Web Development

Lecture 7 - Dynamic Content

Reminders

Main Assignment Details posted to Moodle (top of page)

- Delivery date 15th Dec.

CA Moodle Quiz 14th Nov. Duration 1 Hour. Class continues after quiz.

Watch out for practice quiz in next week’s moodle section.

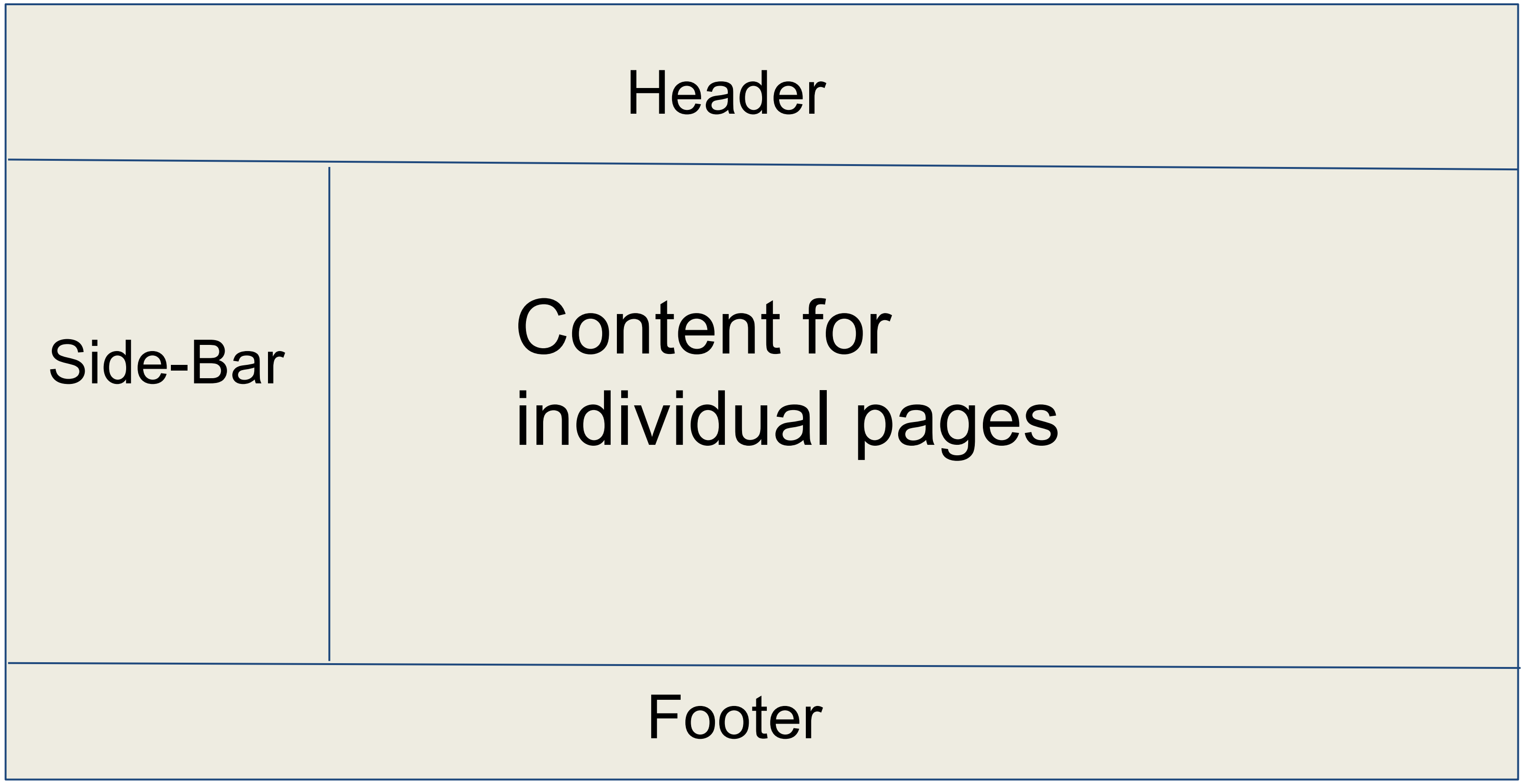
Dynamic vs Static Web Pages

- Consider if you were to generate a full page for every single Amazon product, they would be next to impossible to manually manage
- Pages would get corrupt, lost, mistagged, damaged, etc
- Given that all of them look the same, and the data/content is different, we can look at the notion of a template
- If we use a template, we just fill in the blanks, and alter some portions depending on what we need.

