, the server will not know how to interpret the function as PHP code. We can fix this by extending our current closing PHP tag.

1. Line 34 shows the end PHP tag. Move this to line 52 so that it encompasses the newly created PHP function.

Let’s look closely at the newly created form, we observe that the form tag has a method attribute. The method attribute signifies how to send the data. In our case the data is posted as a super global associate array (more on this later). The next thing to notice is the action attribute in the form tag, this simply tells the form where to send the data once the user clicks the submit button. In our case the form data is posted to itself!? This may initially seem confusing, but it does save us working with two different documents that accomplish an overall task. So far we have created a form to obtain user input.

Next we need to create a function that will process the data from the form, in our case we will put the data from the form into the database.

1. Immediately after the function previously created finishes (i.e. after the final } but before the ?>). Enter the following code.



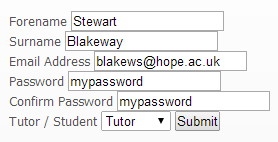
You should recognise most of this code. For example lines 60, 63-72 are identical to when you created the user table. That means you create a connection to the database server, display the SQL and then execute the SQL. The differences are lines 54-58. Nothing too complicated is going on here but it does give us the opportunity to discuss associate arrays.

## Associate Arrays

An associate array is a compound data type. This may sound complicated but basically it means: You have a variable, this variable contains many items of data (i.e. values), and these items of data are accessed by labels. The variable is called $\_POST, the items of data are the values the user entered into the form, the labels are created based on the value of the name attribute defined for each input element. Let’s take a look how this works:

The user enters these values into the form:

Each form element (i.e. each text box) has a name attribute, emboldened in the sample code below:



<input type='text' **name='forename'** />   
<input type='text' **name='surname'** />   
<input type='text' **name='email'** />  
<input type='text' **name='password'** />   
<input type='text' **name='cpassword'** />  
<select **name='type'** />  
 <option value='tutor'>Tutor</option>  
 <option value='student'>Student</option>  
</select>

Once the user clicks submit a **$\_POST** array is created to store the items of data, labels are created in the array so that we can get to the data later. The labels created in the array are **exactly** the same labels as the values for the name attributes (i.e. forename is not the same as Forename).

If you get an error message relating to an undefined index, an incorrectly named label is often the cause of such error. Check your $POST labels match your name attributes from the form. Remember email does not match Email or EMAIL, the names and the labels are case sensitive.

The variable (array) name is **$\_POST**, we access the data (i.e. Stewart) by using the label. The label must be between square brackets. In relation to our example:

$\_POST[‘forename’] contains the data item Stewart.  
$\_POST[‘password’] contains the data item mypassword.

Notice the single quotes; these are used because **all** data sent from a form are treated as text (a string of characters), including an empty textbox, we will revisit this later.

Lines 54-58 copy the data from the array to local variables. We do this to simplify the SQL statement later.

Line 61 and 62 constructs a standard SQL statement to insert data into a database. The only reason that we split the statement over two lines is for readability. Two lines does not mean two statements. **All** statements are terminated by a semi-colon (but not conditional statements).

Notice that we specify the table name, the field names and then the values for those fields. The userID field has been deliberately omitted because it is an integer that automatically increments each time we add (INSERT) a record. We let the database take care of this field for us.

Currently we have two functions. The first function is used to get the data from the user, and the second function to populate the database with the values entered by the user in the first function.

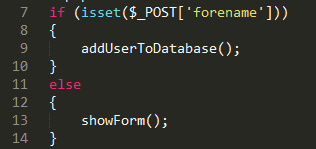
### isset()

PHP has a function called isset() that will check if a variable exists. The function accepts one argument (parameter) which is the name of the variable that we want to check exists. The function will return a value. The value true will be returned if the variable does exist and false if the variable does not exist.

Next we put in place a mechanism for calling the appropriate script at the correct time. We just need to know whether to display the form (invoke the showForm() function) or put the data into the database (invoke the addUserToDatabase() function). If the form has not been submitted then the $\_POST array won’t exist, therefore we can use a PHP function to check if $\_POST exists.

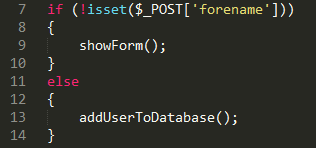
If it doesn’t exist then we need to display the form, if it does exist then the user has already seen the form and clicked the submit button. The use of the isset() function allows us to control the flow of execution.

1. The code on lines 7-28 is not required, delete this code (this is all the code in the main content with the exception of the start and finish PHP tags).
2. Now add the following code:



This reads as: if the posted label forename exists then call (invoke) the function addUserToDatabase(), otherwise show the form to get the forename.

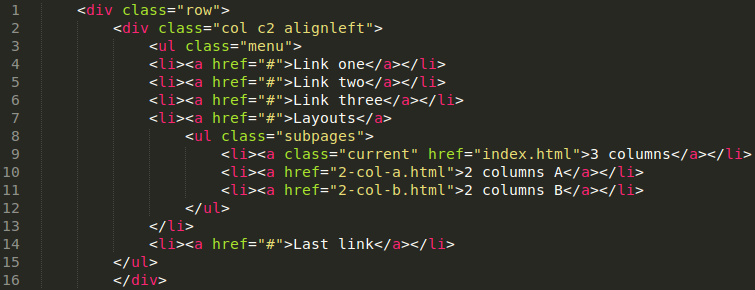
Programmers often prefer to keep things in order, therefore they may add a not (**!**) before the isset() function. The following code demonstrates this.



Notice that we have used a not (**!**) on line 7; this has allowed us to reverse our function calls. This now reads if the posted value forename does **not** exist, then show the form. You can use either method, it really does not matter, but the ! method does make it a little clear which function should be called first.

Next all we need to do is make this page available to the user.

1. Open the leftNav.php file and remove any blank lines. The leftNav.php file should match:



1. Line 4 shows: <li><a href="#">Link one</a></li> replace the text Link one with Register and the # with register.php
2. Save leftNav.php
3. Refresh/reload your index page, you should have link to allow the user to register
4. Create a few test users.

## Retrieving Data from the database

Next let’s look at how we can get data from the database. This can be a little complicated at first, but the more you familiarise yourself with associate multidimensional arrays the easier it will become.

* + Create a connection to the SQL server
  + Create a SQL
  + Execute the SQL
  + Display the result(s)

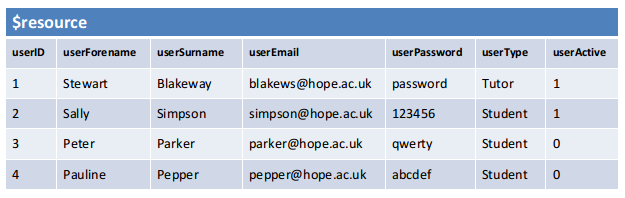
We have already covered points 1 and 3, however point 2 is a little different because we are retrieving data from a database. Let’s take a look at a simple select statement initially.

$sql = "SELECT \* FROM user ";

The above statement once executed will SELECT all columns (**\***) FROM the table called user. This statement will also select all rows. But where does the data go? This is why point 3 is a little different; we need to specify a resource to hold the data.

$resource = mysqli\_query($conn,$sql);

So now we have a $resource with all the records from table user. Let’s visualise what has happened.



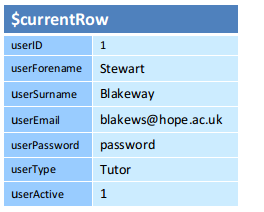
The above image shows what may be stored in $resource, this is all data (all columns and all rows) from the user table.

The next thing to do is to get the first row, this will allow us to access the actual data within the fields. To do this we use a PHP function called mysqli\_fetch\_array().

$currentRow = mysqli\_fetch\_array($resource);

This gets the first row and places the data into a single dimension associate array called $currentRow, this is what has happened:

We have fetched the first row by using the function mysqli\_fetch\_array(). This function needs to know which resource to fetch from, so we specified our $resource. We also had to specify what to do with the row when it was fetched, so we created $currentRow to store that particular row. You can not access the data directly from the resource which is why we have to move each row one at a time into a single dimension associate array.



Now that we have $currentRow we can access the data, for example:

echo ($currentRow[‘userID’]);

will display the value 1. We can change userID to any field name in the table, for example:

echo ($currentRow[‘userType’]);

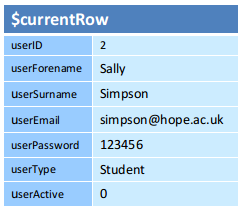
will display the value Tutor.

How do we get the next record? We simply fetch again, each time we fetch a pointer is moved to the next record in $resource.

$currentRow = mysql\_fetch\_array($resource);

Now $currentRow looks like:

We can also loop through the $resource and keep fetching until there is nothing left to fetch:



while ($currentRow = mysqli\_fetch\_array($resource))

{

}

We could display the retrieved data by placing the echo() within the body of the loop, i.e. between the { and }.

We will create our first tutor page to display all users in the database. Later we will customise to show only unauthorised students.

1. Open index.php and save as tutorShowUsers.php
2. Enter the following code in the main content area (lines 6-9)



I hope you are becoming comfortable with most of the code above. Line 7 defines the connection to the database server. Line 8 constructs an SQL statement to get all columns and all rows from the table called user.

Let’s look closely at the subtle differences. Firstly you will notice that the if condition on line 10 has been prefixed with a variable called $resource: this can be read as query the database using the connection string and SQL defined earlier. If the SQL is a valid SQL statement put the return value from the database into a resource called $resource and then enter the true part of the if (lines 12 and 13).

Line 13 is a function call and will call the function called display(). What is different about the display function call is that it sends additional information to the display function. The function definition (line 21) has a variable between the parenthesis.

The data passed to the display function will be copied into this variable. This data are the records that was SELECTED from the database.

Lines 21-30 contain the function, the function accepts the additional information that was passed to it when the function was called and copies this into a variable called $resource. Line 23 fetches from the $resource one line at a time and displays the information contained in $currentLine. Let’s add this new page to our navigation menu.

1. One the file leftNav.php
2. Change the line 5: <li><a href="#">Link two</a></li> to reflect your new page.

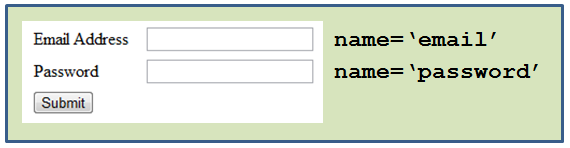
We plan to restrict access to the page that we have just created so that only tutors can access this page. In order to do this we need to get the credentials of the user and then authenticate them. Most systems do this using a username and a password. We will follow suit.

## Authentication and Access Control

Authentication is any process which obtains the credentials of a user (or system) and verifies that the credentials obtained are valid for that entity. This usually involves a username and a password, but there are many other credentials (e.g. Biometric authentication may use fingerprints or face recognition).

Access control is the control of resources to authorised users only; a resource could be a webpage, a file or other entities. In this section we continue our introduction to database administration by creating a login page to get the users credentials, check the credentials and then set access control settings based on the type of user they are (tutor or student).

We will create a new page called login.php, this page will serve three purposes, firstly to get login details, secondly to construct and execute the SQL based on the details entered in the form and then thirdly to authenticate the login details. Getting the login details if fairly straight forward and involves creating the form below, pay particular attention to the name attributes for each text box. Also ensure you set the action attribute to login.php in the form tag.



The page is designed to serve three purposes, so this is an ideal time to use functions. The form that we will create will be contained within a function called showLogin(), the function will not receive any arguments (an argument is extra information sent to the function, like when we sent the resource to the showUsers() function previously). The second function that we will create will take the data from the from and use this to query the database. The third function will check the resource that is returned from the database.

## Functions

A function is often referred to as a block of code or a set of instructions. The instructions inside a function will quite happily sit there and do nothing. The instructions are not executed by the interpreter, that is until there has been a function call.

Lets take a look at an example. We have a function called echo(), the code for the echo() function allows you to output text to the web browser, yet the code for the echo() function does not execute until we call the echo() function. The echo() function is a built in function and is part of the PHP scripting language, there are over 5000 built in functions.

A function should serve a specific purpose and not try to achieve too many things, therefore a function should serve a given goal. If the overall goal is complex you can break it down into several functions, each serving part of the overall goal. There are three components to a function, two of which are optional. A function prototype (signature) looks like:

Return FunctionName Parameters.

Not all functions return a value and not all functions accept additional parameters (called arguments). For example the showForm() function we created earlier. As you continue to work through this document you will be exposed to several examples of simple functions, functions that accept additional information and functions that return values.

## The PHP echo function and HTML forms

The fact that we are using a PHP function means that the form should be within an echo(), every time we want to use a HTML tag in PHP it must be within echo(). Therefore placing your HTML form within an echo function.

echo(" form goes here ");

Change any " within the form to ' . However, do not change the double quotes that relate to the echo(). We are doing this because the " indicate where the string inside an echo() starts and where the string in the echo ends. If we didn’t change the double quotes within the form the echo() function would end prematurely at the second double quote encountered.

Then we will create a function to serve the second purpose of the page: to construct and execute the SQL. This function will be called doLogin() and will not accept any arguments but will return data from the database.

The third function will check the values that were returned from the database. If there was a match then the user successfully logged in, otherwise the login will fail. Let’s take a quick look at the overview of a simple authentication process and then apply PHP.

* + Get username and password
  + Get user from table users where username and password match
  + Was there a match?
    - If there was a match then set access control settings
    - If there was not a match display login failed

Getting the username and password has been accomplished by creating the form, when the form is submitted the form data is sent to the same page. We can access this data by using $\_POST[‘email’] and $\_POST[‘password’].

Let’s recall how to SELECT records from a database?

* + Create a connection to the SQL server
  + Create a SQL
  + Execute the SQL

Let’s take a look at the SQL in step 2. In the previous example we used a SQL SELECT statement to get all the records and all the columns from the user table that resides in the database. This time we need to be more specific about what we want. Assume we only want to return a record of a particular user which consists of the username and password that they entered in the form.

