PHP / MYSQL – BUILDING A CMS WITH PDO

Before starting the other tasks in this tutorial ensure that you have a connection file above the route of your site. We'll add this file to all the pages that query the MySQL database.

In this lab we're going to build a simple Content Management System (CMS).

Note in these code samples the ↳ indicates the code should be added without a line break.

DRILLING DEEPER USING QUERY STRINGS

Open the file *index.php* and add an include to your connection file.

This file uses a prepared statement to query the 'movies' table based on film name. We now want to extend this example such that the list of results, appear as hyperlinks to a page saved as *details.php*. The *details.php* page will display all the details about a chosen film.

In order to do so we will create a hyperlinks around the search results that have querystrings taking name/value pairs to the *details.php* page. The best value to send to the *details.php* page would be that of the primary key. As such you need to amend the output such that the `filmID` value is set to *details.php*.

Amend the PHP at the top of the file to extract the 'filmID' field.

Amend the code in the PHP while loop that produces the output to add a hyperlink to the details page as follows:

```
echo "<p><a href=\"details.php?filmID={$row->filmID}\">
↳{$row->filmName}</a></p>";
```

Open the *details.php* page. Notice how this uses a `PDO` prepared statement to query the database based on the primary key of the 'movies' table.

There is an error in *details.php*. It is using the wrong super global. Change the `bindParam` to use the value from the querystring.

IMPROVING SECURITY

To fix the issue in *details.php* you should have changed the `$_POST` to a `$_GET`.

This will work, however receiving a variable via a URL can be used by hackers to try and inject Javascript into your database. As such we should add some security.

Open the page *includes/functions.inc.php*. This file includes a number of functions that use PHPs `filter_var()` function to cleanse the data. We could extend what we have here to also filter floats by adding:

```
function safeFloat($float){
     return filter_var($float,
╚FILTER_SANITIZE_NUMBER_FLOAT,FILTER_FLAG_ALLOW_FRACTION);
}
```

Notice there is already a function build for integers called `safeInt()`. As the SQL query in *details.php* expects an integer representing a possible value of 'filmID' the primary key in 'movies' then we should check that the value that comes from the user in the `$_GET` is indeed a integer.

Add *includes/functions.inc.php* as an include to *details.php*.

Amend the query in *details.php* by adding a reference to safeInt(). We could create a 'safe' variable, prefixed with a 's' as follows:

```
$sFilmID = safeInt($_GET['filmID']);
```

Then make sure that this new variable is used in the `bindParam` as opposed to a direct reference to `$_GET`.

In the *index.php* repeat the above steps to secure the `$_POST['filmName']`

- Add *includes/functions.inc.php* as an include.
- Create a ‘safe’ variable to use in the `bindParam`.

DISPLAYING IMAGES DYNAMICALLY

In order to display the image in the *details.php* page add a line of PHP that makes use of the 'filmImage' field in the database. You'll find a PHP comment suggesting where to place the image.

Add the following:

```
echo "<p><img src=\"images/$row->filmImage\"></p>";
```

Extend the code above to automatically add an ALT attribute that has the value of the film name.

FORMATTING THE DATE

The date is currently output in the standard YYYY-MM-DD MySQL format. Add the following to create a variable called `$timestampDate` which is formatted for output.

```
$timestampDate = strtotime($row->releaseDate);
$displayDate = date("D d M Y", $timestampDate);
```

DYNAMICALLY LOADED IMAGES

Our database also has a review field `filmReview` that is an integer value between 0 and 5. There is a sub-directory of images, one for each of the possible star ratings.

We can use this variable to add images dynamically:

```
echo "<p><img src=\"images/stars/
star-rating-{$row->filmReview}.png\"</p>";
```

When testing your page in the browser try changing the numeric value of filmID which is exposed in the URL. You should be able to retrieve different records by just changing this value.

BUILDING A SIMPLE CONTENT MANAGEMENT SYSTEM

Notice there is a login form on both *index.php* and *details.php* that links to *cms/cms.php*.

This where we'll build our CMS. For now this is not protected, but you could use techniques such as PHP sessions to restrict access to this page.

- Open the file *cms/cms.php*. Add your connection include file. This will be a different path to that used in `index.html` because the CMS files are in an additional folder.

This file lists all the films in the 'movies' database.

- Amend the SQL so that they are listed alphabetically.
- Add the following to use Bootstraps table styling:

```
<table class="table table-striped">
```

This page will form the basis of a simple Content Management System.

In the table amend the links to the *details.php*, the *edit.php* and the *delete.php* PHP pages to pass the primary value of `filmID` to them via a querystring. Your amended code should look as follows:

```
echo "<td><a href=\"edit.php?filmID=
╚{$row->filmID}\">Edit</a></td>";
echo "<td><a href=\"delete.php?filmID=
╚{$row->filmID}\">Delete</a></td>";
echo "<td><a href=\"../details.php?filmID=
╚{$row->filmID}\">View</a></td>";
```

Don't forget in these code samples the `╚` indicates the code should be added without a line break.

UPDATING A RECORD

Open the PHP page *edit.php*.

- Add your include files to both your connections and functions files.

Notice that this contains an SQL query based on the primary key value of 'filmID'. The results from this query are then used to populate the form fields with the current values for that film. This form will be submitted to the file *process/editRecord.php* to edit the record.

However, to edit the film we also need to send the primary key in order to make sure the correct record is edited. In the *edit.php* add a hidden field to hold the primary key. This can remain hidden as we don't want users to edit it but we do need it to process the query.

```
<input name="filmID" type="hidden"
↳value="<?php echo $row->filmID; ?>">
```

The HTML form's action attribute points the form at a file called *process/editRecord.php*.

