Saving Data and Displaying Pages

Overview

Instead of managing website content using a complex tool like phpMyAdmin, we can create a simple content management system that provides minimal functionality to create new database records. We'll begin with a page containing a form (from week 10) that sends data to a PHP script that can insert new values into the database we set up in the previous activity.

Before starting this activity, download **Lab 11 Files.zip** and extract the contents to the **htdocs** folder of your XAMPP webserver.

- **htdocs** (web server documents)
  - **week11** (sub-folder)
    - index.php (page used to view the website)
    - add.php (page containing form)
    - list.php (page to display all database records)
    - save.php (page to save new database records)
    - **css** (sub-folder)
    - **images** (sub-folder)

Make sure your XAMPP webserver is running with both Apache and MySQL started.

Note: This activity introduces basic PHP database interaction and content management concepts. The code in this activity does not implement security to prevent authorised access or exceptions for data handling. **Do not use this code in production without implementing proper safeguards.**

Connecting to the database

1. Open **save.php** in your code editor. At the very top of the file, before the **doctype**, add some PHP tags. When the server encounters a PHP start tag, it starts processing PHP code. When it reaches a PHP closing tag, it resumes outputting unprocessed HTML.

```
<?php
?>
<!DOCTYPE html>
```

   Between the PHP tags, create a **$db** (database connection) variable and connect to the database:

```
<?php
$db = new mysqli( 'localhost', 'root', 'root', 'fit1050' );
?>
```

   PHP variables declarations require no special keyword or specific datatypes. However, all variable names must begin with a **$** character.

   We are using PHP's **mysqli** (MySQL improved) extension to connect to the database with the following parameters:

- **Host** localhost (can also be a server's IP address)
- **Username** root
- **Password** root
- **Database** fit1050
- ***Port*** *Optional – only required if MySQL is not using port 3306*

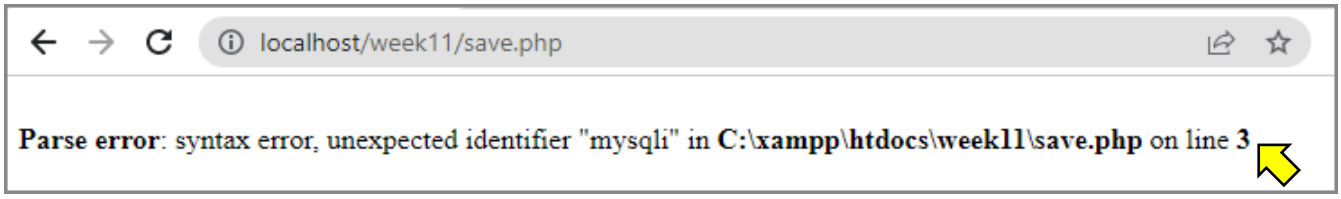
If the connection is successful, an object representing the connection is stored in the **$db** variable. The **$db** database object also allows us to access various methods for working with the database.

2. When the database connection is no longer required, it should be closed.

   Add some PHP tags at the end of the document and close the connection:

```
</html>
<?php
$db->close();
?>
```

3. Open **http://localhost/week11/save.php** in your browser. The code currently produces no output, so if you see an error message, check the line number indicated to identify where the error is:



Note: Depending on your code and server configuration, your output this week might have no errors, or a different error message. The example warning in the screenshot above simply informs us that the server is temporarily assuming a timezone setting of "UTC".

If you see this specific warning, you can manually set the server timezone at the top of your script: https://www.php.net/manual/en/function.date-default-timezone-set.php

Checking for data from a form submission

1. Locate the **main** element in the document. This is where we will do the processing of data and display output to indicate where data was received.