$sql = "SELECT \* FROM user WHERE userEmail = '$\_POST[email]' AND userPassword = '$\_POST[password]'";

The WHERE clause here is used to tell the database to return only those records that have the userEmail field and the userPassword field matching the data that is contained in the posted variables email and password (from the login form). The very fact that we used AND means that both userEmail and userPassword must match.

Next we execute the SQL. We need to account for any data that may be retrieved from the database because we are using a SELECT statement. The $resource variable will hold the data returned from the database.

$resource = mysqli\_query($conn,$sql);

To determine if the user attempting to login is a valid user we check the contents of the $resource variable. We could use the mysqli\_fetch\_array() function and then check the username is not null.

Alternatively we could simply count how many rows have been returned to the resource (there should be 1 or 0). To count the number of rows in a resource we use the function mysqli\_num\_rows() and specify the resource for which we are counting the rows, theoretically we could be working with several resources at once. This is an example of a function that accepts both an argument (the resource to be counted) and returns a value. The return value is how many rows are inside the resource.

if (mysqli\_num\_rows($resource) == 1)

In PHP any statement that can be evaluated to either true or false (but nothing else) can be used within a conditional statement. Above we are using the return value from the mysqli\_num\_rows() function and comparing the equality to the constant value 1. This reads: if the number of rows inside resource is equal to 1 then do something (the something would be between the { } which signify the start of the true section for the if and the end of the true section respectively).

Thus we can set user access controls, we achieve this by setting the userType value to a session variable. The userType is a field within the database, the value for userType is inside the $resource variable. By creating a session variable we can retain the value inside userType thus remember the type of user as they move around the website.

Setting a session variable is relatively easy but before we do that we have to get the data from the resource. How do we do this?

$currentRow = mysqli\_fetch\_array($resource);

The above statement will fetch a row from the resource. The resource is the data that was returned from the database. Since currently the resource contains only one row we know it’s the correct one and we are able to set the session variable:

$\_SESSION['userType'] = $currentRow['userType'];

At the same time, it’s worth also recording the identification number of the current user for future use.

$\_SESSION['userID'] = $currentRow['userID'];

Thus we have completed the authentication for the current valid user and created two session variables to retain the user ID for the user that is logged in and the user type. In the next section we look at how we can put user type to good use by controlling the access the certain areas of the website. First we should deal with the alternative eventuality of the authentication process.

So far we have used the ‘if’ statement which can only take the values true or false. Let’s now get ‘else’ involved as it can deal with the false case.

The condition that is being evaluated to produce the true or false value is: mysqli\_num\_rows($resource) == 1 which as described above counts how many rows has been returned from the database and placed inside resource. If there are not any rows in the $resource (i.e. An empty record set was returned from the database) then we should inform the user that login was unsuccessful. The following code achieves this:

else

{

echo ("<br /> login --- FAIL");

}

By now we have three functions called: showLogin(), doLogin() and checkUser(). Which one do we call and when? We check if the form has been submitted: if no, then we call the showLogin() function which allows the user to enter there credentials and submit the form. The page at this point reloads and the same check (has the form been submitted) is performed: if yes, then we call the doLogin() function to query the database. The doLogin() function the takes responsibility to pass the returned data from the database to the checkUser() function. The checkUser() function counts the number of rows in the resource and performs the appropriate action dependent on the number of rows. How do you know if the form has been submitted?

Each time a form is submitted, the data contained on the form is submitted, therefore if $\_POST[‘email’] exists then the form must have been submitted. If $\_POST[‘email’] does not exist then the form has not been submitted. We use a function called isset() to check if something exists in PHP. We can reverse the function by putting a **!** in front of the isset() function (i.e. !isset), this essentially reads if not set. The isset() function requires one argument (what we are checking exists).

if (!isset($\_POST['email']))

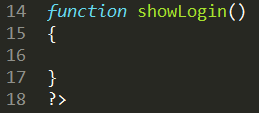
{

showLogin();

}

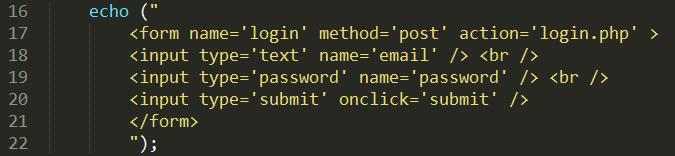
Let’s put this information to good use.

1. Open the index.php file and save as login.php
2. Create the showLogin() function on line 14 (do not delete the closing PHP tag, simply move it down a few lines). Your code should look like:



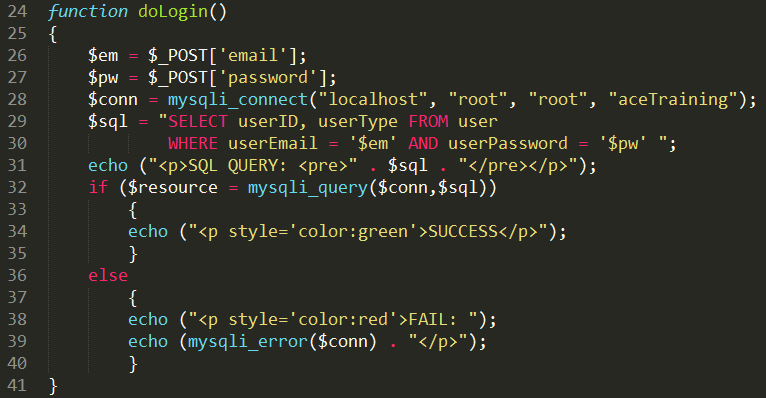
You will notice that the function starts on line 15 (identified by the open curly brace) and finishes on line 17 (identified by the close curly brace). All code relating to this function must be within the curly braces, we can of course move the curly brace on line 17 down as we add new lines of code.

1. Enter the following code in the showLogin() function:



The echo above will output the login form. Next we plan to process the data entered into the form and finally check that the credentials are valid.

1. Enter the following code:

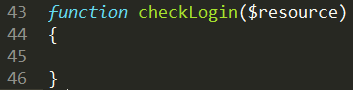


The above code does not actually authenticate. It simply queries the database and outputs if our SQL statement was correct or not. You should be able to recognise most of the lines.

We will create a new function to check if the user is a valid user. Before this however, notice line 32 creates a $resource based on the result of executing the SQL statement. The SQL statement being correct does not mean that a row has been returned from the database. A correct SQL statement that finds no matching criteria in the database returns an empty record set.

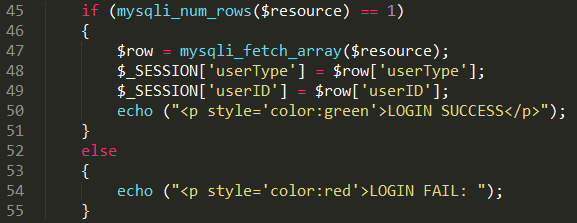
This $resource only exists within this function (when the function finishes or if we call another function the $resource will not exist in the other function). We will allow our new function to accept a copy of this $resource as an argument so that it can be checked if a row or an empty record set was returned from the database.

1. Enter the following code:



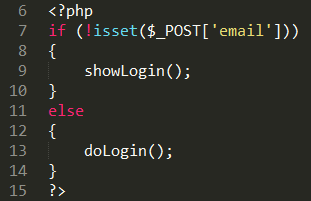
Notice that we have coded this function to accept additional information when called. We are going to send a copy of the $resource that was created on line 32 to this function so that it can be checked for a user.

1. Enter the following code inside the function:



Next we need to add some flow control to call the correct function at the appropriate time.

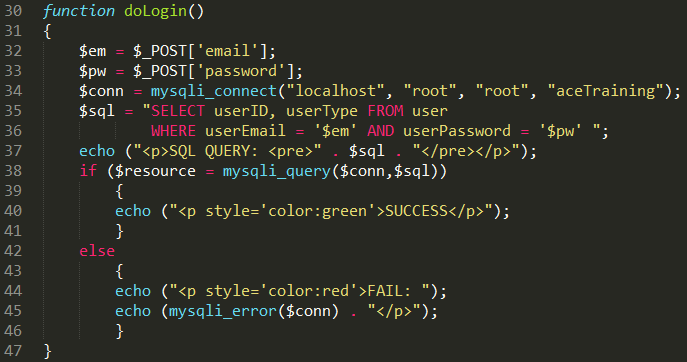
1. Lines 6-9 between the divs represent our main content area (leave the divs intact). Replace the current lines of code between the divs with the following code:



You may be wondering about the third function and you would be correct to do so. When do we check the login? We could do this immediately after the doLogin() function which is on line 13, however, this function is designed to ensure our SQL statement was correct. We would not want to call the checkLogin($resource) function if the SQL was incorrect. In addition to this remember that the $resource (information from the database) only exists inside the doLogin() function. This would be lost after completing line 13 so would not be available at line 14.

Therefore we will call the checkLogin() function from inside our doLogin() function, but only if the SQL was correct.

If you have adhered to the same format as the examples provided, then the doLogin() function is now declared on line 30. The if condition, that checks if the SQL is correct, is on line 38, and the code to be executed if the SQL is correct is between lines 39-41. See below:



1. Immediately after the echo() function on line 40 add the following line: **checkLogin($resource);**

This will call the checkLogin() function, notice that the $resource is between the parenthesis. We are essentially saying the resource that exists in this function, send a copy to the checkLogin() function.

Before we finish we need to discuss sessions a little further.

## Session Variables

When you are working with an application, you open it, do some changes and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But there is one problem on the internet: the web server does not know who you are and what you do because the HTTP address doesn't maintain state.

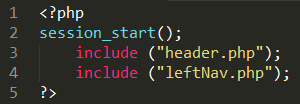
A PHP session solves this problem by allowing you to store user information on the server for later use (i.e. username, shopping items, etc). However, session information is temporary and will be deleted after the user has left the website.

Sessions create a unique id (UID) for each visitor and allow the creation of variables that hold data based on this UID. The UID is either stored in a cookie or is propagated in the URL.

Before you can store user information in your PHP session, you must first **start up the session**.

Note: The session\_start() function must appear BEFORE the <html> tag, i.e. starting a session **must** be the first thing you do! Each and every page that requires access to the session variables (i.e. the type of user or the user ID) must have a session\_start() at the top of the page. In our scenario all the tutor pages must ensure that the userType is a tutor so will need to start a session.

1. Go to the very top of the login.php text document, immediately after the start PHP tag **<?php** and before the first include enter the code for to start the session. The top of the document should look like:



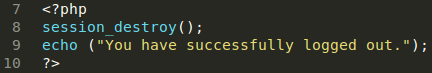
Thus the login and the authentication page is complete. We now need to add it to our navigation menu.

1. Open leftNav.php
2. Find Link 3 (line 6) and replace: <li><a href="#">Link three</a></li> with: <li><a href="login.php">Login</a></li>
3. Save leftNav.php and now test it. Register a user and then login as that user.

## Logging Out

The code for logging out of the website is extremely easy in comparison to logging in. We only need one instruction to logout and no checks are necessary. Once we have created the logout.php page we will update our left navigation menu to show either the login or the logout link but never both at the same time.

1. Open index.php and save as logout.php.
2. Remove all the lines between the open <div> and the close </div>
3. The add the following code between the <div> tags.

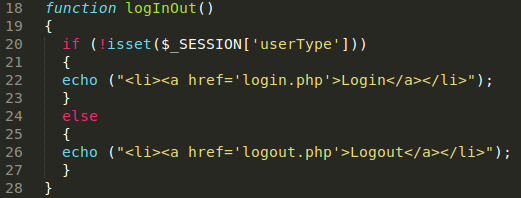


Next we create the link so that the user can access the logout page. We should not show the logout link unless the user is logged in. Also we should not show the login link if the user is already logged in.

1. Open leftNav.php and replace the line: <li><a href="login.php">Login</a></li> with the following code: <?php logInOut(); ?>

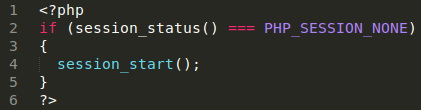
The code will call a PHP function called logInOut(), the purpose of this function is relatively simple. If the user is logged in then display a link to log out. If the user is not logged in then display a link to log in. When a user logs in we create two session variables: $\_SESSION[‘userType’] and $\_SESSION[‘userID’]. We will use one of the session variables to determine if the user is logged in or not logged in.

1. At the very end of the leftNav.php document, after the final closing </div> add the following code:



All that remains is to start a session. Remember was are not allowed to access session variables unless we start a session. You may recall that we start a session by using the session\_start() function. This function will start the session as discussed earlier, however, this page is an external file to be included within a main file. What if the session had already been started in another page? This is very likely as every page currently includes the left navigation menu. Let’s change the code slightly so that it checks a session has not already started before starting this session.

1. Edit leftNav.php so that the following code is included at the very top of the page:



We will use this script for any files that are included into a main document and uses session variables.

## Dynamic Pages

Once a user has logged in the user should have the option to navigate to an appropriate page that has been tailored for their needs. In this section we will see how to show a specific link depending on what type of user logged in.

This process is similar to what we covered in the previous section when we showed either the login link or the logout link depending on if the user was logged in or not. However, in this case not only are we concerned if a session variable has been set or not, but what the value of the session variable is.

The most appropriate time to show the user a link to a page that has been tailored for their needs is when the user logs in. Thus we will work with the login.phppage.

We know the user type because when the user registered this information was stored in the database. When the user logs in the value in the userType field is copied into a session variable.