Header/Footer standardisation

- In any set of company pages, we typically have a standard set of header and footers that are the same on all or most pages.
- There can be other elements as well, not just headers and footers
- They can include common side bars, images, elements for different pages
- We will look at the example of headers and footers, and these can be generalised to other forms yourself

Page Layout



Justification for single header/footer

- We are going to create a single file to hold the code for the header and a separate one for the footer
- We then call these on every page that has a header and/or footer
- The idea is that we only have one place to change the code if we decide to change the header or footer, rather than make changes on every page in the site
- This is the same basic idea as creating functions for code reuse rather than rewriting the same code every time
- Particularly useful for menus as they are generally common across an application

How to set this up

- To do this, we need separate files for each bit of code that will be common to all files/pages. Create folder **products** under htdocs
- For now we will create a header, footer, index page and second page
  - header.php
  - footer.php
  - index.php
  - second.php
- We will use the index and second page to show the same content on both pages

header.php

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<link rel="stylesheet" href="menu.css">
<link rel="stylesheet" href="style.css">
<ul>
  <li><a href="index.php">Home</a></li>
  <li><a href="createProduct.php">New</a></li>
  <li><a href="products.php">List</a></li>
  <li><a href="#about">About</a></li>
</ul>
```

- We put whatever should be in every header page in here, in our case,
- Stylesheets
- Menu
- Page title

menu.css

```
ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #333;}
li {
    float: left;}
li a {
    display: inline-block;
    color: white;
    text-align: center;
    padding: 14px 16px;
    text-decoration: none;}
li a:hover {
    background-color: #111;}
```

- Create menu.css in the same folder.
- This will make the menu as a horizontal menu

footer.php

```
<div id='footer' >
  <br/><br/>&copy; CCT
College 2017
</div>
</body>
</html>
```

HTML special characters,
for more see here

- There is a lot more that we could have in our footer, some pages have very little, just contact info, some have a lot more links, as are appropriate for the specific business requirements
- In our case, we just define a piece of text that says

© CCT College 2017

Include function

To add the **header** and **footer** to another page we can use the php

include() function.

We just pass as a parameter the file we want included on the page

This can be just a snippet of html code, like we’ve seen or php code or a mixture of both

index.php

```
<?php
include('header.php');                    // include the header page
?>


    <h1>Welcome To our Products App</h1>
    <p>All sorts of normal content go in here</p>

<?php
    include('footer.php');        // include the footer page
?>
```

secondpage.php