Open the file *process/editRecord.php*.

- Add includes to both your connection and function files.

Before building an UPDATE query create safe variables using the functions in the functions include.

```
$sFilmName = safeString($_POST['filmName']);
$sFilmDescription = safeString($_POST['filmDescription']);
$sFilmImage = safeString($_POST['filmImage']);
$sFilmPrice = safeFloat($_POST['filmPrice']);
$sFilmReview = safeInt($_POST['filmReview']);
$sFilmID = safeInt($_POST['filmID']);
```

Now add the code to UPDATE the movies table.

```
// prepare SQL
$sql = "UPDATE movies SET filmName = :filmName,
                          filmDescription = :filmDescription,
                          filmImage = :filmImage,
                          filmPrice = :filmPrice,
                          filmReview = :filmReview
                          WHERE filmID = :filmID";
$stmt = $pdo->prepare($sql);
$stmt->bindParam(':filmName', $sFilmName, PDO::PARAM_STR);
$stmt->bindParam(':filmDescription', $sFilmDescription,
PDO::PARAM_STR);
$stmt->bindParam(':filmImage', $sFilmImage, PDO::PARAM_STR);
// use PARAM_STR although a number
$stmt->bindParam(':filmPrice', $sFilmPrice, PDO::PARAM_STR);
$stmt->bindParam(':filmReview', $sFilmReview, PDO::PARAM_STR);
$stmt->bindParam(':filmID', $sFilmID, PDO::PARAM_INT);
$stmt->execute();
// redirect browser
header("Location: ../cms.php");
// make sure no other code executed
exit;
```

This page is not designed to be viewed but is a 'process' file that will edit the record and then 'bounce' the user back to the *cms.php* page through the use of a the PHP header function.

INSERTING A RECORD

Open the file *insert.php* file. This contains a HTML form with all the required fields for the 'movies' table. Notice that the form does not include a field for 'filmID'. This is because the 'filmID' field in the 'movies' table is set up as auto-increment. Therefore a unique primary key value will be allocated to any new entry automatically.

We can use some PHP in here to avoid repetitive HTML. Locate where users will add a star rating for the film and the following loop to dynamically build 5 radio buttons.

```
<?php
// loop 5 star checkboxes
$noStars = 5;
for($i=1;$i<=$noStars;$i++){
?>
<label class="radio-inline">
<input type="radio" name="filmReview" value="<?php echo
$i;?>" >
<?php echo $i;?>
</label>
<?php
}
?>
```

Notice that the action of the form is to post the values to a file called *process/insertRecord.php*. The *insertRecord.php* file will be in the same vein as the *editRecord.php* example above. That is it is a file that will process the necessary SQL but will not produce any HTML to be displayed and will return the user to the cms.php page.

Open the file *process/insertRecord.php*.

- Add includes to both your connection and function files.

Before building an INSERT query create safe variables using the functions in the functions include.

```
$sFilmName = safeString($_POST['filmName']);
$sFilmDescription = safeString($_POST['filmDescription']);
$sFilmImage = safeString($_POST['filmImage']);
$sFilmPrice = safeFloat($_POST['filmPrice']);
$sFilmReview = safeInt($_POST['filmReview']);
$sFilmID = safeInt($_POST['filmID']);
```

Now build the prepare statement. You will find it useful to review *editRecord.php* and the way the prepare statement is created. You will also need to know the syntax of a SQL INSERT statement.

```
INSERT INTO tablename(field1, field2, field3)
VALUES (value1, value2, value3)
```

Remember you don't need to insert a value for 'filmID' the primary key.

DELETING A RECORD

Review the file *delete.php*. This file will be linked to from the *cms.php* file. The link you added previously will pass the primary key of the chosen film to the *delete.php* file.

- Add includes to both your connection and function files.

The *delete.php* file needs to query the database to retrieve the chosen film's name and display it to the user so they can decide whether to delete the record.

Inside *delete.php* file is a HTML. The form is designed to send the film's `filmID` primary key to the *process/deleteRecord.php* file. This form uses the more secure POST method.

The secondly there is a hyperlink that will just return the user to the *cms.php* file without further action. To make the hyperlink appear like a button use Bootstrap styling by amending the link to:

```
<a href="cms.php" class="btn btn-success">Save</a>
```

Edit the first form to add a hidden field for the 'filmID" value to ensure the current 'filmID' is sent to the *deleteRecord.php* file.

The code required is as follows:

```
<input name="filmID" type="hidden"
↳value="<?php echo $row->filmID; ?>">
```

Open the file *process/deleteRecord.php*.

- Add includes to both your connection and function files.
- Make sure that you sanitize the `filmID` with the `safeInt()` function.
- Build a prepare statement to delete the film
- Redirect the user.

Your code may appear something like:

```
ini_set('display_errors', 1);

require('YOURPATH-TO-Conn.inc.php');

require(''YOURPATH-TO-functions.inc.php');

// sanitize user variables

$sFilmID = safeInt($_POST['filmID']);

// prepare SQL

$sql = "DELETE FROM movies WHERE filmID = :filmID";

$stmt = $pdo->prepare($sql);

$stmt->bindParam(':filmID', $sFilmID, PDO::PARAM_INT);

$stmt->execute();

// redirect browser

header("Location: ../cms.php");

// make sure no other code executed

exit;
```

Save and test.