   Add PHP tags containing an **if** statement:

```
<?php
if () {

}
?>
```

   In the condition of this **if** statement, we'll check if the page is being viewed without receiving POST data corresponding to form's fields found in **add.php**:

```php
if ( !isset( $_POST['title'], $_POST['path'], $_POST['content'] ) ) {

}
```

3 variables are being checked within the request's **$_POST** object

- **$_POST['title']** from **<input name="title" />**
- **$_POST['path']** from **<input name="path" />**
- **$_POST['content']** from **<textarea name="content"></textarea>**

If any of these values are not set, we won't have enough information to create a database record, and should exit processing and display an error message in the page:

```php
if ( !isset( $_POST['title'], $_POST['path'], $_POST['content'] ) ) {
  exit( '<p>Error: Could not create page - missing data.</p>' );
}
```

When the **exit** statement runs, it prints a string into the HTML output then ignores the rest of HTML and PHP the code in the page.

Note that strings in PHP can use single or double quotes.

- **Single quotes** Print the string exactly as it appears in the code.
- **Double quotes** Print the string using **variable interpolation**

What is variable interpolation? When using double quoted strings in PHP, any variables that appear within the quotes will automatically be replaced with the variable's value.

Test the page in your browser. Because we are loading the page directly without performing a form submission, you should be able to see the error message:

2. Now try accessing the page via the form created in Activity 2.

Open **http://localhost/week11/add.php** in your browser. Fill in the form and submit it. The browser will load the **save.php** page and no error will be shown.

You're now ready to save some data to the database!

Saving a record to the database

1. Before we save data from a form into a database, we should make sure that any special characters that might be in the data are correctly formatted for the database.

   Add the following after the closing brace of the **if** statement:

   ```
   }

   $title = $db->real_escape_string( $_POST['title'] );
   $path = $db->real_escape_string( $_POST['path'] );
   $content = $db->real_escape_string( $_POST['content'] );
   ```

   We now have 3 new variables that contain sanitised versions of the data from the form. The **$db** object's **real_escape_string()** method encodes the form data using characters the database can accept.

2. We can use the 3 variables to construct an SQL query statement. Create a variable called **$sql** to store the SQL query command:

   ```
   $content = $db->real_escape_string( $_POST['content'] );

   $sql = "INSERT INTO pages (title, path, content) VALUES()";
   ```

   The query is a string that contains SQL code:

   - **INSERT INTO** — The database operation (adding a new record)
   - **pages** — The name of the database table to write the data to
   - **(title, path, content)** — names of table fields that data will be stored in
   - **VALUES()** — An SQL command that specifies data to use as field values

   Provide data to the **VALUES()** command by inserting the 3 variable we created in step 1:

   ```
   INTO pages (title, path, content) VALUES ( '$title', '$path', '$content' )";
   ```

   Note that each variable is enclosed in single quotation marks because SQL requires this data to be provided as strings. PHP's variable interpolation replaces the variables with their values.

3. With the statement complete, a database query can be executed:

   ```
   $sql = "INSERT INTO pages (title, path, content) VALUES ( $title, $path, $con
   $result = $db->query( $sql );
   ```

   The result of the execution is assigned to a new variable **$result**. For a query that insert new records, the variable will contain a **true** value if the query is successful, or **false** if it is unsuccessful.

   Add a final **if** statement at the end of this PHP code to output messages depending on the result of the database query:

```
if ( $result ) {
  echo '<p>Success: New page added.</p>';
} else {
  echo '<p>Error: Could not save page.</p>';
}
```

Test the page by submitting data using the **add.php** form. You should see a success message.

Displaying a list of database records

1. Open the listing page in your browser: **http://localhost/week11/list.php**

   This page will display a table that lists the pages saved in the database and provide links to view each page directly.

   Open **list.php** in your code editor and add code at the top of the file to open a database connection.

```
<?php
$db = new mysqli ( 'localhost', 'root', 'root', 'fit1050' );
?>
```

   Also add code at the bottom of the file to close the connection

```
<?php
$db->close();
?>
```

   Within the **main** element, add PHP tags where the table will be created:

```
<?php

?>
```

2. Within the PHP tags, we can set up up an SQL query statement to get records from the database:

```
<?php
$sql = 'SELECT id, title, path FROM pages';
?>
```

   This SQL query is a little different from the one used earlier to save data:

   - **SELECT** The database operation (retrieving one or more records)
   - **id, title, content** Names of database fields to get data from
   - **FROM pages** Which database table to retrieve records from.