```
<?php
    include('header.php');        // include the header page
?>

    <h1>Second Page</h1>
    <p>All sorts of normal content go in here</p>

<?php
    include('footer.php');        // include the footer page
?>
```

Dynamic Page Content - List of Products

- Now, we want to generate pages based on database entries
- That is, for some online sites that have a large database of products, we don't want to write a separate page for each product, rather, we want to generate the page, or fill in a template based on what comes back out of the database
- Even if the number of products is quite small, we should not hardcode a list of products or hardcode the individual pages

Dynamic Page Content - List of Products

- We're going to create a Products application

- **C**reate - add new product to database (INSERT)
- **R**ead - read product from database (SELECT)
- **U**pdate- edit existing product on database (UPDATE)
- **D**elete- remove product from database (DELETE)

Let’s get set up to try this

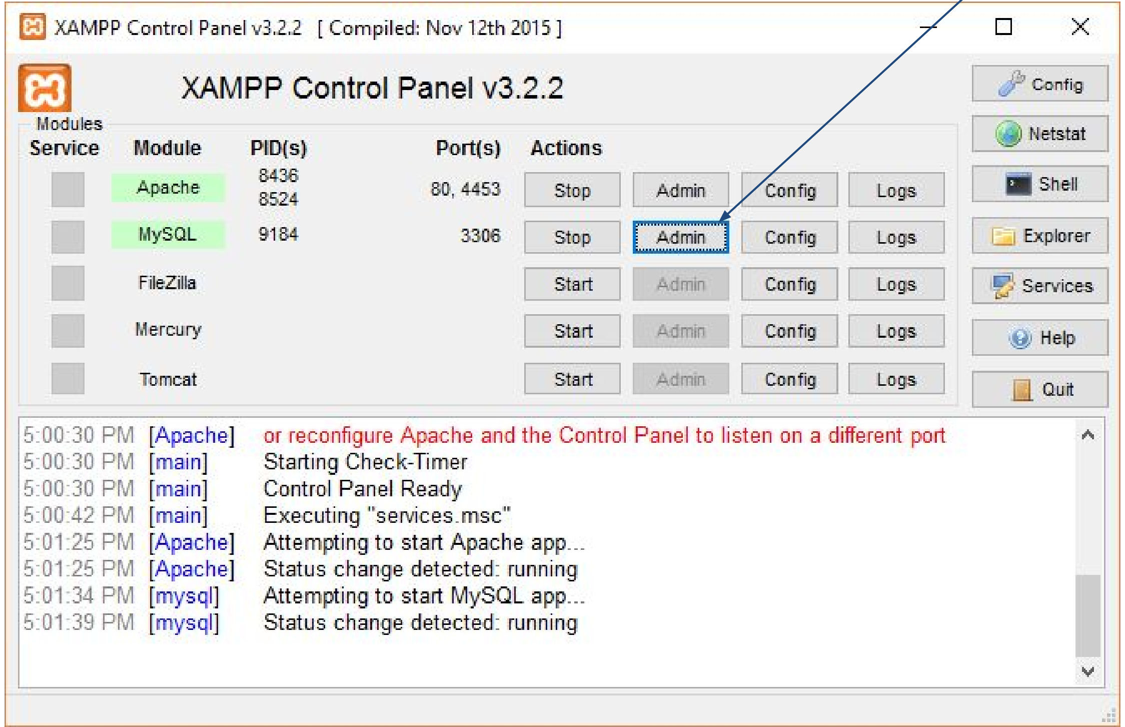
- Start your xampp server and associated MySQL Server
- Open your Heidi or **phpMyAdmin** and create a new table:
- Make sure you have selected a database first (wdtest)