The following code checks to see if the value of the $\_SESSION[‘userType’] variable is equal to tutor. If it is a hyper-link is displayed that will allow the tutor to navigate to a new page called tutorHome.php.

if ($\_SESSION['userType'] == "tutor")

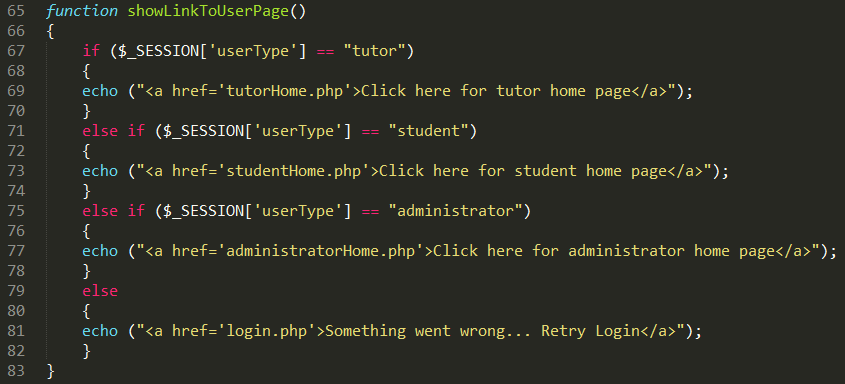
{

echo ("<a href='tutorHome.php'>Click here for tutor home page</a>");

}

You will need to add similar code for the student and the administrator. This code could be placed in the checkLogin() function, specifically after the echo displaying LOGIN SUCCESS. However, this is not strictly checking the login, it’s displaying a link to the users home page, therefore we will make use of a function instead (this also keeps our code in nice manageable chunks).

1. Open the login.php file.
2. Locate the final PHP close tag, and the curly brace that signifies the end of the checkLogin($resource) function. Between these elements create this function:



Now we need to call the function just created to display the correct link dependent on the type of user (userType) logged in.

1. Directly under: echo (“p style=’color:green’>SUCCESS</p>”); add the following function call: showLinkToUserPage();

# Tutor Pages

This section will detail what we expect from the tutor pages. Remember that we set a session variable when the user logged in, this will allow us to make sure an unauthorised user (student or somebody not logged in) has not somehow navigated to the tutor pages. As a minimum the tutor must be able to:

* + create a new course
  + upload resources
  + authorise students to their course
  + manually add a student to their course
  + add a list of students (from a text file) to their course

Over the next few pages we will address each of these requirements starting with a requirement for all tutor pages: Access Control.

## Access Control

The very fact that we set a session variable in the doLogin() function means that we recorded the type of user when the user logs in (Tutor or Student). This makes checking access for controlled pages very easy. We will create a new page called tutorHome.php, because we are using sessions we need to ensure that the first line calls the function session\_start() as you do with all pages that require access to session variables.

The tutorHome.php page is to become the page that welcomes the tutor and presents tutor options. This will primarily allow for tutor functionality, but before we do this we have to ensure access to this page has been controlled and only tutors (or administrators) have access. In order to do this we need to do three things:

* + Check the user has logged in
  + Check the user is not a student (tutors and administrators can access the page)
  + If not a student display the tutors page, else display a suitable not authorised message

Checking the user has logged in can be accomplished by checking if a session variable exists (we only create the session variables if login is successful). We know from the earlier section that we can check something exists by using the isset() function.

if (!isset($\_SESSION['userType']))

{

// user must not be logged in!

}

Next we need to check to see if the user is a student or not, we can do this by using the data stored in userType.

if ($\_SESSION['userType'] == "student")

{

// user is a student!

}

If either of the above if statements equate to true, then we display a suitable not authorised message. The fact that we do the same thing for each if means we can combine the conditions into a single if:

if (!isset($\_SESSION['userType']) or $\_SESSION['userType'] == "student")

{

echo ("you are not authorised to view this page");

die();

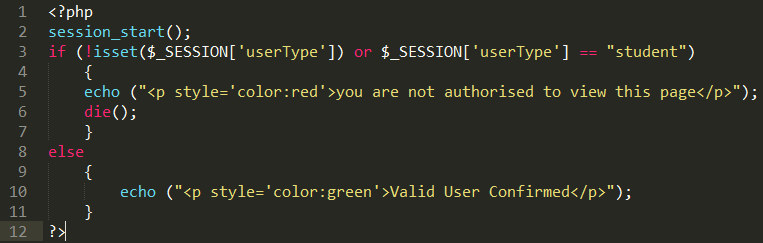
}

Notice that keyword or means if either condition evaluates to true then the entire condition will evaluate to true. We saw something similar to this in an SQL statement when authenticating a student, however, we used and instead of or. The difference is that when using and both conditions must be met for the entire condition to be true. When using or, either condition can be met for the entire condition to be true.

Note that we are using die(), remember that this function will stop the PHP parsing in its tracks.

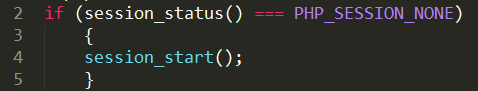
We could create a function to enclose the above code, however, every single tutor restricted page will need to make use this code. Thus, in this case it would be better to have the code as a separate file and include into the tutorHome.php document.

1. Open index.php and save as tutorCheck.php
2. **Delete all** code
3. Add the following code:



There is a slight issue with line 2. This line will start the session as discussed earlier, however, this page is an external file to be included within a main file. What if the session had already been started in tutorHome.php? Let’s change the code slightly so that it checks a session has not already started before starting this session. You may recall we did this for leftNav.php.

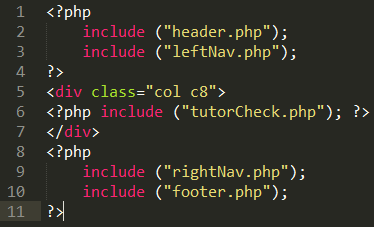
1. Edit the script so that the first part of the code reads:



We will use this script for any files that are included into a main document and uses session variables.

Next we need to create a tutorHome.php page and include this file.

1. Open index.php and resave as tutorHome.php
2. Remove the code in the main content (lines 6-9).
3. Now add the following line <?php include ("tutorCheck.php"); ?> Your code should look like:



Do not try and add a vertical line after the ?> on line 11. This is just the flashing cursor!

Including the tutorCheck.php takes care of access control; we always show the not authorised message if the usertype is a student or if the user has not logged in. Otherwise (else) we show a message Valid User Confirmed.

## Tutors Options

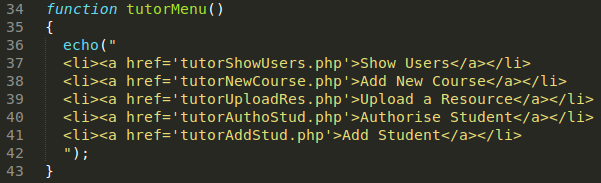
We stated earlier that as a minimum the tutor should be able to:

* + create a new course
  + upload resources
  + authorise students to their course
  + manually add a student to their course
  + add a list of students (from a text file) to their course

In this section we will customise our left navigation to show the tutor (and only the tutor) the available options. At the same time we will remove the Show Users option we created earlier from the main navigation to the navigation that only the tutor can see.

We start by creating a new function immediately after the logInOut() function. We will call this function tutorMenu(). Then we move all the html links that belong to the tutor into this function (and create a few additional ones), remember that we must enclose all HTML within an echo.

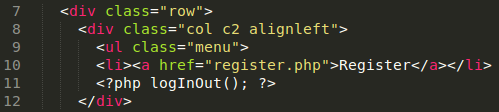
1. Open the text document leftNav.php.
2. Locate the closing curly brace for the logInOut() function and the closing PHP tag. Between these elements create the following function:



You may have noticed that I have moved the Show Users link into the echo function too (this is line 37 above and was originally on line 11).

1. Remove all links that we are not using and tidy up the indentation, do **not** remove any <div> tags.

The section immediately after the closing curly brace that signifies the end of the conditional if statement should look similar to this:



The start PHP tag that contains the two functions logInOut() and tutorMenu() now starts at line 13.

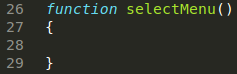
Now we have two common links that everybody can see (lines 10 and 11) and some hidden links inside a function further down the page for the tutors eyes only. The trick is to check which user is logged on (i.e. tutor) and load the appropriate menu for that user.

We do this by checking the value of the userType session variable. If the userType value is equal to ‘tutor’ call the tutorMenu() function.

1. Locate the function call for the logInOut() function. In my code this is on line 11 (see code screen shot above).
2. Immediately after the logInOut() function call create a new function call that will call a function named selectMenu() be sure to extend your closing PHP tag. Your code should look similar to:

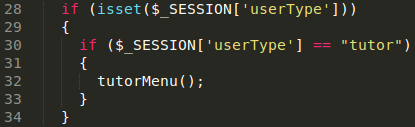


1. After the closing curly brace that signifies the end of the logInOut() function create a new function called selectMenu();



This function will contain the code that firstly checks a session variable userType exists. If the session variable does exist another check will be performed to see if the value of the session variable userType is equal to tutor. If this is it will call the tutorMenu() function.

1. Add the following code to the selectMenu() function.



We could have combined our conditions like we did earlier, for example: if (isset($\_SESSION[‘userType’]) and $\_SESSION[‘userType’] == “tutor”).

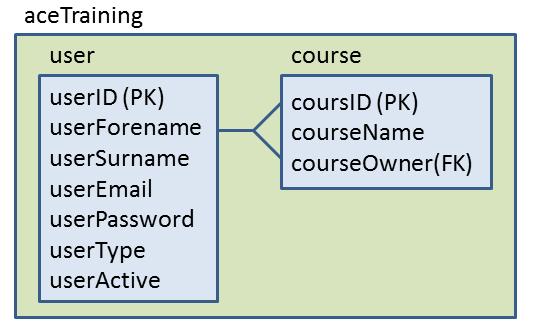
We chose not to combine the conditions for the purpose of readability when we come to add menus for the student and administrator later.

That’s it for now in regards to the navigation menu. You should test to ensure that all is working ok.

1. Refresh/Reload the index.php page. Log in as a tutor, you should see a link that will allow the tutor to navigate to the tutor home page, when the tutor clicks on the link the additional menu options will be available.
2. Log in as a student. Click the link that takes the student to their home page. You should then only see two links.

## Tutor Add Course

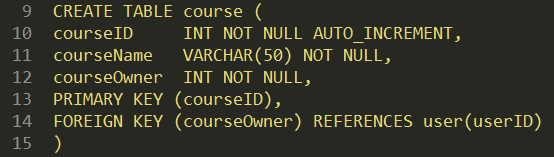
In this section we will create the page that will allow a tutor to add a course. We will keep the course table as minimalistic as possible.



1. Open createUserTable.php and save as createCourseTable.php.

Once you understand the fundamentals of PHP and MySQL it is very easy to quickly progress. The same piece of code can be used frequently, for example when creating a course we can use most of the code that we used to create the user table (we just need to tweak the SQL statement to reflect the table and field names).

1. Replace the SQL (lines 9-16) so that it matches:

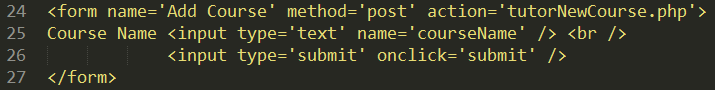


Now let’s add a link to our right navigation menu.

1. Open rightNav.php
2. Change: <a href="#">Sample link 3</a>   
   to: <a href="createCourseTable.php">Create Course Table</a>
3. Reload/Refresh index.php and click the Create Course link to ensure the table creates without any issues.

Now since we have a table we can allow the user to add a course. We do this in two stages. Firstly get the course information and secondly add the course information to the database. This is similar to register, when registering we get the user information, then add the user to the database.

1. Open the register.php file and save as tutorNewCourse.php
2. Locate the showForm() function. An echo() function that displays a form is inside this function.
3. Change the code within the echo() function so that it matches:



*Pay particular attention to the action attribute on line 24 and the name attribute on line 25.*

1. Find the function called addUserToDatabase() and rename to addCourseToDatabase()
2. Remove the first 4 lines of the code within the function called addCourseToDatabase(). The first line of code within the function should be: **$tp = $\_POST['type'];**
3. Change **$tp = $\_POST['type'];** to **$cn = $\_POST['courseName'];**
4. Immediately after the line that you altered add the following line: **$ow = $\_SESSION['userID'];**

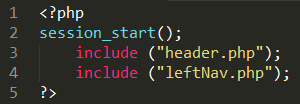
The last thing we need to change in the addCourseToDatabase() function is the SQL statement.

1. Change the SQL statement so that it matches:



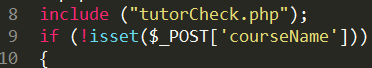
You may have noticed that we are using a session variable to record the owner (creator) of the course. Therefore, we need to start a session.

1. Ensure the very top of the document matches:



This page should be restricted to tutors only (and administrators), therefore we need to include our tutorCheck.php file. We don’t want to show the form, or add data if the user is not authorised.

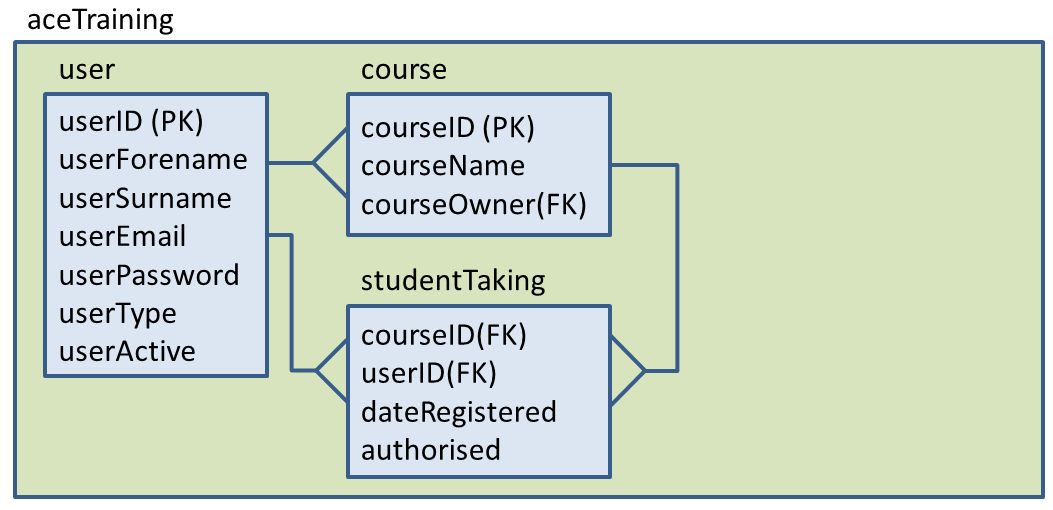
1. Add line 8 of the following code just before the if statement as indicated:



1. Change the label in the $\_POST array to courseName as depicted above.
2. Save and Test (we have already created the link in the tutor menu).