   Next, we execute the query and store the result in a variable:

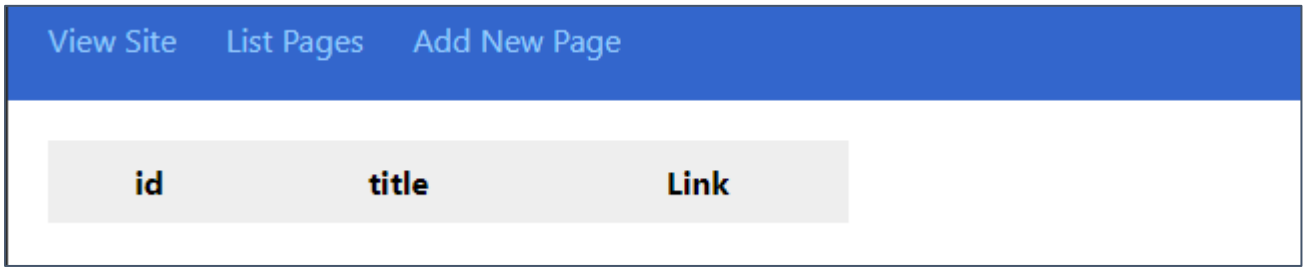
```
$sql = 'SELECT id, title, path FROM pages';
$result = $db->query( $sql );
```

The result will be an object that contains data for all matching records – which could be 0 or more rows of the data depending on what was found in the database.

3. Continue adding to the PHP code by printing the start tag of a table element, as well as the table row containing header cells:

```
$result = $db->query( $sql );

echo '<table>
  <tr>
    <th>id</th> <th>title</th> <th>Link</th>
  </tr>';
```

Test your code by reloading the page in your browser:



4. To display the records in table format we need process each record one-by-one and output the data as a new table row.

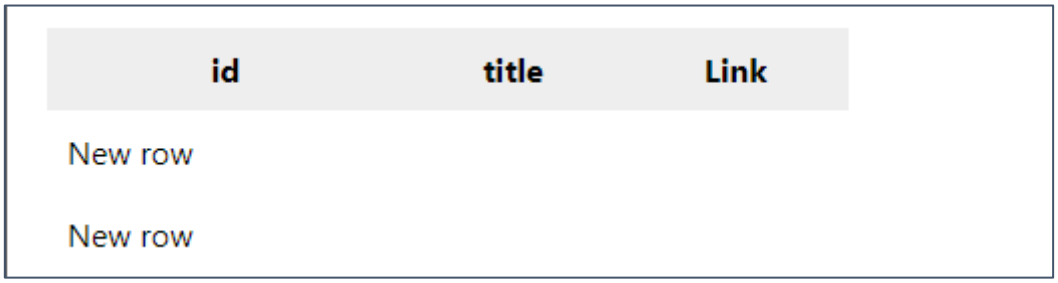
   This can be done by using a **while** loop to iterate through the **$result** object and print out a new table row containing a table data cell.

```
while ( $row = $result->fetch_assoc() ) {
  echo "<tr>
    <td>New row</td>
  <tr>";
}
```

The **fetch_assoc** method gets the next available record from the **$result** object as an associative array and stores it in **$row**. In a **while** loop, each record is fetched and processed one at a time.

Note that we are also using double quotes here so that we can use variable interpolation later.

Test the page and you should see rows matching the number of records saved in your database.