```
CREATE TABLE Products(id INT NOT NULL AUTO_INCREMENT,
                      product_name VARCHAR(50) NOT NULL,
                      product_description VARCHAR(100),
                      cost DECIMAL(8,2),
                      PRIMARY KEY(id));
```

phpMyAdmin

Click Admin button

To start phpMyAdmin on xampp

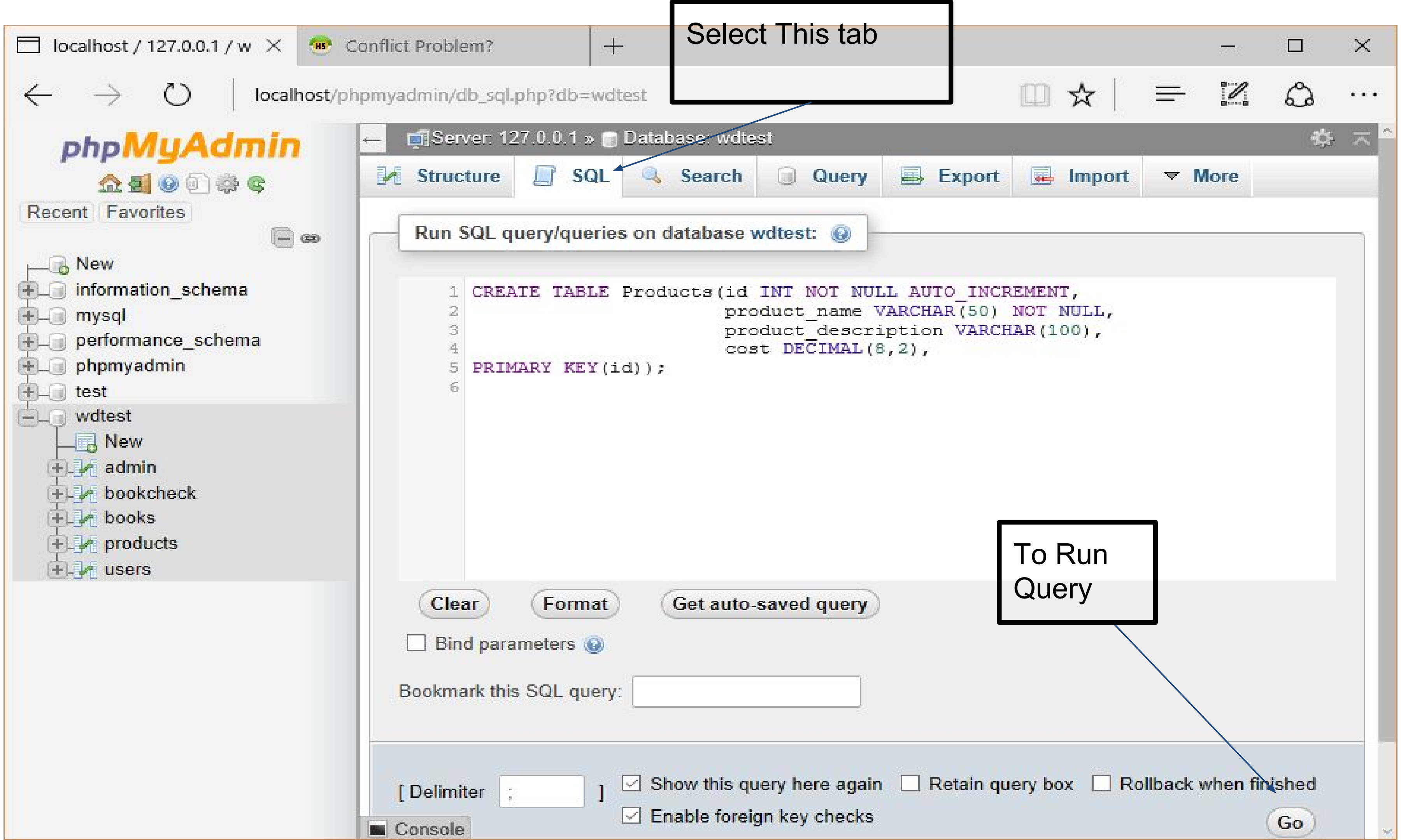

XAMPP Control Panel v3.2.2 [Compiled: Nov 12th 2015]
XAMPP Control Panel v3.2.2
Config
Netstat
Shell
Explorer
Services
Help
Quit
Modules

Service	Module	PID(s)	Port(s)	Actions			
	Apache	8436 8524	80, 4453	Stop	Admin	Config	Logs
	MySQL	9184	3306	Stop	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat			Start	Admin	Config	Logs