# Student Enrol for Course

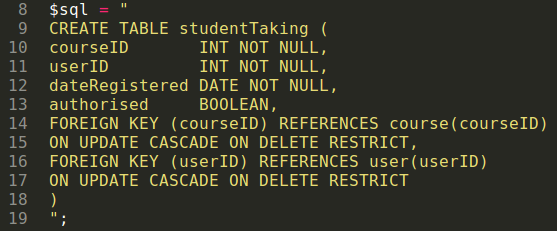
We have created the option to allow the tutor to add a new course. It might be the most appropriate time to allow a student to enrol onto a course. Our initial design did not allow for this so we have adapted by adding a new table between course and user. This allows for a student to register on a course(s) and also allows for the tutor to authorise student(s) on to their course.



1. Open createCourseTable.php and save as createStudentTakingTable.php

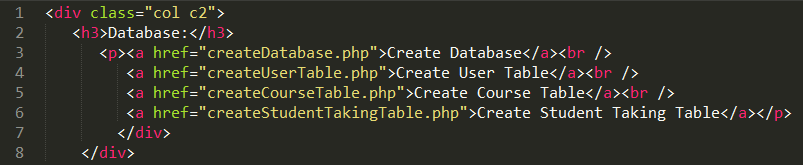
The only line we need to change in this file to accommodate a new table is the $sql statement.

1. Alter the SQL statement so that your code reflects:



Next we will create a link to this file from our right navigation menu.

1. Open the file rightNav.php
2. After the hyper-link to create the course table, add a new hyper-link that links to the create student taking table. Your code should look like:



1. Save and test that you new hyper-link is working.

Now that we have our structure in place it is time to allow a student to register (enrol) for a course. We do this in stages:

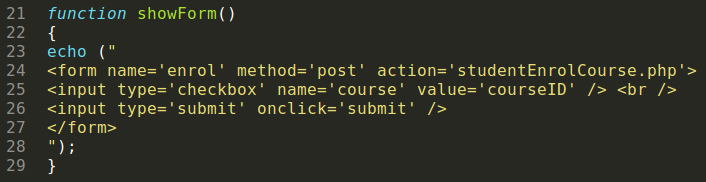
* + Show the available courses with checkboxes
  + Send the students selections to be processed
  + Update the database

This structure is similar to register.php so we will use this file as our basis.

1. Open register.php and save as studentEnrolCourse.php
2. Locate the function called showForm()

You will notice that the HTML form within the echo() function has various text boxes. We don’t need any text boxes so we will remove these and create a check box.

1. Remove all the form elements (except the submit button) and add a checkbox.
2. Change the action attribute to studentEnrolCourse.php.
3. Change the form name to enrol. Your function should look like:



Before continuing let’s look at lines 24-28.

* Line 24 creates a form and specifies what to do with the data
* Line 25 displays a checkbox for a course
* Line 26 displays a submit button
* Line 27 closes the form

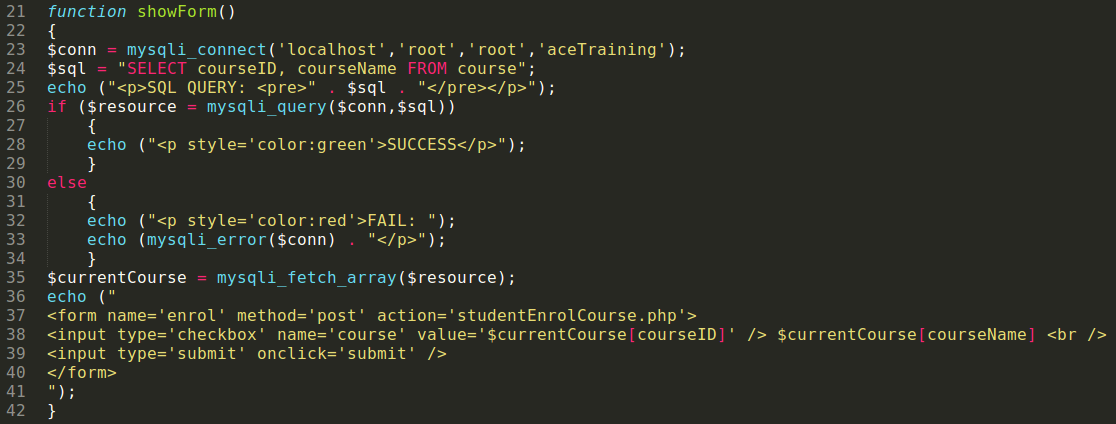
Relatively straight forward, the difficulty we run into is that there could be more than one course. Also the course details are in the course table in the database. Before we concern ourselves with multiple courses we will attempt to display one course.

1. Just before the echo() function add the code to create a connection string to the database
2. Create the SQL to select courseID and courseName for all records from the course table.
3. Execute the SQL and place the results into a resource called $resource
4. Fetch the first line from $resource and place into **$currentCourse**
5. Change the value attribute of the checkbox from **courseID** to **$currentCourse[courseID]**

As well as displaying a checkbox, it would be desirable to display the course name that the checkbox relates to.

1. Immediately after the checkbox close tag but before the <br /> add **$currentCourse[courseName]**

Here is what your code should look like:



Let’s test what we have done so far. Before that, however, we need to alter our flow control. The if statement is looking for forename, since we have deleted this, forename does not exist.

1. Change !isset($\_POST['forename']) to !isset($\_POST['course'])
2. Save the text document
3. If you have not already done so, create a tutor (register)
4. Login as a tutor
5. Add two courses: Website Development and Database Technology
6. If you have not already done so, create a student (register)

We could now login as a student, however, the student does not yet have a home page. Not only this but the student does not have their custom navigation links. We will rectify this before attempting to log on as a student.

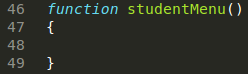
1. Open tutorHome.php and save as studentHome.php
2. Change the line that includes the tutorCheck.php file to studentCheck.php

Now the student has a home page. Next we need to ensure that only students can access this page.

1. Open the file tutorCheck.php and save as studentCheck.php
2. Locate the line: if (!isset($\_SESSION['userType']) or $\_SESSION['userType'] == "student") and change to: if (!isset($\_SESSION['userType']) or $\_SESSION['userType'] != "student")

The final action is to give the student some options; we will start by giving the student the option to enrol onto a course(s).

1. Open leftNav.php
2. Immediately after the function tutorMenu() create a function called studentMenu(). Remember the function end is at the closing curly brace }. Your code should match the following:



1. Create a single HTML link within the studentMenu() function. The fact that we are using HTML means that we have to enclose the link within an echo() function, similar to that of the links in the tutorMenu() function. The new link should have the label: “Enrol on Course” and the link will be: studentEnrolCourse.php. Your code should match:

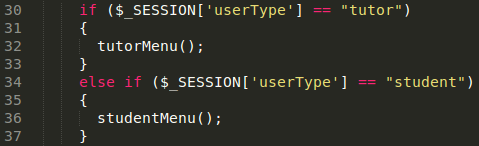


Now we have created a function that will display the student’s options when a student logs in to the system. At the moment however we are not invoking (calling) this function.

1. Locate the conditional if statement that displays the calls tutorMenu(). In my code this statement is on line 32 and is inside the selectMenu() function.

We want a similar statement to check for a user that is a student. We will extend the if statement to become an else if.

1. Amend the if statement that checks if the session variable userType is equal to a tutor so that it becomes and if else statement. This way we can check if the variable userType is equal to a student should it not be equal to a tutor. Your code should match:

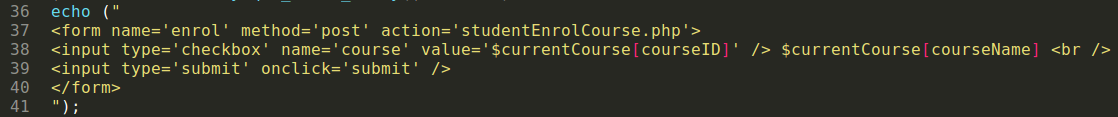


1. Login into the website as a student to check the new hyper-link to enrol onto a course is working as intended.

So far our page only displays the first course where there may be many. To deal with multiple courses we have to break thing up a little and create a loop to show the other courses. We want to avoid creating a new form for each course.

1. Open the file studentEnrolCourse.php

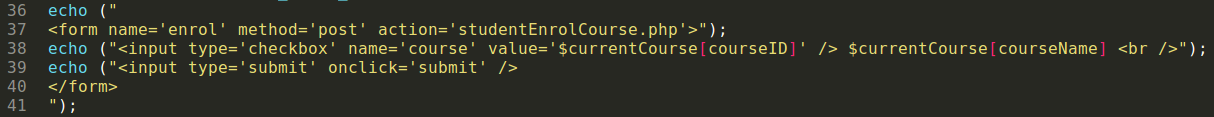
Currently the form looks like this:



The aim is to split up this statement so that the only repeating line is the actual checkbox and course information.

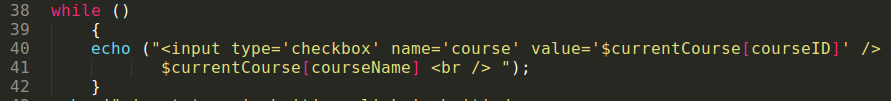
1. At the end of line 37, where the start form tag closes, close your echo() function.
2. Create a new echo() function on the next line to handle the checkbox type
3. Add a new echo() function on the next line.

Your code should result in this:



Now the trick is to do the first echo() function, then loop the second echo() function for as many courses exist and then finally close the form.

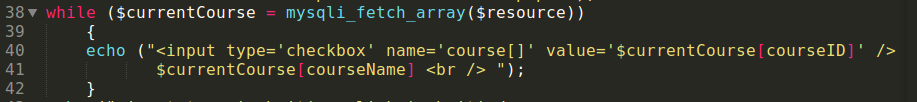
1. Locate the line where you fetch a row from the array called $resource and put inside $currentCourse.
2. Create a while loop that starts after the first echo() function, contains the second echo() function, and finishes before the third echo() function. In my example I have split line 40 over two lines for readability. Your code should look like:



The final step is to make the function that fetches a row from the resource part of the condition.

1. Locate the code: $currentCourse = mysqli\_fetch\_array($resource);
2. Move this entire line between the parenthesis of the conditional if statement.
3. Because this is now a condition being evaluated we need to remove the end of line terminator. This simply means remove the ;

The resulting code should look like:



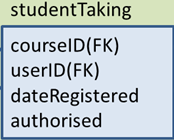
The above code can be read as: while there are rows to fetch from the array called resource, fetch a row and put it inside current course. Each time you fetch move the pointer to the next row, so each time you fetch the next row. For each row, output a checkbox with a value set to the courseID and also output the course name.

1. If you look closely at line 40, you will also notice that I have used [] after the value for name. This is required because we may post multiple values (i.e. the user can select more than one course) within the same name. Update your code to reflect this.
2. Now test your code, you should be able to see all the courses instead of just the first.

Now the student users are able to select the courses they wish to enrol onto. We also need to add this information into the database. In studentEnrolCourse.php file you will see a function called addUserToDatabase(), we will now finish the course enrolment code by updating this function.

1. Change the name of the function addUserToDatabase to addEnrolmentToDatabase()
2. We only have one posted field from the form which is course[] so replace the first 5 lines of this function to this: $course = $\_POST['course'];

The table on the left shows the structure for the studentTaking table. We will populate this table once the student has selected the course(s).

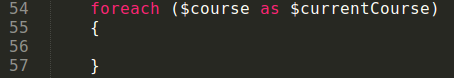


We have the courseID’s inside course[]. We have the userID inside a session variable that was created when the user logged in. The date registered on can be obtained from the system. Initially we set the authorised to 0 (false). The students are not authorised unless the tutor changes the value to 1 (true).

1. Add the following: **$userID = $\_SESSION['userID'];** immediately after $course = $\_POST['course'];
2. In order to get the date we use the date function, type in the following: **$today = date("Ymd");** immediately after **$userID = $\_SESSION['userID'];**

Remember that $course could contain many values (one for each course ticked). We used the [] brackets earlier to indicate there may be multiple values and so to treat course as an array. Therefore we need to create a simple loop to loop through the array.

1. Directly under the connection string: mysqli\_connect ('localhost', 'root', 'root', 'aceTraining'); add a foreach loop. A foreach loop allows us to loop through an array and can be read as: for each item in course put into currentCourse and do the actions between the { and }.



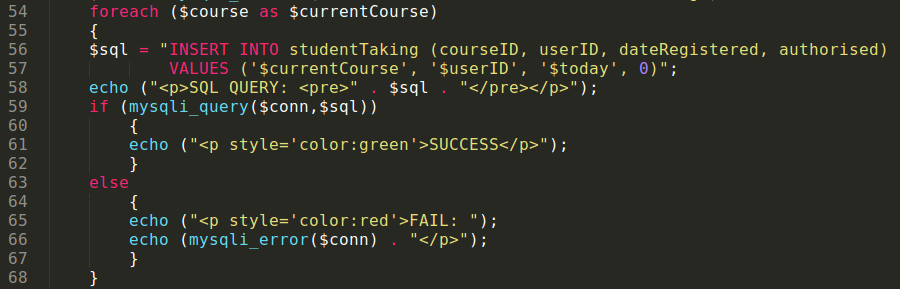
Next we update the SQL statement to put the information from the form into the database.

1. Change the SQL statement so that it matches:



Now we put the appropriate code within the foreach loop.

1. Select all the lines of code from and including the SQL statement down to the end of the end of the function and place inside the foreach loop. Your foreach should look like:



Earlier we changed the name of the function from addUserToDatabase to addEnrolmentToDatabase, therefore we need to update the function call in the conditional if statement (the else section).

