

## PHP PART TWO

### WORKING WITH MYSQL VIA PHPMYADMIN

In the previous lab you will have installed XAMPP.

Along with PHP, XAMPP installed MySQL. MySQL is the database server and PhpMyAdmin is a web-based tool for administering a MySQL database. With XAMPP running to view PhpMyAdmin visit:

[localhost / phpMyAdmin 5.1.1](http://localhost/phpMyAdmin/5.1.1)

### SETTING UP THE DATABASE WITH PHPMYADMIN

From here we can create a new database called *db1*:



Give the Database a name of *db1* and use the default collation. The collation indicates the character encoded to be used.

Now we can create a table in the database. Create a new table named Films with eight fields.

Structure SQL Search Query Export Import

⚠ No tables found in database.

Create new table

Table name:  Number of columns:

You will now be prompted to add the field names, data types and settings such as whether a field is a primary key. The fields you will adding are:

Field name	Field Type	Notes
<b>filmID</b>	Integer 4 characters maximum.	The Primary Key. Set as A/I for auto increment.
<b>filmTitle</b>	varchar 70 characters maximum.	Long enough for the largest film name.
<b>filmCertificate</b>	varchar 5 characters maximum.	To allow 'u', 'pg' as well as numeric values '12', '18' etc.
<b>filmDescription</b>	Text	Allows for larger amounts of text.
<b>filmImage</b>	Varchar 50 characters maximum	The name of an image file to create an image path if needed. Images themselves stored separately.
<b>filmPrice</b>	Decimal(5,2)	Five digits, two after the decimal place.
<b>stars</b>	Int 11 characters	Will hold value of 0-5 as a 'star' rating for the film.
<b>releaseDate</b>	Date	Added in MySQL standard format of YYYY-MM-DD.

The above settings will appear in PhpMyAdmin as follows:

Table name:  Add  column(s)

Column Name	Type	Length	Null	Default	Extra	Primary
filmID	INT	4	None			<input checked="" type="checkbox"/> PRIMARY
filmTitle	VARCHAR	70	None			<input type="checkbox"/>
filmCertificate	VARCHAR	5	None			<input type="checkbox"/>
filmDescription	TEXT		None			<input type="checkbox"/>
fimImage	VARCHAR	50	None			<input type="checkbox"/>
filmPrice	DECIMAL	5,2	None			<input type="checkbox"/>
stars	INT	2	None			<input type="checkbox"/>
releaseDate	DATE		None			<input type="checkbox"/>

The settings can always be edited or deleted but ensure you get the correct names and data types set as this stage to avoid issues with later examples. Your table should now appear as:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/> 1	filmID	int(4)			No	None		AUTO_INCREMENT	<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
<input type="checkbox"/> 2	filmTitle	varchar(70)	utf8mb4_general_ci		No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
<input type="checkbox"/> 3	filmCertificate	varchar(5)	utf8mb4_general_ci		No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
<input type="checkbox"/> 4	filmDescription	text	utf8mb4_general_ci		No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
<input type="checkbox"/> 5	fimImage	varchar(50)	utf8mb4_general_ci		No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
<input type="checkbox"/> 6	filmPrice	decimal(5,2)			No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
<input type="checkbox"/> 7	stars	int(2)			No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>
<input type="checkbox"/> 8	releaseDate	date			No	None			<a href="#">Change</a> <a href="#">Drop</a> <a href="#">More</a>

## IMPORTING THE DATA

The data for the table originated in an Excel file. The file had the column titles removed and was saved as a CSV, a comma delimited file, that can be found in the data folder of the repo. The raw data appears as:

```
1,Cinema Paradiso,15,"A famous film director returns home
2,Hear My Song,15,"Nightclub owner Mickey O'Neill (Adrian
3,The Shawshank Redemption,15,"Andy Dufresne is a young ar
4,Harry Potter and the Philosopher's Stone,PG,"Sometimes t
5,Fargo,18,"Jerry works in his father-in-law's car dealers
6,Jurassic World Dominion,PG,"Dinosaur blockbuster picks u
7,The Big Lebowski,15,"When ""The Dude"" Lebowski is mista
8,"Lock, Stock and Two Smoking Barrels",18,"Four Jack-the-
9,The Matrix,15,"Thomas A. Anderson is a man living two li
10,0 Brother Where Art Thou,12,"Loosely based on Homer's
```

This CSV file can then be used to populate the *Films* table. Import the file data/Films.csv file via the 'import' tab.