```
5:00:30 PM  [Apache]   or reconfigure Apache and the Control Panel to listen on a different port
5:00:30 PM  [main]     Starting Check-Timer
5:00:30 PM  [main]     Control Panel Ready
5:00:42 PM  [main]     Executing "services.msc"
5:01:25 PM  [Apache]   Attempting to start Apache app...
5:01:25 PM  [Apache]   Status change detected: running
5:01:34 PM  [mysql]    Attempting to start MySQL app...
5:01:39 PM  [mysql]    Status change detected: running
```

Select This tab

To Run Query

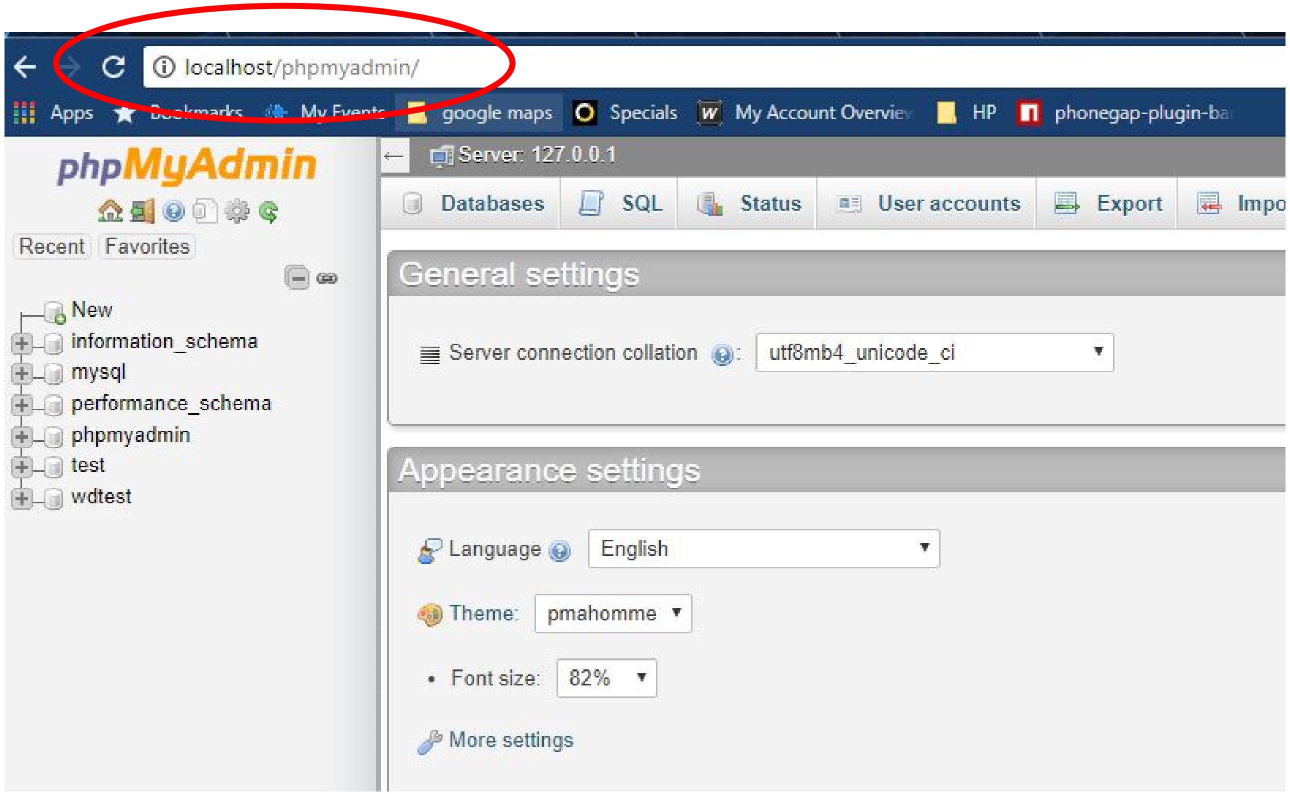


```
CREATE TABLE Products(id INT NOT NULL AUTO_INCREMENT,
                              product_name VARCHAR(50) NOT NULL,
                              product_description VARCHAR(100),
                              cost DECIMAL(8,2),
PRIMARY KEY(id));
```

☑ Show this query here again ☐ Retain query box ☐ Rollback when finished
☑ Enable foreign key checks

phpMyAdmin

Or via browser. localhost/phpmyadmin



Server: 127.0.0.1

- New
- information_schema
- mysql
- performance_schema
- phpmyadmin
- test
- wdtest

General settings

Server connection collation: utf8mb4_unicode_ci

Appearance settings

Language: English

Theme: pmahomme

Font size: 82%

More settings

Add some data

- Now that we have a table, let’s add some data to the table
- Specifically some products
- Add 6 - 7 products to the table, either via SQL statement or ia the INSERT tab
- Eg

1	phone	Some amazing phone An	235.43
2	tablet	even better tablet Snazzy	452.00
3	laptop	laptop	1234.54
4	desktop	Big fancy gaming machine	2345.43
5	monitor	32 inch flat screen	345.32
6	mouse	standard mouse	9.99
7	keyboard	wireless keyboard	20.99