1. Near the top of the code find **addUserToDatabse()** and change to: **addEnrolmentToDatabase()**

We are dealing with a student page. You may have noticed a convention that we have been using throughout this document. All my tutor pages start with tutor, i.e. tutorAddCourse.php and all my student pages start with student, i.e. studentEnrolCourse.php. This makes it very easy to distinguish between the files belonging to which group, because we can easily check that the user viewing this page is of the correct type. Therefore, all my files that are prefixed with student will have an include(“studentCheck.php”); and all my files prefixed with tutor will have an include(“tutorCheck.php”).

1. Immediately after the PHP start tag in the main content section (just before the **if(!isset($\_POST[‘course’]))** ) add **include ("studentCheck.php");**

This concludes this section but before moving on it is worthwhile pointing out that although we have a functioning set of webpages that are some improvements that could be made.

* All courses are displayed, even if the student has already enrolled for a particular course
* Thus a student could enrol onto the same course several times, even if they have already completed that course!

## Showing and authorising waiting students

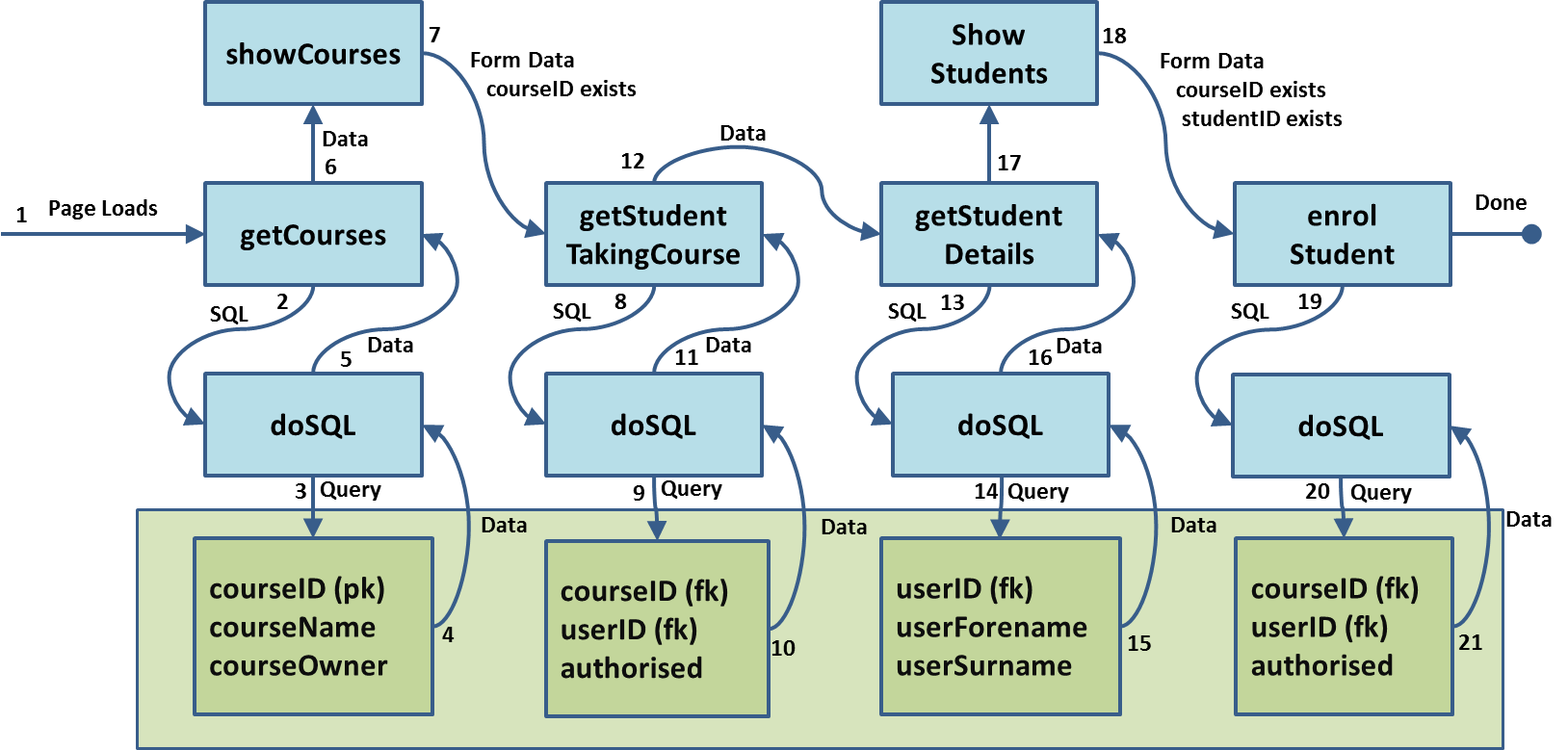
This section will allow the tutor to see a list of students waiting to be authorised. There are several ways of tackling this. For example, we could show all the students waiting to be authorised, we could show all the students waiting to be authorised by a particular tutor (the tutor logged on), or we could ask the tutor that is logged on which course they wish to authorise students for (from a restricted list of the courses that belong to them). Alternatively we could do it from a student approach, i.e. select a particular student and then show the courses they are waiting to be authorised for.

The approach that this document will take is:

* + Display a list of courses that belong to the tutor that is logged on
  + The tutor will select a course
  + A list of students waiting to be authorised for that course will be shown
  + Select required student(s)
  + Authorise students

The difficulty is that in order to authorise a student onto a particular course we need to interact several times with the database, this is further complicated by the fact that at different times we interact with different tables.

Consider the following flow diagram.



The page loads and calls a function to get the courses that belong to the tutor that is logged on. The purpose of this function is just to construct the SQL statement ready for execution (it does not execute the SQL). The getCourses function then calls (invokes) another function named doSQL to handle the querying of the database.

In order for the doSQL function to query the database it needs the calling function (in this case getCourses) to pass the SQL statement to it.

Once the doSQL function has queried the database the database will return data, this is placed in a resource, and this data could be information about several courses.

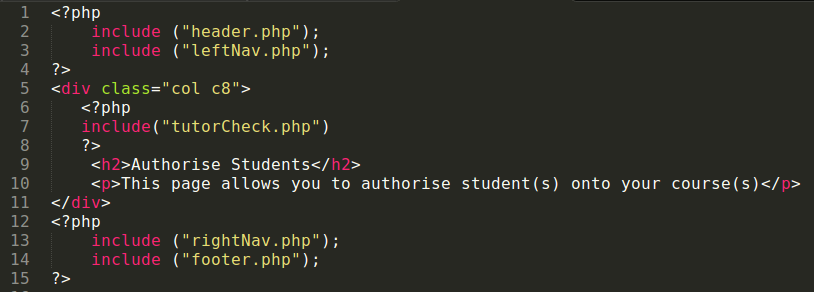
The doSQL function has almost finished its job, but before terminating it passes the data from the database to the invoking function.

We could have simplified this process by constructing and executing the SQL within the getCourses function rather than invoke another function called doSQL, after all this is what we have been doing previously. However, in this case the doSQL code is executed 4 times, and rather than have duplicated the same code 4 times it made more sense to have one instance of the code and simply call it at the appropriate times. If you notice in the diagram we call the function doSQL every time we interact with the database. The database tables are in green.

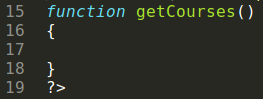
Once we have got the courses from the database the next step is to show them. We create another function to take care of this, again we could have done this within one function but splitting the actions simplifies the code.

Before discussing the flow diagram further, we will code the functions previously mentioned.

1. Open index.php and save as tutorAuthoStud.php
2. This is a tutor page so include the tutorCheck.php file.
3. Adapt the heading and paragraph text to reflect the purpose of the page. You should currently have:



1. Create a function called getCourses, we are not sending any information to this function so leave the parameter parenthesis empty. Start your function on line 15 (so that the function is before the closing PHP tag).



The point of the function is to create the SQL statement to select courses that belong to the tutor that is logged in. Remember once a user logs into the website we create a session variable called userID.

1. Copy the value from the session variable into a local variable called $userID.

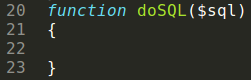


1. Next we create the SQL that will select all the courses that belong to the user that is logged on.



The next line is to call the function to execute the SQL (we will call this doSQL), this function does not exist yet so we will finish the code in the getCourses function once we have create the doSQL function.

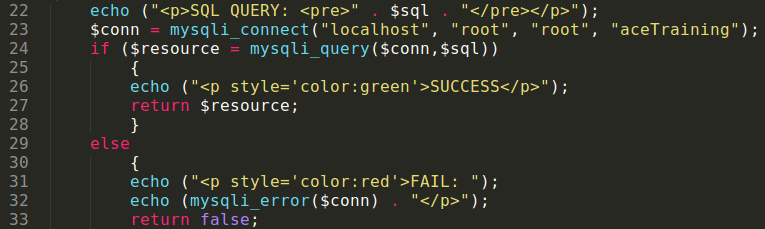
1. Directly underneath the getCourses function (after the final closing curly brace), create a new function called doSQL. The job of this function is to execute the SQL passed to it; therefore this function must accept a parameter. The function should look like:



The doSQL function does not do anything you haven’t seen before with the exception of returning a value to the invoking function (the function that called doSQL). Therefore, we create a connection string, display the SQL, execute the SQL.

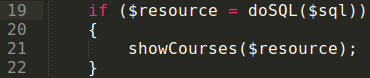
* + If the SQL was successful display, a suitable message and then return the data that was selected from the database to the invoking function.
  + If the SQL was unsuccessfully, display a suitable message with the SQL error and then return false to the invoking function (returning false to the invoking function allows the doSQL function the ability to let the invoking function know that something went wrong).

1. Add the following code to the doSQL function:



Now we need to go back the getCourses function and call the doSQL function. You already know that in order to call a function we use the function name: doSQL(). In this case we send additional information to the doSQL() function, so we provide the additional information between the parenthesis: doSQL($sql). We also know that the doSQL() function will return a value (either the data from the database or false to indicate something went wrong). Therefore we need a variable to accept this return value, so our function call now looks like $resource = doSQL($sql). If the return value was false (i.e. something went wrong) we don’t want to proceed, however, if the return value is the data from the database we will want to call another function called showCourses(). Therefore it make logical sense to use an if statement.

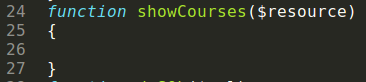
1. Alter your code in the getCourse() function so that it includes the following immediately after the statement that constructs the SQL.



Line 19 reads, call the doSQL function and send this function the SQL that we constructured immediately above. Once this function has finished we expect a return value to be placed inside resource. If the return value is false we do not execute line 21. If however, the return value is not false (i.e. the data from the database) then the overall condition for the if statement evaluates to not false (i.e. true). In the latter case line 21 will be executed.

Next we will create the function called showCourses(), this function is designed to show the data that was passed back from doSQL to getCourses, therefore, the data that was passed back will be passed forward to the showCourses() function by including the $resource between the parenthesis: showCourses($resource). This is depicted in line 21 above.

1. Create a new function called showCourses immediately after the closing curly brace that signifies the end of the getCourses function. The showCourses function should accept a single argument which will be the resource. Your code should look like:



The function is designed to show the courses and allow the tutor to select one. Therefore we need to make use of a form. The tutor could own several courses, so we will need to create a loop that will loop through the resource. We have done something similar to this earlier.

* + Start a form
  + Create a dropdown list with a course (and keep doing this for all courses)
  + Create a submit button
  + Close the form

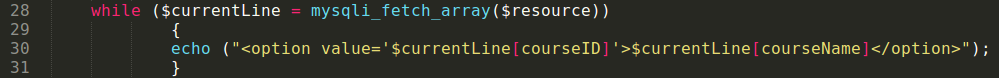
1. Add the code to create a form, the name of the form will be showCourses, the form data will be posted to tutorAuthoStud.php. Your code should look similar to:



Notice that we have also created the select form element. Next we create the dropdown list using option and populate with all the data inside $resource (each row in the resource will be a separate option). We do not know how many courses a particular tutor will own so we need to use a while conditional statement.