The screenshot shows the phpMyAdmin interface for importing data into the 'films' table. The top navigation bar includes 'Browse', 'Structure', 'SQL', 'Search', 'Insert', 'Export', and 'Import'. The 'Import' tab is active, displaying the 'File to import:' section. This section includes instructions on file formats (gzip, bzip2, zip, or uncompressed) and a file selection area where 'films-data-my-csv.csv' is chosen. Below this, the 'Character set of the file:' is set to 'utf-8'. The 'Partial import:' section has a checked option 'Allow the interruption of an import in case the script detects it is close to the PHP timeo' and a 'Skip this number of queries (for SQL) starting from the first one:' field set to '0'. The 'Other options' section has a checked option 'Enable foreign key checks'. The 'Format' section at the bottom shows 'CSV' selected.

Server: 127.0.0.1 » Database: db1 » Table: films

Browse Structure SQL Search Insert Export Import

**File to import:**

File may be compressed (gzip, bzip2, zip) or uncompressed.  
A compressed file's name must end in `.[format].[compression]`. Example: `.sql.zip`

Browse your computer: (Max: 40MiB)

Choose File `films-data-my-csv.csv`

You may also drag and drop a file on any page.

Character set of the file:

`utf-8`

**Partial import:**

☒ Allow the interruption of an import in case the script detects it is close to the PHP timeo  
This might be a good way to import large files, however it can break transactions.

Skip this number of queries (for SQL) starting from the first one:

`0`

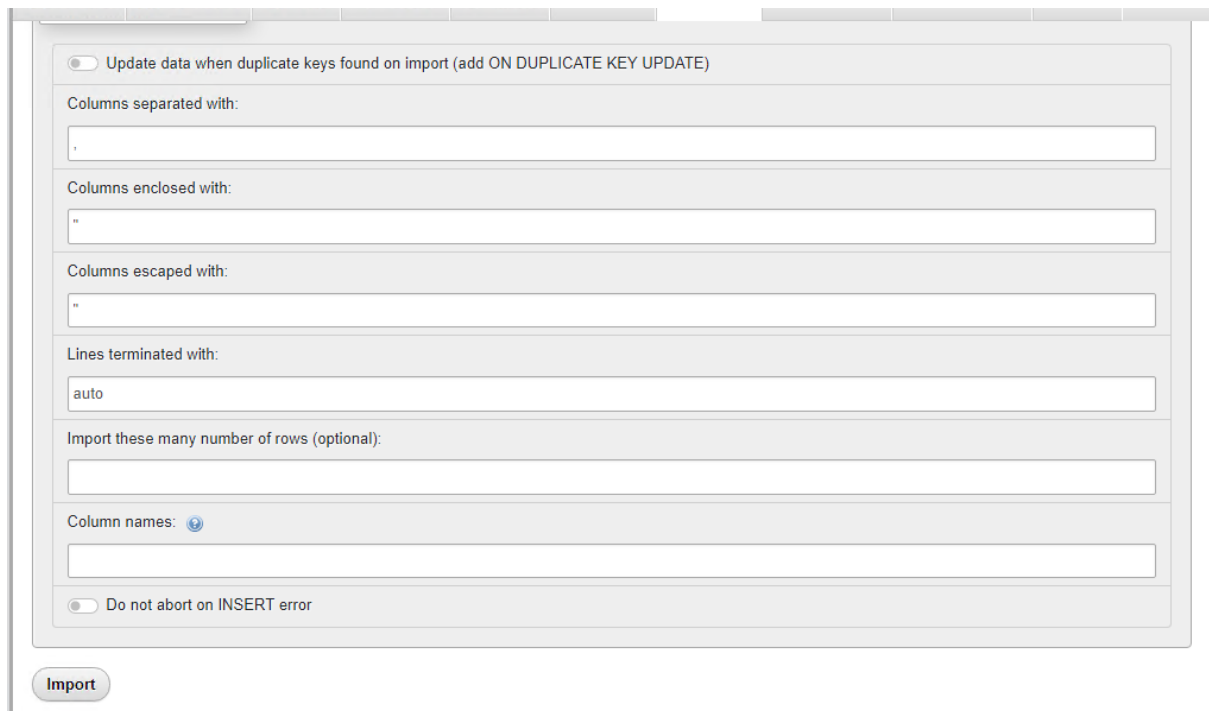
**Other options**

☒ Enable foreign key checks

**Format**

`CSV`

Ensure import setting match that of a CSV that is comma delimited and uses quotes to enclose columns. The settings should be:

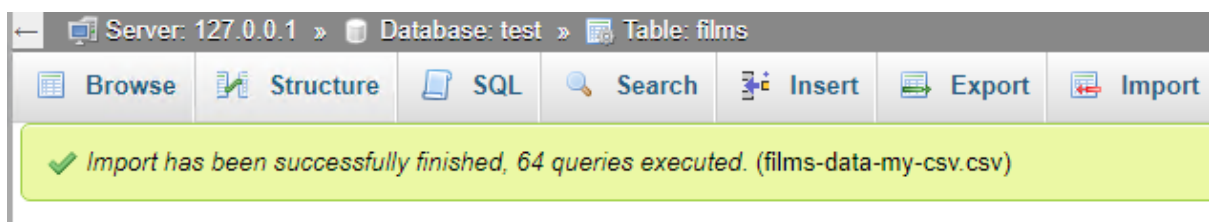


The screenshot shows the 'Import Data' dialog box in MySQL Workbench. The settings are as follows:

- ☐ Update data when duplicate keys found on import (add ON DUPLICATE KEY UPDATE)
- Columns separated with: ,
- Columns enclosed with: "
- Columns escaped with: \"
- Lines terminated with: auto
- Import these many number of rows (optional):
- Column names:
- ☐ Do not abort on INSERT error

At the bottom left, there is an 'Import' button.

If successful you should see the following:



The above indicates that 64 records were imported.

Note: once you have the data in MySQL you could use the export tab to create an SQL file backup of your data, the reserve of the procedure we have just done.

Take time to look at the data via the Browse option on the tab in PhpMyAdmin. Knowing your data is extremely useful when building an application.

	filmID	filmTitle	filmCertificate	filmDescription	filmImage	filmPrice	stars	releaseDate
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	1	Cinema Paradiso	15	A famous film director returns home to a Sicilian ...	cinemaParadiso.jpg	0.99	5	1988-02-23
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	2	Hear My Song	15	Nightclub owner Mickey O'Neill (Adrian Dunbar), a ...	hearMySong.jpg	1.50	5	1991-03-13
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	3	The Shawshank Redemption	15	Andy Dufresne is a young and successful banker who...	shawshankRedemption.jpg	1.00	2	1994-02-17
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	4	Harry Potter and the Philosopher's Stone	PG	Sometimes the best plan is to do things by the boo...	philosopherStone.jpg	1.00	5	2001-11-16
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	5	Fargo	18	Jerry works in his father-in-law's car dealership ...	fargo.jpg	1.99	5	1996-05-31
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	6	Jurassic World Dominion	PG	Dinosaur blockbuster picks up four years after the...	jurassicWorldDominion.jpg	1.00	3	2022-06-10
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	7	The Big Lebowski	15	When "The Dude" Lebowski is mistaken for a million...	bigLebowski.jpg	1.00	5	1998-05-01

## CONNECTING TO MYSQL

Open the PHP file in the *includes* folder called *config.php*. Set the constant values of DB\_SERVER, DB\_USERNAME, DB\_PASSWORD and DB\_NAME to match your application. (Note root and a blank password are the defaults in XAMPP).

```
<?php
/* Database credentials */
define("DB_SERVER", "127.0.0.1");
define("DB_USERNAME", "root");
define("DB_PASSWORD", "");
define("DB_NAME", "db1");
/* Attempt to connect to MySQL database */
$mysqli = new mysqli(DB_SERVER, DB_USERNAME, DB_PASSWORD,
DB_NAME);
// Check connection
if($mysqli === false){
    die("ERROR: Could not connect. " . $mysqli-
>connect_error);
}
?>
```

This file creates a `$mysqli` variable that contains a mysqli object. It is this object that we will use to query the database. As such any file that wishes to use the database connection should 'include' the *config.php* file.