```
insert into  products (product_name,product_description,cost)
 values('pc','Laptop Lenovo',789.00);
```

Double check

- Double check that the data is actually in there:

```
SELECT * FROM Products;
```

Products.php

- First, let’s create a page that will show our **list of products** from the database.
- Essentially,
  - we want to call the database
  - use our select statement to get the information from the database and
  - display this information to the page
- Later we will link these list entries with the specific **product pages**
- If you haven’t done it already. Let’s create a folder under htdocs folder called **products** where we will put our new files

db.php

```
<?php
    try{
        $host ='127.0.0.1';
        $dbname = 'wdtest';
        $user = 'root';
        $pass = '';
        $DBH = new PDO("mysql:host=$host;dbname=$dbname",$user,$pass);
    }catch (PDOException $e) {echo $e->getMessage();}
?>
```

Create this file and save in products folder and we will include it in all our files

db.php

```
<?php
	try{
		$host = '127.0.0.1';
		$dbname = 'wdtest';
		$user = 'root';
		$pass = '';
		$port=3307;
		$DBH = new
		PDO("mysql:host=$host;dbname=$dbname;port=$port",$user,$pass);
	}catch (PDOException $e) {echo $e->getMessage();}
?>
```

If you need to change the port number

PDO revisited

```
$DBH = new PDO(“mysql:host=$host;dbname=$dbname”,$user,$pass);
```

$DBH is an instance of the class PDO

$DBH now has a number of functions (or methods) available to it including the prepare() function

PDO prepare()

Prepares an SQL statement to be executed by the PDOStatement::execute() method.

The SQL statement can contain zero or more named (:name) or question mark (?) parameter placeholders for which real values will be substituted when the statement is executed

```
$stmt = $DBH ->prepare("insert into users
                         (username, email, password)
                          values (?, ?, ?)");
```

Arrow notation allows access to function (or methods) of the instance of the class

Positional Placeholders

PDO Statement Class

The prepare function (or method) returns an instance of the class PDOStatement

In the previous example $stmt is now an instance of the PDOStatement class

PDOStatement has a number of methods we can use, including

bindParam() - bind parameter to placeholder in prepare statement
execute() - execute sql statement

PDO Statement Class

```
$stmt = $DBH->prepare("select * from users where username = ?" );
$stmt->bindParam(1, $username);
$stmt->execute();
```

Once the sql statement has been executed we can use other functions to return data from the sql execution. Generally we are returning rows of data in an **associative array**, where the key is the database table column name and the value is the column data.

```
$stmt->fetch()     -    return single row from a query
$stmt->fetchAll()  -    return all rows in a query
```

Products.php

```
<?php
    // create the connection
    include('db.php');
    // select the correct table
    $stmt = $DBH->prepare("SELECT * FROM Products");
    $stmt->execute();
    // get the rows and put it in a variable
    $rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
    foreach($rows as $row){
        echo $row['id'].", ".$row['product_name'].", ".$row['product_description']."<br/>";
    }
?>
```

errordb.php

We have been having problems with inserts not working and not getting any useful errors. To remedy this we can add the following after the ->execute() of our SQL statement

```
<?php
      $arr = $stmt->errorInfo();
      if (isset($arr[2])) {// we have an error
        echo "<br/> Database Error Code: ".$arr[0];
        echo "<br/> Driver Error Code: ".$arr[1];
        echo "<br/> Database Error Message: ".$arr[2];
        exit();
      }
?>
```

Save as errordb.php and include it in all our php files after where we execute sql query

Products.php