1. Create the following code:

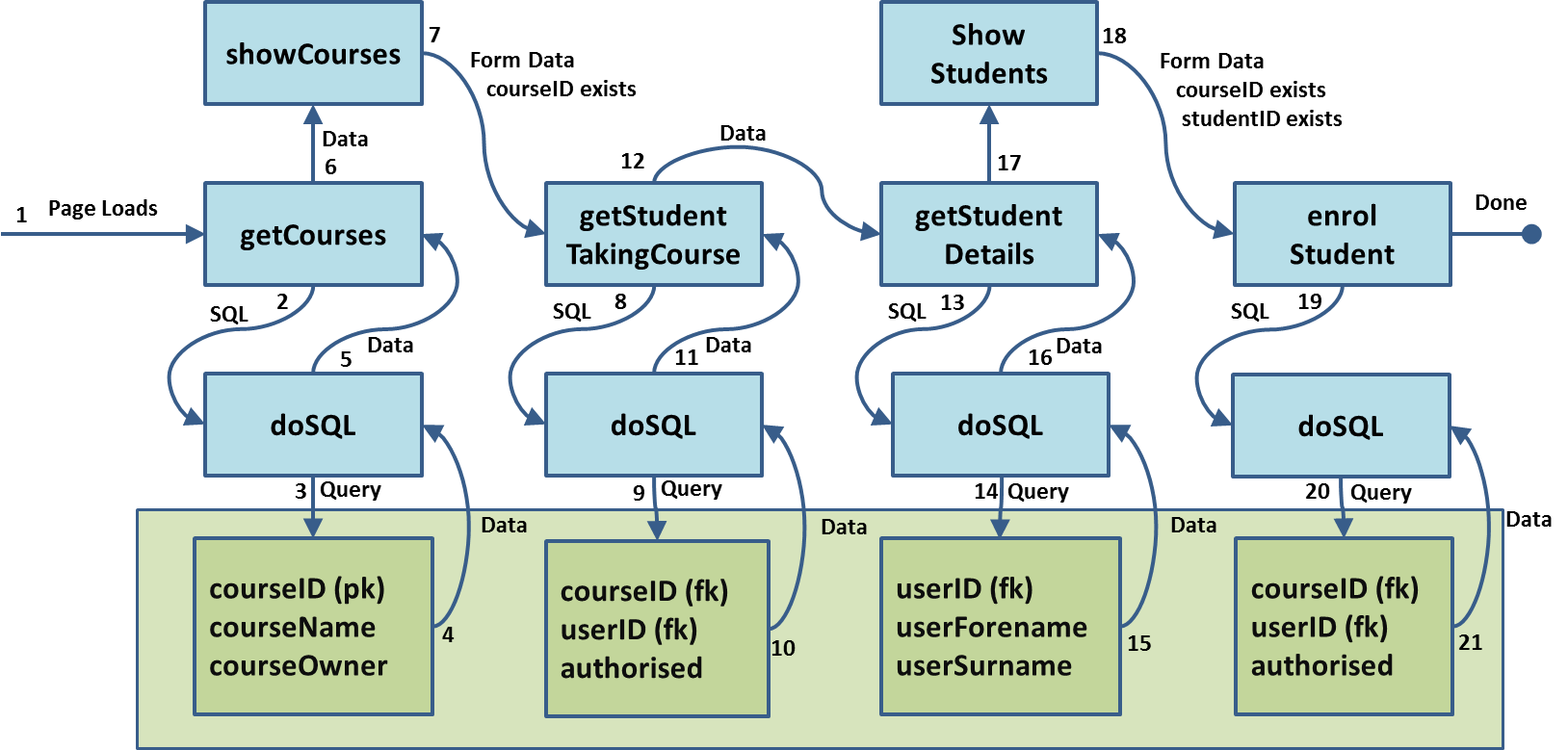
All that remains to do now is to close the SELECT, create a submit button and then close the form.



1. Create the following code:

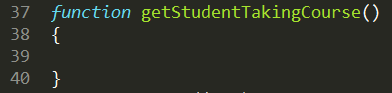


1. Locate the line that prints a paragraph explaining the purpose of this page. Immediately after **<p>This page allows you to authorise student(s) onto your course(s)</p>** add the following: **<?php getCourses(); ?>**
2. We have now done enough to test the first stage. At this point I want you to login as a tutor and create some courses for that tutor. Then click the link Authorise Student from the left navigation menu.



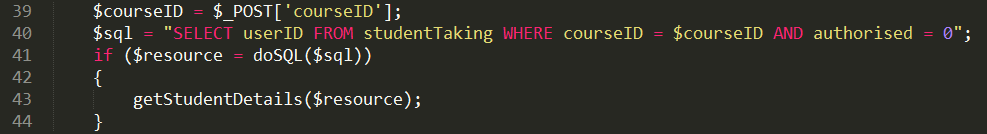
If you refer back to the flow diagram you will notice that the next stage is to getStudentTakingCourse. The objective now is to construct the SQL that will select all students’ id numbers for a particular course (only students currently not authorised). Once the SQL has been constructed the getStudentTakingCourse will call the function doSQL and pass the SQL to this function. Like before the doSQL function will execute the SQL and query the database. If all goes to plan the doSQL function will pass the data from the database back to the calling function (in this case getStudentTakingCourse). The latter function will then pass the data to another function called getStudentDetails that will construct the SQL to get the full student’s details. As before the SQL is passed to the function doSQL so that the user table can be queried. The data from the user table is passed back from the doSQL function to the getStudentDetails which then passes it forward to a function that will display the full student’s details using a form and checkboxes that will allow the tutor to select the required students before submitting. We will continue and create the functions discussed above before moving on to the discussion of the final part of the process.

1. Create the function called getStudentTakingCourse. This function does not accept any arguments (additional data) so leave the parenthesis empty. Your code should look like this:



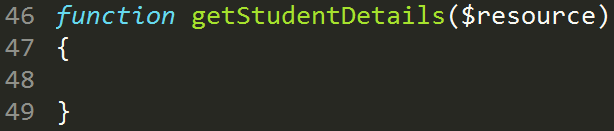
You may have notice I placed this function between the showCourses and the doSQL function. It could have been placed anywhere, however, the way I used allows you to see the natural flow in which the order of the functions are called better.

1. Add the following between the curly braces:



Line 39 takes a posted value from the form showCourses and puts the value into a local variable. Line 40 creates the SQL statement to get all the userID’s from the studentTaking table where the courseID is equal to the courseID that was posted from the form (only when authorised is also set to 0). Line 41 is the same as in the previous function. Call the doSQL function and send it the $sql we constructed in line 40. I expect a return value and I want to put this value inside $resource. If the return value is not false then call the function getStudentDetails and send it the $resource that was returned from the doSQL function.

1. Immediately after the getStudentTakingCourse function (remember the function ends at the closing curly brace, not before) create a function called getStudentDetails. The latter function should accept one parameter, so include $resource between the parenthesis.



Nothing too complicated will happen in this function. It simply uses the userIDs inside the resource to construct the SQL that will be used later to get the full details of the students based on the IDs. We need to tweak the SQL statement slightly. Think what is stored in resource.

|  |  |  |  |
| --- | --- | --- | --- |
| $resourse   |  | | --- | | 2 | | 3 | | 4 | |

The table above presents a resource with three records. We cannot access the IDs until we fetch the records. Just to recall we do this by using the mysqli\_fetch\_array function like so:

**$currentLine = mysqli\_fetch\_array($resource);**

This creates:

|  |  |  |
| --- | --- | --- |
| $currentLine   |  |  | | --- | --- | | userID | 2 | |

Now we can construct the SQL to fetch the user details from the user table based on the id, for example we would do something like this:

**$sql = “SELECT userID, userForename, userSurname FROM user WHERE userID = '$currentLine[userID]' ”;**

The actual statement looks like: **SELECT userID, userForename, userSurname FROM user WHERE userID = '2' ”;**

Then we get the next row: **$currentLine = mysqli\_fetch\_array($resource);**

This creates:

|  |  |  |
| --- | --- | --- |
| $currentLine   |  |  | | --- | --- | | userID | 3 | |

And we ***update*** the SQL statement so that it becomes:

**$sql = “SELECT userID, userForename, userSurname FROM user WHERE userID = '3' OR userID = '$currentLine[userID]'**

The actual statement looks like: **SELECT userID, userForename, userSurname FROM user WHERE userID = '2' OR userID = '3'”;**

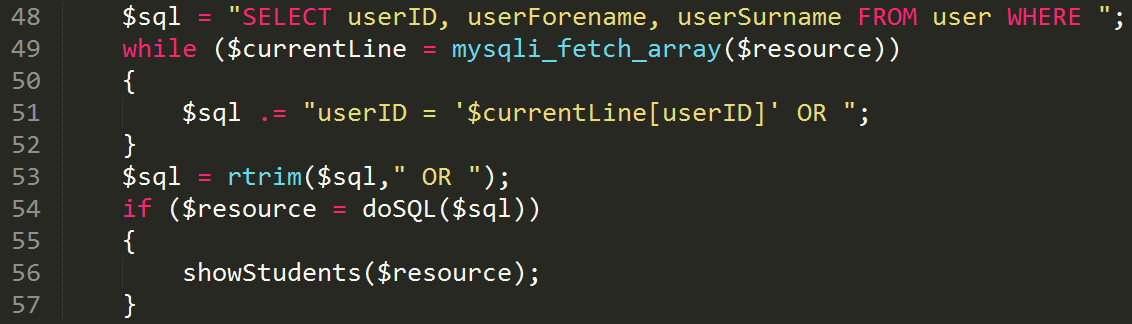
And we keep doing the section: **userID = ‘x’ OR** where x is the number of the userID fetched from the resource. Therefore we need to break the SQL down into stages (similar to what we did with a form when populating a drop down list). This is the structure the code will take within the SQL:

* **$sql = "SELECT userID, userForename, userSurname FROM user WHERE ";**
* **$sql = $sql + "userID = '$currentLine[userID]' OR ";**
* **$sql = $sql + remove the last OR.**

This might seem confusing, but essentially the first statement executes once, then second statement executes for as many userIDs as there are inside resource. The last statement we do just once when there are no more userIDs inside resource to process.

Once we have constructed the SQL we call the doSQL function as previously mentioned and pass the SQL statement to that function. The doSQL function queries the database and passes back a new resource which contains the userID, userForename and userSurname of all the users for the userIDs that where in the original resource.

1. Write the code to accomplish the above now, this is what the code within the function should look like:



The **$sql.=** means it joins (concatenates) whatever is to the right of the equals sign to what is already inside $sql. For example, if **$sql** contained the value Hello, and we did the following: **$sql .=** **“Everybody”;** the new value of **$sql** would be HelloEverybody.

So if $resource did actually contain values 2,3 and 4. Line 49 of the code fetches each value one at a time and builds the SQL statement.

Currently the SQL statement contains: **SELECT userID, userForename, userSurname FROM user WHERE** because of line 48.

We do line 51 for the first value inside resource, the SQL statement now becomes:

**SELECT userID, userForename, userSurname FROM user WHERE userID = 2 OR**

we then fetch the next row (line 49), then we do line 51 again, the SQL statement then becomes:

**SELECT userID, userForename, userSurname FROM user WHERE userID = 2 OR userID = 3 OR**

we then fetch the next row (line 49), then we do line 51 again, the SQL statement then becomes:

**SELECT userID, userForename, userSurname FROM user WHERE userID = 2 OR userID = 3 OR userID = 4 OR**

we do line 49 again. However, this time there is nothing left to fetch, so the if condition evaluates to false and line 51 is skipped.

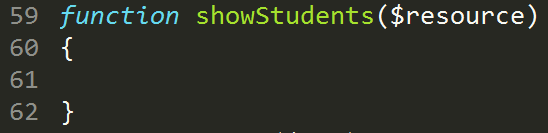
This is what the SQL statement looks like at this stage:

**SELECT userID, userForename, userSurname FROM user WHERE   
userID = 2 OR   
userID = 3 OR   
userID = 4 OR**

This is not a valid SQL statement because of the last OR, so we us a function called rtrim (right trim). The function will trim any character(s) specified from the right-hand side of a string. We want to trim the last or, we do this by **$sql .= rtrim($sql, “ OR ”);** The later essentially reads: make the contents of the $sql variable equal to what is already inside the $sql variable but trim the OR from the right hand side.

We can now proceed and code our next function, this function will show the student details next to a checkbox which will allow the tutor to check (tick) the student(s) that they wish to authorised.

1. Directly after the getStudentDetails function but before the doSQL function, create a new function called showStudents, which will accept one argument. Your code should look similar to the following:



The aim of this function is to create a form that displays all the students inside the resource as well as displaying a check box (tick box) for each student so that he/she can be selected or not selected for authorisation. A submit button will allow the tutor to submit the form, the data posted on submission are the userIDs of the students that were ticked. The only new technique for you is creating a hidden text box.

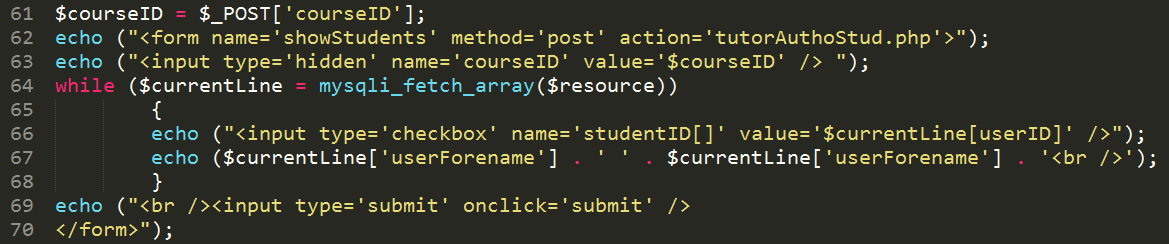
So why do we need a hidden text box? When we submit the form all previous posted values are lost, only the new posted values are retained. Therefore we lose $\_POST[‘courseID’] which was created when the tutor selected a course in the function showCourses().

The purpose of submitting the userIDs of the students is to authorise them, but for which course? This is why we need to retain the value of courseID, otherwise it would have served very little purpose asking the user to select the course in the first place.

Therefore we create a textbox and place the courseID inside the textbox so that this is posted too. We do not really want the user to see the courseID or the textbox (this would mean nothing to the user), and besides which if they did see the courseID in the textbox they might change it!

Therefore we set the input type to hidden rather than text.

1. Create the following code inside the function showStudents()



There is not too much to explain here as this code has been used earlier. You might remember that we use [] brackets to indicate an array. An array is simply a set of values of the same type stored in one variable. In our case studentID[] on line 66 is an array. Once the form is posted this becomes $\_POST[‘studentID’] and will contain the userIDs of all the students that were checked (ticked). If we did not use the [] brackets we would only ever get the userID of the last box that was checked (ticked), thus only one ID.

Next we need to authorise the students using the IDs that are POSTED from the form. We also POSTED the courseID so we know which course to authorised each of the IDs for.

1. Create a new function called enrolStudent after showStudents but before doSQL. This function will not accept any arguments so leave the parenthesis blank.

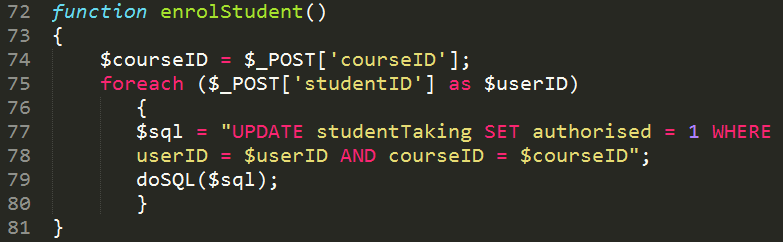
This function is relatively straight forward, we just take the first value stored in the posted studentID, and using this value and the courseID that was also POSTED. We update the authorised value from its current value to 1 signifying that this student is now authorised for this particular course.