## CREATING A SIMPLE QUERY - SINGLE RECORD

Open the file *index.php*. We will add a query to extract one value from the database.

When extracting data to use in a web page the requirement will always be:

- Include the connection *config.php* file.
- Build a SQL query.
- Use the `mysqli` extension to send your query to MySQL.
- Create an object to hold the results of your query.

First you need to include the connection file *config.php* file. Place this code at the very top of the file before the `<!doctype html>` declaration.

```
<?php
require_once('includes/config.php');
?>
```

Add a string variable that includes an SQL query:

```
<?php
require_once('includes/config.php');
$queryFilms = "SELECT filmTitle, filmImage FROM Films WHERE filmID = 10";
?>
```

The SQL query above will extract the *filmTitle* and *filmImage* fields from the *Films* table where the *filmID* is equal to '10'.

We next use the `$mysqli` object created in the *config.php* file and call the `query()` method sending it the SQL as a parameter. The result of this is a `mysqli_result` object. This represents the result set obtained from a query against the database and is stored in `$resultFilms`.

```
<?php
require_once('includes/config.php');
$queryFilms = "SELECT filmTitle, filmImage FROM Films WHERE filmID = 10";
$resultFilms = $mysqli->query($queryFilms);
?>
```

In PHP object properties and methods are called with `->` syntax (equivalent to a `.` in Javascript).

We can now extract the data from the result set as an object using the `fetch_object()` method. It is from this object that we can finally retrieve the individual values.

Your code should now appear:

```
<?php
require_once('includes/config.php');
$queryFilms = "SELECT filmTitle, filmImage FROM Films WHERE filmID = 10";
$resultFilms = $mysqli->query($queryFilms);
$obj = $resultFilms->fetch_object();
?>
```

To display a value, in the body of the HTML file now use `echo` to output the value of *filmTitle* within the `<section>` element.

```
<!-- Featured Films Here -->
<?php
    echo "<h2>{$obj->filmTitle}</h2>";
?>
```

Notice that `{...}` are placed around the dynamic value represented by `$obj->filmTitle`. When `echo` a value retrieved from an object or array place them in `{...}`.

Amend the above to output the *filmDescription* field in a `<p>` element underneath the *filmTitle*. To do this you will need to amend the SQL to extract the *filmDescription* and then you can output it follows:

```
<!-- Featured Films Here -->
<?php
    echo "<h2>{$obj->filmTitle}</h2>";
    echo "<p>{$obj->filmDescription}</p>";
?>
```



Instead of one film, we'd like the four most current films listed. We can do this by changing the SQL to:

```
$queryFilms = "SELECT * FROM Films ORDER BY releaseDate DESC LIMIT 0,4";
```

... and remove the object creation so the full code at the top is now:

```
<?php
require_once('includes/config.php');
$queryFilms = "SELECT * FROM Films ORDER BY releaseDate DESC LIMIT 0,4";
$resultFilms = $mysqli->query($queryFilms);
?>
```

In the body of the document change the output to:

```
<!-- Featured Films Here -->
<?php
    while ($obj = $resultFilms -> fetch_object()) {
        echo "<div>";
        echo "<h3>{$obj->filmTitle}</h3>";
        echo "<p>{$obj->filmDescription}</p>";
        echo "</div>";
    }
?>
```

Notice that in the above the `$obj` is created inside of a while loop. This is because we expect to receive more than one value from the result set held in `$resultFilms`.

As such the while loops around, extracting all the films that match the query. In this example, the loop will run four times and output different HTML for each film.