```
<?php
    // create the connection  include(‘db.php’);
    // select the correct table
    $stmt = $DBH->prepare(“SELECT * FROM Products”);
    $stmt->execute();
    include('errordb.php');
    // get the rows and put it in a variable
    $rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
    foreach($rows as $row){
        echo $row[‘id’].”, “.$row[‘product_name’].”, “.$row[‘product_description’]."<br/>";
    }
?>
```

Add this line.
Lets test this,
Change the table name from Products to xyz and reload in browser. Observer the error.

Products List vs Products Page

- We have the list of products, now we want to see a specific product page
- Typically, our database would have more fields to really fill out a product page, but let’s just show the bare bones, and build it up from there
- We will use the information as is in the database to populate a standard product page
- Every product page will look the same, with different data filled into specific areas of the page, again, like a template

Dynamic Products Pages

- In our database we use an **id** to uniquely identify our products
- We are going to use this **id** to call up the information for our individual product to fill in the product page
- To do this we can pass our id from our products list page to the product view page
- Up to now we have passed data from page to page via a form or php sessions etc, this time we’re going to link to the product from the products list via a **href** link passing the id as a parameter

viewProduct.php

- We pass our data from one page to the next, in this case via the URL
- Remember we had two methods of passing data from one form to the next?
    - GET
    - POST
- The POST encoded the information in the HTTP request whereas the GET sends it via the URL
- Using the GET allows us to send data directly via the URL
- We just want to send the id via the URL as a parameter

http://localhost/products/viewProduct?id=1

Passing Parameters via the URL

http://localhost/products/viewProduct?id=1

After the question mark are the parameters

- In this case we are sending the value 1 in the variable id
- If we wanted to send multiple parameters we would separate them with an &

viewProduct.php

Just to make sure it works:

```
<?php
    $pid = $_GET['id'];
    echo $pid;
?>
```

Create a file with this content and save as

viewProduct.php

viewProduct.php

- So we know we can pass the id of the clicked link through to the product page and print it
- Now, we want to use that id to get the specific info out of the database and actually use it
- This time, we are not doing a general SELECT * call to the database, instead, we are doing a SELECT * FROM Products **WHERE** …
- We know we will only get one result back as we are using the Unique Primary Key to get at the data

viewProduct.php

```
<?php
	$pid = $_GET['id'];
	include('db.php');
	$stmt = $DBH->prepare("SELECT * FROM Products WHERE id= :pid");
	$stmt->bindValue(':pid', $pid);
	$stmt->execute();
	include('errordb.php');
	$row = $stmt->fetch(PDO::FETCH_ASSOC);
	echo $row['id'].", ".$row['product_name'].", ".$row['product_description'];
	echo ", ".$row['cost']."<br/>";
?>
```

products.php

```
<?php
// create the connection
include('db.php');
// select the correct table
$stmt = $DBH->prepare("SELECT * FROM Products");
$stmt->execute();
include('errordb.php');
// get the rows and put it in a variable
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
foreach($rows as $row){
	echo $row['id'].",".$row['product_name'].",".$row['product_description'].
	"<a href=viewProduct.php?id=".$row['id'].">View</a>"."<br/>";
}
?>
```

Create a link to viewProduct.php with product id

viewProduct.php

```
<?php include('header.php'); ?>
<h2>%%Put Title Here%%</h2>

<?php
// existing code
?>
<?php include 'footer.php'; ?>
```

Add header and footer to viewProduct.php and products.php.

products.php

```
$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
echo "<table>";
echo "<tr><th>Id</th><th>Name</th><th>Description</th></tr>";
 foreach($rows as $row){
          echo "<tr>";
           echo "<td>";
           echo $row['id'];
           echo "</td>";
           echo "<td>";
           echo $row['product_name'];
           echo "</td>";
           echo "<td>";
           echo $row['product_description'];
           echo "</td>";
           echo "<td>";
           echo "<a href=viewProduct.php?id=".$row['id'].">View</a>";
           echo "</td>";
           echo "</tr>";
}
echo "</table>";
```

Tidy things up with a table. Replace **foreach {}** with red code

Adding Style file to Header

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<link rel="stylesheet" href="menu.css">
<link rel="stylesheet" href="style.css">
<ul>
```

header.php, created earlier

Tidy things up with some css styling. Create **style.css** file in same folder

```
table {
    border: 1px solid black;
}
th, td {
    padding: 15px;
    text-align: left;
}
th{
background-color: mediumseagreen;
color: white }
```

deleteProduct.php

We’re going to add a page to delete a product .