We construct the SQL statement which would be similar to:

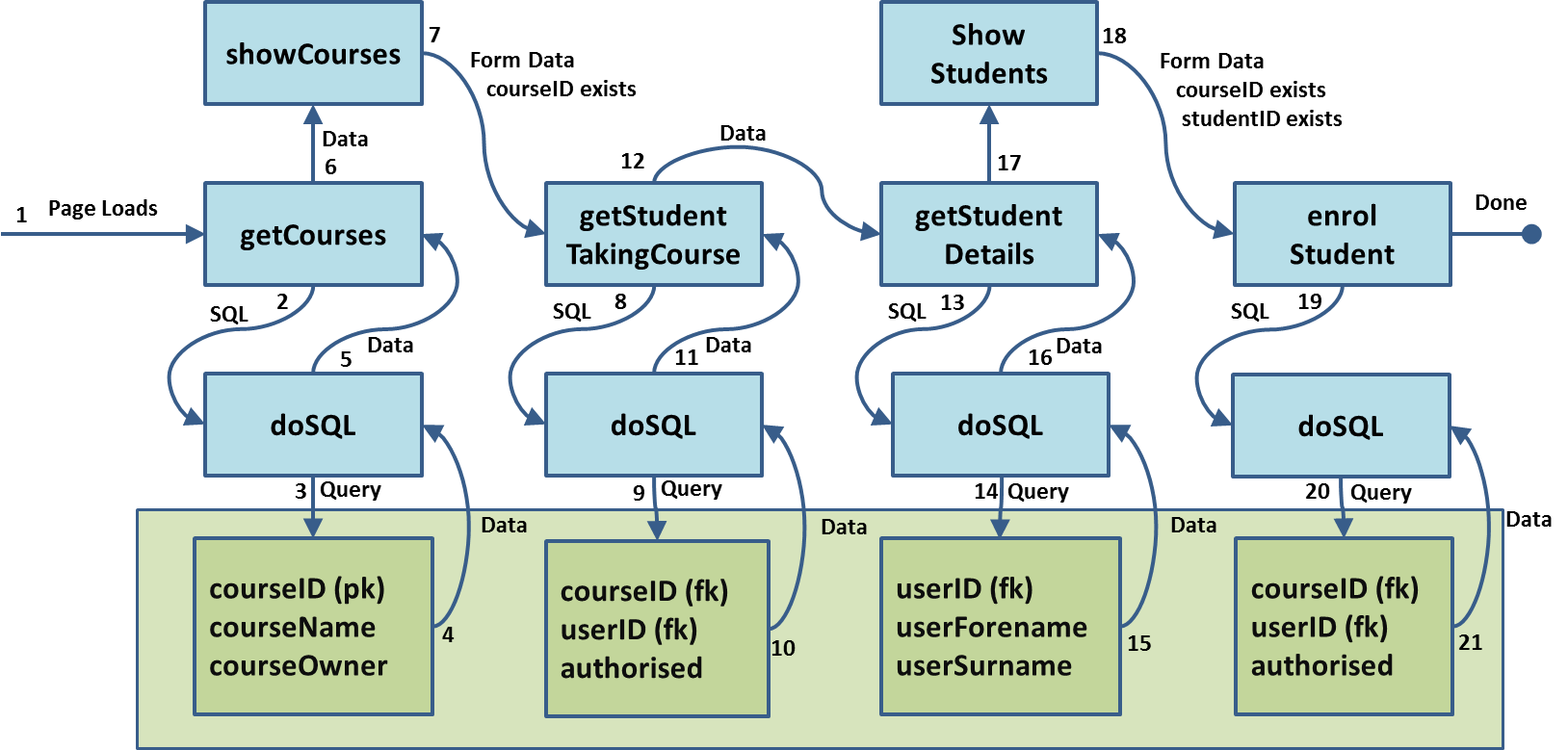
UPDATE studentTaking SET authorised = 1 WHERE userID = $userID AND courseID = $courseID

Then we execute the SQL using the doSQL function. There could be several student IDs so we do the same for each value in the posted studentID.

Your function and could should match:

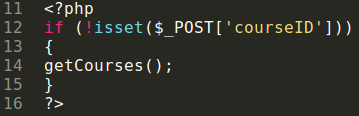


Next we need to add a some flow control so we know which function to call and when.



If we refer back to the flow diagram you can see the first function to call is getCourses(), but we only call this if the user has not had the opportunity to select a course.

1. Immediately after the paragraph that states the purpose of the page add the following script.

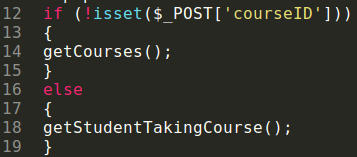


This actually calls three functions. It begins by calling getCourses(), then the getCourses function calls the doSQL function (and passes a resource back to getCourses(). Then getCourses takes this resource and calls showCourses and passes the resource to this function.

Show courses displays a form, which contains a select name called courseID, it is this we are using in our not is set on line 12.

The next function call is getStudentTakingCourse. So extend our conditional statement to include an else.

1. Amend the if to include the else like so:



This function will get the students registered for the selected course (but only the ones not currently authorised), then this function will call the function doSQL and pass it the SQL constructed in getStudentTakingCourse, the doSQL function will execute the SQL and pass a resource back to getStudentTakingCourse, the getStudentTakingCourse function will then call the getStudentDetails and send it the resource.

The getStudentDetails function accepts this resource and uses it to construct a single SQL statement. Once this is complete the getStudentDetails function will call the doSQL function and pass it the newly constructed SQL statement for execution.

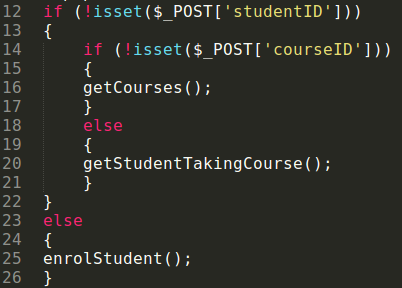
The doSQL function accepts the SQL, queries the database and then passes a resource back to getStudentDetails. The get studentDetails function accepts this resource and calls the showStudents function, the studentDetails function will pass the resource to the showStudents function.

The showStudents function accepts this resource and based on the rows in the resource will echo out the student details for each row. In addition to this a check box is created and assigned the value userID. This value is stored in aa array called studentID[] (the brackets indicate there could be more than one checkbox ticked). Once the user clicks submit, the $\_POST[‘studentID’] is created, and at this point we should call the enrol student function (but only if $\_POST[‘studentID’] exists). Therefore we amend our flow control to reflect this:

1. if (isset($\_POST[‘studentID’]))  
   {  
   enrolStudent();  
   }  
   else  
   {  
   // do the actions previously mentioned.  
   }

Like before we aim to keep the function calls in order, so it might be reasonable to use a not (!).

1. Amend the flow control code to match this:



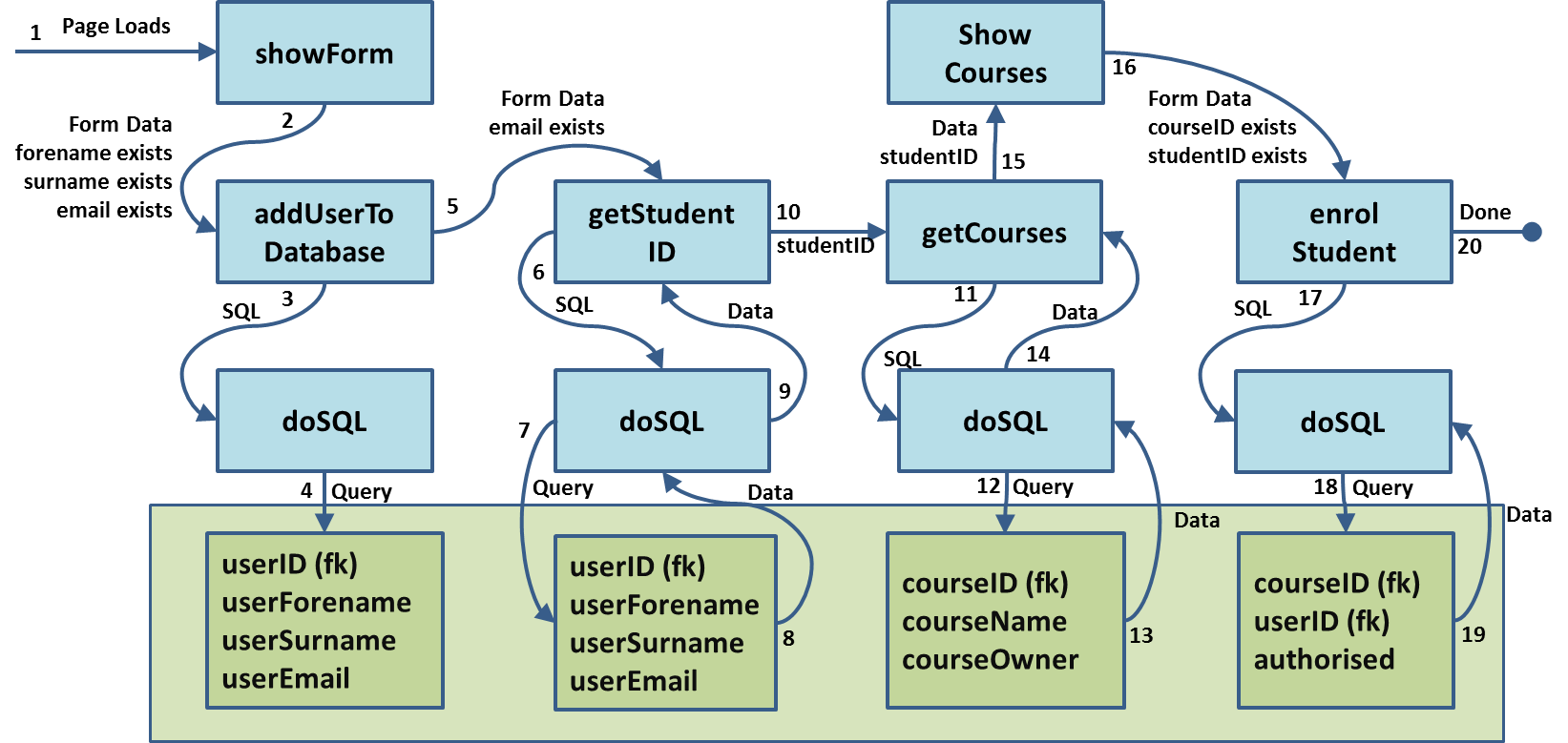
Let’s test what we have done so far.

1. Register a new user as a tutor, use any values you like but for the email use: [tutor1@ace.com](mailto:tutor1@ace.com)
2. Login as this tutor and follow the link to the tutor home page.
3. Using the navigation create a course called: An Introduction to PHP
4. Create another course called: HTML and CSS
5. Register a new user as a student, use any values you like but for the email address use: [student1@ace.com](mailto:student1@ace.com)
6. Register another 2 students with the same email address but substitute the 1 for 2 and 3 respectively.
7. Login as student1 and register for both courses.
8. Login as student2 and register for the HTML and CSS course.
9. Login as student3 and register for the PHP course.
10. Login as tutor1 and see if you can enrol the students onto the courses.

## Enter student registration details manually

This section intends to address how to enter students manually by tutor option. We can assume that because the tutor is entering the student details that the student should automatically be active for a given course. One thing that we do have to account for is which course the tutor is registering the student on.

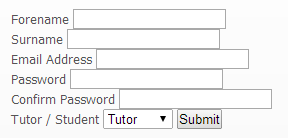
The page will take this form:



* + Show Registration Form
  + Add user to database
  + Get the user ID for the user just added to the database
  + Get List of Course that belong to that tutor
  + Show the courses and all tutor to select
  + Register Student added to database onto the course(s) selected.

We already have code that addresses the first two bullet points. We just need to tweak it a little to alter some of the characteristics, the changes we want to make are:

Remove the password and confirm password fields. When a tutor registers a student a default password will be created based on their email address.



Remove the Tutor / Student dropdown box; this is a tutor registering a student.

1. Open register.php and save as tutorAddStud.php
2. Locate the showForm() function and remove the password, confirm password and tutor / student labels and text boxes.
3. Change the action attribute of the form tag to match tutorAddStud.php
4. Locate the addUserToDatabase() function and remove the lines that copy the password and user type into local variables.

Currently the construction of the SQL statement looks like this:

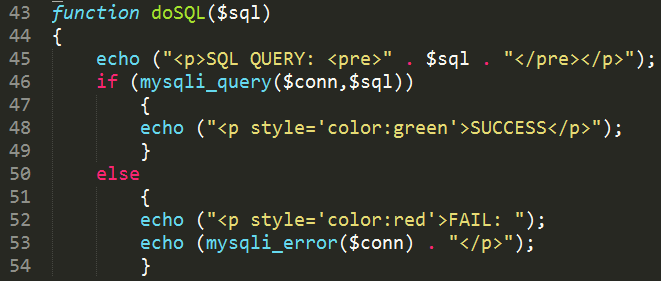


1. We said earlier that we will set the email to a default value based on the email address, therefore replace $pw with $em.
2. The $tp field signified the user type, we have removed this as we know that this user will be a student, replace $tp with student.
3. Finally, because a tutor is adding this student we can assume that they are authorised, change the 0 to a 1. Your SQL should look like:



The rest of this function contains the code to echo the SQL, execute the SQL and then inform the user about the success of the SQL query. We could leave this sitting here quite happily; however, because later on we will need to execute another three instances of SQL we will move this to its own function.