## ADDING IMAGES DYNAMICALLY

It would be good to show the image for each of the four featured films. To do so we'd need write some HTML as follows:

```

```

To do this with dynamic values we need to add the *filmImage* value to the `src` attribute. As this is an attribute, and as we `echo` the HTML we need to escape the quotes with backslashes as follows:

```
echo "<img src=\"images/{\$obj->filmImage}\"";
```

This is because the double quote would match that of the `echo` unless escaped and course an error. Escaping is a technique in programming that treats a character as a 'normal' character as opposed to applying its special meaning.

We could also add the `alt` attribute dynamically with:

```
echo "<img src=\"images/{\$obj->filmImage}\" alt=\"{\$obj->filmTitle}\">";
```

To complete the example, place the image above the *filmTitle* as follows:

```
<!-- Featured Films Here -->
<?php
    while ($obj = $resultFilms -> fetch_object()) {
        echo "<div>";
        echo "<div>";
        echo "<img src=\"images/{\$obj->filmImage}\" alt=\"{\$obj-
>filmTitle}\">";
        echo "</div>";
        echo "<h3>{\$obj->filmTitle}</h3>";
        echo "</div>";
    }
?>
```

## LISTING ALL THE RECORDS

Open the file *catalogue.php*. This will feature a list of all the films. Add the following to the top of the page:

```
<?php
require_once("includes/config.php");
// query to get all films
$queryFilms = "SELECT * FROM Films";
$resultFilms = $mysqli->query( $queryFilms );
?>
```

This loads the database configuration include and then runs a `SELECT *` SQL query to extract all the records.

Add an output to the body of the file of:

```
<!-- Film Listing Here -->
<?php
while ($obj = $resultFilms -> fetch_object()) {
    echo "<p>{$obj->filmTitle}</p>";
}
?>
```

This loop creates a `<p>` for each film.

You could extend this with more data and change the HTML written into a HTML table.

```
<div class="listing">
  <table>
    <tr>
      <th>Film</th>
      <th>Certificate</th>
      <th>Price</th>
    </tr>
    <?php
      while ($obj = $resultFilms -> fetch_object()) {
        echo "<tr>";
        echo "<td>{$obj->filmTitle}</td>";
        echo "<td>{$obj->filmCertificate}</td>";
        echo "<td>&pound; {$obj->filmPrice}</td>";
        echo "</tr>";
      }
    <?>
  </table>
</div>
```

## THE DETAILS PAGE

Review the page, *film-details.php*. This page will display the information related to one film in more detail. It will do so by receiving the film to be queried in the query string, the URL, in the form of:

```
http://localhost/webdev-php-mysqli-project/film-  
details.php?filmID=n
```

The query string value will then be used in a SQL query based on the *filmID*, the primary key in the database, such as:

```
SELECT * FROM Films WHERE filmID = n
```

As the above query is constructed from values received from the user, this needs a subtly different approach. This is because we need to ensure that the value from the user, in the query string, is not malicious and an attempt to hack our data via a technique known as SQL injection. To prevent this, we construct the query as a prepare statement via a prepare / bind procedure. This look as follows:

```
$getFilmID = $_GET["filmID"];  
$stmt = $mysqli->prepare( "SELECT * FROM Films WHERE filmID  
= ?" );  
$stmt->bind_param( 'i', $getFilmID );  
$stmt->execute();
```

In this example the SQL query is ‘prepared’ with the question mark (?) indicating the value that can change.

```
$stmt = $mysqli->prepare( "SELECT * FROM Films WHERE filmID  
= ?" );
```

We bind this parameter through a second command that we can use to ensure that it is integer, via the **i** flag above.

```
$stmt->bind_param( 'i', $getFilmID );
```

In the above the value retrieved in the query string is used ie `$_GET["filmID"]` that has been saved into `$getFilmID`.

**Binding flags:** In the above **i** is used as the bind type but you can also use **s** for string.

The execute command then runs the query against the database.

```
$stmt->execute();
```

To retrieve the values requires two more lines:

```
$result = $stmt->get_result();
```

This gets a result set from the prepared statement. Then as in the previous example we can extract the object that represents the results with:

```
$obj = $result->fetch_object();
```

The `$obj` is then used to output values later in the file.