To do this we’re going to populate a form with data from our database table using the **id** of the product selected.

When we have deleted we will return to the products.php file

So we’ll start by creating our form and adding data to it from the table

We’ll make the input fields readonly so the can’t be amended. Using the input attribute **readonly**

deleteProduct.php

```
<h2>Delete Product</h2><br></br>
<form class='form-style' action="deleteProduct.php" method="post">
Product: <input type="text" name="product" value="<?php echo $product; ?>"
readonly/>
Description: <input type="text" name="description" value="<?php echo
$product_desc; ?>" readonly/>
Cost: <input type="text" name="cost" value="<?php echo $cost; ?>" readonly/>
<input type="submit" name="submit" value="Delete" class='button'/>
</form>
```

deleteProduct.php

Add php code above html form

```
<?php
$pid = $_GET['id'];  // from link in products.php
include('db.php');
$stmt = $DBH->prepare("SELECT * FROM Products WHERE id= :pid");
$stmt->bindValue(':pid', $pid);
$stmt->execute();
include('errordb.php');
$row = $stmt->fetch(PDO::FETCH_ASSOC);
$product = $row['product_name'];
$product_desc = $row['product_description'];
$cost = $row['cost'];
?>
```

deleteProduct.php

Add a few lines to **products.php** file to link to the delete page , directly under the viewProduct link

```
echo "<td>";
echo "<a href=deleteProduct.php?id=".$row['id']."">Delete</a>";
echo "</td>";
```

Test the link in the browser

deleteProduct.php

We need to distinguish between a GET and a POST in the file. When we click the link in products.php we are doing a GET.

When we submit the form we are doing a POST

When we see a post we want to delete the row from the table using the DELETE sql method using the data from the form

We need to add a line in the form to store the id of the row we are going to delete. We do not want to display this id on the form.

```
<input type="hidden" name="pid" value="<?php echo $pid; ?>" />
```

deleteProduct.php

Our HTML now looks like

```
<html>
<body>
<h2>Delete Product</h2><br></br>
<form class='form-style' action="deleteProduct.php" method="post">
Product: <input type="text" name="product" value="<?php echo $product; ?>"
readonly/>
Description: <input type="text" name="description" value="<?php echo
$product_desc; ?>" readonly/>
Cost: <input type="text" name="cost" value="<?php echo $cost; ?>" readonly/>
<input type="hidden" name="pid" value="<?php echo $pid; ?>" />
<input type="submit" name="submit" value="Delete" class='button'/>
</form>
</body>
</html>
```

deleteProduct.php

The GET part of php code

```
<?php
include('db.php'); //note we moved this line
if ($_GET){
	$pid = $_GET['id'];
	$stmt = $DBH->prepare("SELECT * FROM Products WHERE id= :pid");
	$stmt->bindValue(':pid', $pid);
	$stmt->execute();
	include('errordb.php');
	$row = $stmt->fetch(PDO::FETCH_ASSOC);
	$product = $row['product_name'];
	$product_desc = $row['product_description'];
	$cost = $row['cost'];
}
?>
```

deleteProduct.php

The POST part of php

```
if ($_POST) {
        $pid = $_POST['pid']; // from hidden input field

        $stmt = $DBH->prepare("DELETE FROM Products WHERE id= :pid");
        $stmt->bindValue(':pid', $pid);
        $stmt->execute();
        include('errordb.php');
        header("Location: products.php");
}
```

Add directly after the if ($_GET) { }

deleteProduct.php

Add the **header** and **footer** php code to deleteProduct.php as before.

Test the app as it stands

Homework

To complete our application we need an UPDATE and CREATE option.

There is an exercise on moodle asking you to do this with the steps you need to complete.

I’ll post a solution file by next tuesday.

Homework

Watch out for additional tutorials in the next weeks moodle section

Practice Quiz

Watch out for practice moodle quiz. Top of moodle page. Hopefully by next tuesday.