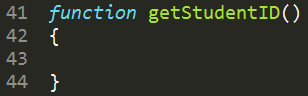
1. Create a function called doSQL($sql)
2. Move **all** the code immediately after the construction of the SQL statement and paste into the doSQL function. Your function should look like this:



1. We also need some return values. After the SUCCESS line, add the line **return $resource;** and after the mysqli\_error($conn) function which is on line 53, add a line below **return false;**
2. We do not have a $resource variable on line 46 because the original query was to insert data into the database. Line 46 above shows: mysqli\_query($conn, $sql) which evaluates the return value. This function could return data from the database. Therefore as well as evaluating we need to store any data returned. Place a variable just before the function so that the line 46 reads: **if ($resource = mysqli\_query($conn,$sql))**
3. Locate the addUserToDatabase() function. Locate the variable $conn that holds the connection parameters to the SQL server. Move this line into the doSQL() function so that it becomes the first line.
4. Locate the addUserToDatabase() function and add the instruction to call the doSQL function immediately after the construction of the SQL statement. You should pass the $sql variable to the doSQL function. The syntax for the function call is: **doSQL($sql);**

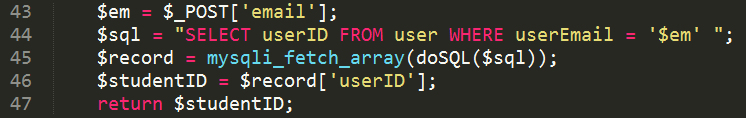
This takes care of the tutor adding a student manually. The next step is to get the automatically generated userID (we will need this when enrolling the student onto courses selected by the tutor).

1. Create a new function called getStudentID, this does not accept any additional parameters so leave the parenthesis empty. Place this function after the addUserToDatabase function and before the doSQL function.



We can get the userID by using the email address, the form values that were posted earlier are still valid (this is because we have not left the page, or resubmitted the form). Therefore we can create a simple SELECT sql statement to get the userID based on the email address.

1. Enter the following code within the function.



Let’s take a closer look at line 45. We read this statement from the inner most nested parenthesis first. Therefore we doSQL first and we pass the SQL defined immediately above. This will return a value (this value is placed in a temporary variable, let’s assume this is called $tmp).

Next we mysqli\_fetch\_array, remember this function will fetch the first row from a resource (in this case our resource is $tmp – although not explicitly declared), this row is placed inside an array called $record. Try not to be confused by this; it simply demonstrates that you can use a returned value (or variable that contains the value) from function without having to put it into a variable that you define, this shorthand version of the code is essentially the same as:

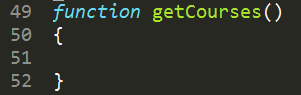
$resource = doSQL($sql);  
$record = mysqli\_fetch\_array($resource);

This function has not been called. According to the flow diagram we call this function after the function addUserToDatabase() has finished.

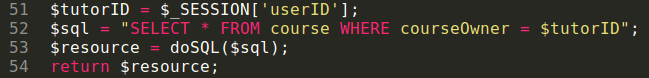
1. Locate the function call: addUserToDatabase(), in my code this is line 13. Immediately after this function call, on the next line, add the following function call:   
    **$studentID = getStudentID();**

The next step is to offer the tutor the ability to enrol this student for one or more of their course. Nothing too complicated here, and in fact we have done similar actions previously. We will do this using two function, the first function to get the courses that belong to the tutor that is logged in. The second function to show the courses that were obtained in getCourses.

1. Create the following function:



1. Add the following code within the function:



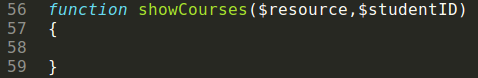
This code will set a local variable called $tutorID to the session variable $\_SESSION[‘userID’] (the session variable is created when a user logs in to the website).

Line 52 construts the SQL statement and will select all courses from the course table. The WHERE clause signifies only to select courses that belong the user that is logged in.

The next line sends the SQL to the doSQL() function to be executed, the doSQL() function will return the data from the database to this function and this return data will be placed inside a local variable called $resource. We then pass $resource back to the point that this function was invoked (you will see later that this is in our flow control section that has not yet been created).

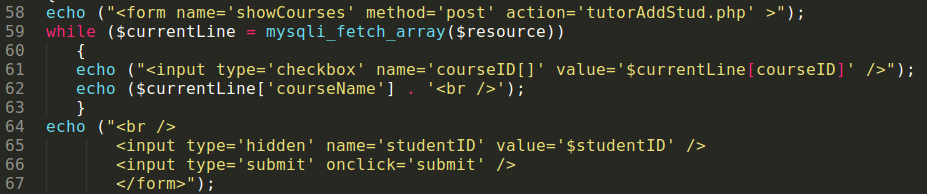
Now we have the courses that belong to the tutor we are ready to create our next function to show those courses.

1. Create the following function:



The purpose of the showCourses() function is to show the courses that are inside $resource. Notice that we are also passing the $studentID. We have to pass the student ID because we are outputting a form, once the form has been submitted, $studentID will no longer exist, therefore we have to create a hidden textbox so that the $studentID is sent with the form.

1. Enter the following code within the showCourses() function:

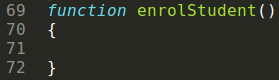


Line 58 echoes the start of the form. Line 59 will fetch a line from the resource and then output a checkbox, notice that we have used courseID[] to signify several checkboxes could be ticked. The value inside the checkbox is set to the courseID (which is inside $currentLine – the current row from $resource). Then we echo out the course name and use this as a label for the check box. We will do this for all the rows inside resource.

When all the rows have been fetched from the resource a break is echoed out (line 64), along with a hidden text box called studentID. We set the value of this hidden text box to the $studentID that we passed, this way we won’t lose this value when the form is submitted. Next we create a submit button and then close the form.

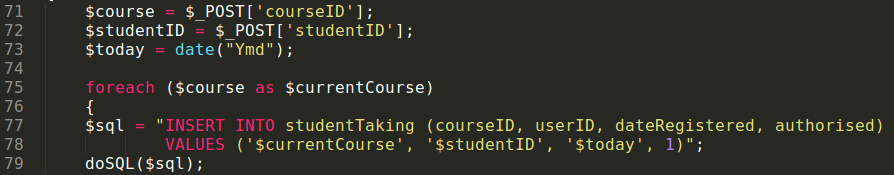
This has allowed the tutor to select the courses that they wish to enrol the student for. The final stage is to take these courses along with the studentID and register the student onto those courses.

1. Create a function called enrolStudent()

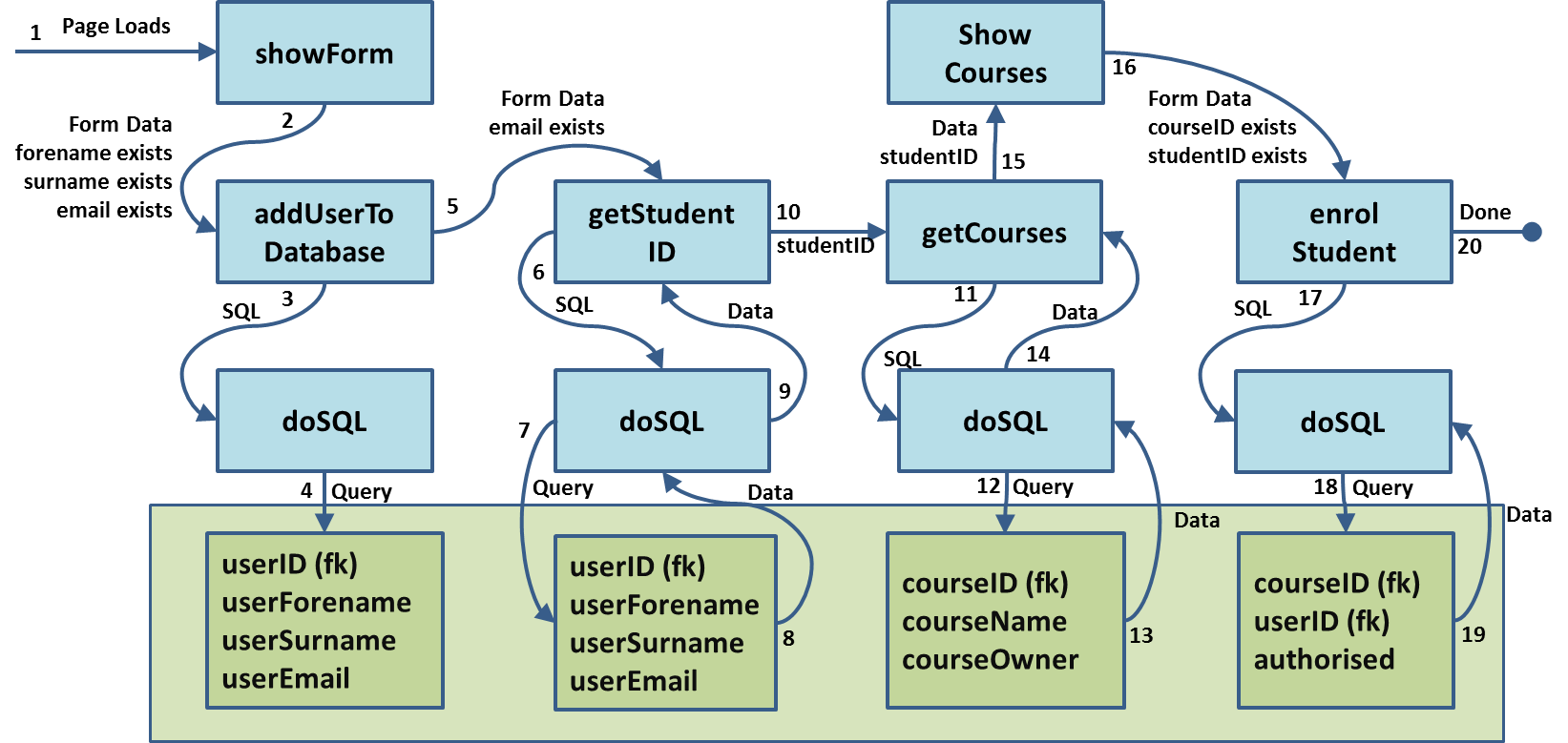


This function will take each course ticked (one at a time) and add this to the studentTaking table, along with the studentID, the date that the student was registered onto that course and automatically set authorised to 1 (indicating yes).

1. Enter the following code inside the enrolStudent function:



We have completed our function for this section, next we need to create the flow control. Which function do we call and when?



The first function we call is showForm(), but only if the form has not already been submitted. Therefore we create a conditional if:

if (!isset($\_POST[‘email’]))  
{  
showForm();  
}

If the $\_POST[‘email’] variable does exist it means that the form has been submitted and we can proceed to enter the data into the database.

else  
{  
addUserToDatabse();  
}

Immediately after adding the user to the database we need to get the userID (remember this is automatically generated by the database so we have no idea what it is). Therefore we alter our else to reflect this:

else  
{  
addUserToDatabse();  
$studentID = getStudentID();  
}

Once the getStudent() function has finished, all the variables in that function are destroyed, therefore we ask the getStudent() function to return the student ID, we place this into a variable called $studentID.

Next we get the courses that belong to the user.

else  
{  
addUserToDatabse();  
$studentID = getStudentID();  
$resource = getCourses();  
}

The next step is to show the courses, remember variables do not exist inside a function unless we create them inside the function, or pass them to the function. Therefore in order for $studentID and $resource (currently holding all the courses that belong to the tutor logged in) to be used inside the showCourses() function we have to send them to the function.

else  
{  
addUserToDatabse();  
$studentID = getStudentID();  
$resource = getCourses();  
showCourses($resource, $studentID)  
}

It may make sense to see why we are sending $resource to the showCourses() function, after all the $resource holds all the courses for the tutor that is logged on.

It may be a little confusing why we are sending $studentID, this is because we are outputting a form. The moment the user clicks submit all data is lost except the form data in showCourses(). Therefore in order to retain $studentID we must also make this part of the form data.

Currently our if else looks like:

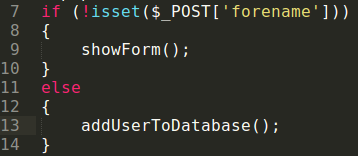
if (!isset($\_POST[‘email’]))  
{  
showForm();  
}  
else  
{  
addUserToDatabse();  
$studentID = getStudentID();  
$resource = getCourses();  
showCourses($resource, $studentID)  
}

The next stage is the tricky part, the tutor will click submit (from inside showCourses()). Therefore nothing will exist except the posted values from this form, these are: courseID[] and studentID, there could be many courseIDs which is signified by the square brackets (an array).

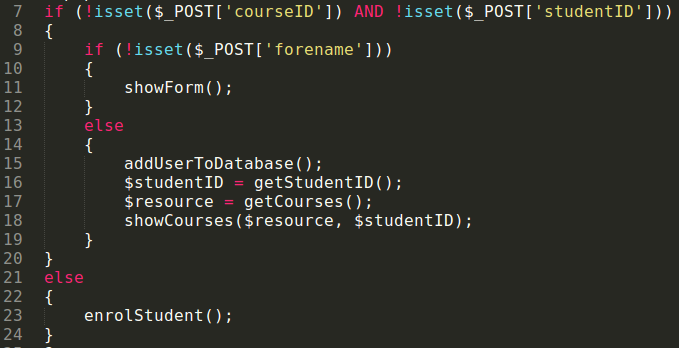
Therefore when the page reloads with the submitted data, $\_POST[‘email’] will not exist, however, we do not want to do the showForm() function all over again… at this stage we want to enrolStudent. The simplest way to do this is to create the following conditional statement.

if (!isset($\_POST[‘courseID’] AND $\_POST[‘studentID’]))  
{  
 // do all the earlier stuff.  
}  
else  
{  
 enrolStudent();  
}

1. Locate the lines that control the flow of execution. In my code it starts at line 7 and finishes at line 14.



1. Replace your flow of execution so that it matches:



We have nearly completed this section, however, there is one last thing to do. The page is called tutorAddStud.php which signifies that this is a page that should only be accessed by a tutor. Therefore we need to include the tutorCheck.php file to ensure only tutors have access the this page.

1. Immediately before the flow control section that you just altered after the <?php tag, add the include statement: include(“tutorCheck.php”);

***End of Portfolio***