## ADDING QUERY STRING LINKS

In the *catalogue.php* page edit the code that displays the *filmTitle* to make each a link to the *film-details.php*. Each link should also pass the *filmID* value as a query string. The output of the code should build links that in the case of the first film would look like:

```
<a href="film-details.php?filmID=1">Cinema Paradiso</a>
```

We are generating this dynamically however, so the amended line of code should appear as:

```
echo "<td><a href=\"film-details.php?filmID={$obj->filmID}\">{$obj->filmTitle}</a></td>"
```

Note again the use of backslashes to ensure the double-quotes around the `href` attribute value do not break the `echo` statement.

Similarly, return to the *index.php* page and add links to the *film-details.php* for all the films listed.

```
<section class="homePage">
  <!-- Featured Films Here -->
  <?php
    while ($obj = $resultFilms -> fetch_object()) {
      echo "<div>";
      echo "<a href=\"film-details.php?filmID={$obj->filmID}\">";
      echo "<div>";
      echo "<img src=\"images/{$obj->filmImage}\" alt=\"{$obj->filmTitle}\">";
      echo "</div>";
      echo "<h3>{$obj->filmTitle}</h3>";
      echo "</a>";
      echo "</div>";
    }
  ?>
</section>
```

In the above `<a>` is added around the image and `<h3>` passing the *filmID* as a query string value to the *film-details.php* page.

## TIDYING UP THE DETAILS PAGE

We can improve the details page by making more use of the data. Add the film certificate underneath the film description. To do this in the main output block add:

```
echo "<p>{$obj->filmCertificate}</p>";
```

Remember page titles should be unique. We can get the database to do this for us by dynamically generating a `<title>` from the values in the database. Add the *filmTitle* to the `<title>` element with:

```
<title><?php echo $obj->filmTitle; ?></title>
```

Given that the *film-details.php* page relies on a query string value to run the query we need to improve the error handling of the page. For example, what if no value is included as a query string ie:

```
film-details.php
```

... or the *filmID* given is out of range, given the values in the database:

```
film-details.php?filmID=9999
```

We need to handle these possibilities.

First, we can check for the existence of the *filmID* in the query string. This can be done with:

```
if(!isset($_GET["filmID"])){  
    $getFilmID = null;  
}else{  
    $getFilmID = $_GET["filmID"];  
}
```

This uses the PHP method `isset()` to check if there is a *filmID* value in the query string. The `!` indicates an 'is not set' /negative version of the command.

If no query string value for *filmID* is found we set the `$getFilmID` to null.

Next, after the SQL query has run we can check if any results were returned by accessing the `num_rows` property of the result set. If no records are returned, we can use a redirect which in PHP would appear as:

```
if($result->num_rows <= 0){  
    header("Location: catalogue.php");  
}
```

The redirect works by setting the HTTP header to the location value given. In this case we redirect our user to the catalogue list page.



## REFACTORING

The if/else statement above can be refactored using a ternary operator. This is a one-line if/else statement that takes the form of:

```
Question ? Correct answer response : Incorrect answer response.
```

So the if/else could be replaced with:

```
$getFilmID = isset($_GET["filmID"]) ? $_GET["filmID"] : null;
```

As the use of `isset()` is a common use-case, PHP allows even further refactoring by using a feature known as a Null Coalescing Operator. This applied with two question marks and the above example would be refactored to:

```
$getFilmID = $_GET[ 'filmID' ] ?? null;
```

## BUILDING A SEARCH

So far, we have listed the films in the database for users. It would be better to allow users to search the database themselves. In order to do this, we will need a HTML form, where the user can add they search term, and then we need to build an SQL query to run the against the database. As a user input is needed we will need to use the prepare statement technique outlined above but this time using a string rather than an integer.

In the *search.php* file there is a HTML form with an input type `text` with a `name` of `q`. As the form has no method or action, it will default to a method of `get` and an action that will reload the current page. In the *search.php* page therefore:

```
<form>
```

... could be written as:

```
<form method="get" action="search.php">
```

As the method is `get` the value of `q` will be submitted as a query string.

Test this by adding the following in the body of the HTML where prompted with `<!-- Search Results Here -->`.

```
<?php
echo $_GET["q"];
?>
```

At the top of the file before `<!doctype HTML>` declaration add the following to check whether a search value has been submitted:

```
//check for search value,
$searchQuery = $_GET[ 'q' ] ?? null;
```

If a value was submitted then `$searchQuery` will contain the value, if not it is equal to null. The above user the Null Coalescing Operator seen above and could have been written as:

```
if(isset($_GET[ 'q' ])){
$searchQuery = $_GET[ 'q' ];
}else{
$searchQuery = null;
}
```

Next, we build a prepare statement. The SQL we would like to run will be something like:

```
SELECT * FROM Films WHERE filmTitle LIKE '%captain%';
```

The above assumes the user submitted 'captain' as their query. Note that this uses a SQL LIKE query that allow the use of the wildcard `%`. We could have used an equal but that would only return exact matches and would be less user friendly.

```
SELECT * FROM Films WHERE filmTitle = 'Captain Marvel';
```

In the equals example the user would have to type in 'Captain Marvel' in the query to get a match, whereas with the LIKE example the query will find any films with a 'captain' in the title.

To facility the LIKE search we will concatenate the `$searchQuery` value with a `%` at each side. Concatenate simply means adding to the string. In PHP we concatenate with the dot (`.`) character.

```
$searchQuery = "%" . $searchQuery . "%";
```

We can now prepare an SQL statement with:

```
$stmt = $mysqli->prepare( "SELECT * FROM Films WHERE filmTitle  
LIKE ?" );
```

Remember the questionmark (`?`) is the value to be bound to the statement. Do this with:

```
$stmt->bind_param( 's', $searchQuery );
```

This time we bind the parameter with a flag of `s` for string.

Now we can execute the query:

```
$stmt->execute();
```

... and then get the result set with:

```
$result = $stmt->get_result();
```

It will be useful to create a variable to check if the search is valid that can be used to control the output aspect of the page as such the final logic for the top of the page will appear as:

```
<?php,
require_once( "includes/config.php" );
//check for search value
$searchQuery = $_GET[ 'q' ] ?? null;
if ( is_null( $searchQuery ) || empty( $searchQuery ) ) {
    $validSearch = false;
} else {
    $validSearch = true;
    $searchQuery = "%" . $searchQuery . "%";
    // query by filmTitle
    $stmt = $mysqli->prepare( "SELECT * FROM Films WHERE filmTitle
    LIKE ?" );
    $stmt->bind_param( 's', $searchQuery );
    $stmt->execute();
    $result = $stmt->get_result();
}
?>
```

To output the values of the search in the body of the document add:

```
<?php,
if ( $validSearch ) {
echo "<p>Your search found {$result->num_rows} result(s)";
while ( $obj = $result->fetch_object() ) {
echo "<h3>{$obj->filmTitle}</h3>";
echo "<p><a href=\"film-details.php?filmID={$obj->filmID}\">More Details</a></p>";
}
} else {
echo "<p>Search for a film.</p>";
}
?>
```

This uses techniques we have seen before such as the `while` loop (as we may get more than one result) and this time the output is wrapped in the if/else to only output a result if a search has taken place.

## TAKING THINGS FURTHER

Extending the above project could you create an include for the sidebar. The site owner would like the side bar to randomly feature a film from the database. To help you the SQL for this would be:

```
SELECT * FROM Films ORDER BY RAND() LIMIT 1
```

You may also try placing the values submitted by the user in the contact form to a database table. You would need a new table for this that matches the fields in the contact form. Then you would need to run a SQL INSERT statement that took the user inputs and via a prepare statement added them to the new table. This would involve binding multiple values. This cheat sheet might help:

<http://www.mustbebuilt.co.uk/2011/11/17/mysqli-cheatsheet/>