5. We can now populate the rows with real data from the database. Remove the words **“New Row”** from the code replace with 3 empty **td** elements to create table cells:

```
echo "<tr>
  <td></td>
  <td></td>
  <td></td>
<tr>";
```

Because this **echo** uses double quotes, we can easily populate the cells using **$row** variable and accessing form fields using the field names as array keys

```
<td>$row[id]</td>
<td>$row[title]</td>
<td>$row[path]</td>
```

6. Don’t forget to close the table element after the end of the **while** loop’s closing brace.

```
}

echo '</table>';
?>
```

Test the page again. This time you should see values from the database.

id	title	Link
1	Welcome to my website	home
2	This is Page 2	page2

7. The text displayed in the **Link** column can be turned into a clickable link with a little extra HTML.

Replace the content of the third **td** element with HTML code for a hyperlink:

```
<td>$row[id]</td>
<td>$row[title]</td>
<td><a href='index.php'></a></td>
```

This creates a link to the homepage (**index.php**) in the **week11** folder. Note that the value of the hyperlink is using single-quotes because double-quotes are being used by the surrounding echo.

Use a query string to pass information to **index.php** about which page to load.

```
<td><a href='index.php?view=$row[path]'></a></td>
```

For example, if the page’s **path** is **"home"**, the resulting URL is **"index.php?view=home"**.

Finally, we can use the path value again to add text within the hyperlink element.

```
<td><a href='index.php?view=$row[path]'>$row[path]</a></td>
```

When you reload the page, you should see a clickable links for each page:

id	title	Link
1	Welcome to my website	home
2	This is Page 2	page2

Note that the **id** value for each new record is being assigned automatically – when this field was created in phpMyAdmin, its **A**_I (auto-incrementing) option was enabled.

Displaying different pages using a single page template

1. The **index.php** page is the document that will be used to display user-facing pages of the website. Think of it as a template page that can hold different content.

   It can be accessed using the following URLs in your browser:

   - **http://localhost/week11/index.php**
   - **http://localhost/week11/**

   By default, when a folder URL is requested, the server will try to return the folder's index page. Appending a query-string to the URL passes the query variables to the server as a GET request.

2. Open **index.php** in your code editor. Like the other pages, this contains normal HTML code – the code is almost identical to the week 10 lab activities.

   Add code at the top of the file to open a database connection.

```
<?php
$db = new mysqli( 'localhost', 'root', 'root', 'fit1050' );
?>
```

   Also add code at the bottom of the file to close the connection

```
<?php
$db->close();
?>
```

   Within the **article** element, add some PHP tags. This is where a database query will be used to retrieve content for whatever page needs to be displayed.

```
<?php

?>
```

3. The page will decide which database record to fetch by using a query string variable. For example, if we wanted to load the homepage, we would have a query containing the path **"home"**:

   http://localhost/week11/index.php?view=home

   But if the page is loaded without a **view** variable in the query string, we should display the home page by default anyway.

   We can use an **if** statement to check if a **view** variable has been set in the query string, and if it has not been set, we assign it the default value of **"home"**:

```
<?php
if ( !isset( $_GET['view'] ) ) $_GET['view'] = "home";
?>
```

   This allows the website home page content to be accessed using 3 different URLs:

   - **http://localhost/week11/**
   - **http://localhost/week11/index.php**
   - **http://localhost/week11/index.php?view=home**

   If you test the page now, you will see a blank page because the code to query and display the page content has not been added yet.

4. We'll use the value of **$_GET['view']** to perform a database query. Before we use the value we should make sure it is correctly encoded for the database.

   Add the following code to create a new **$view** variable with the properly encoded value.

```
if ( !isset( $_GET['view'] ) ) $_GET['view'] = "home";

$view = $db->real_escape_string( $_GET['view'] );
```

   Use the variable in an SQL statement that searches the **pages** table for records that have a **path** that matches the value of **$view**:

```
$view = $db->real_escape_string( $_GET['view'] );

$sql = "SELECT * FROM pages WHERE path='$view'";
$result = $db->query( $sql );
```

5. If the query is successful, the result should contain a single row for the page we want to display. However, there is a possibility that the **view** value might not match any of the page paths in the database. If this is the case, the result returned by the query will have **0** rows.

   We can check if the query failed to match any records by checking the number of rows in the result.

```
$result = $db->query( $sql );

if ( !$result->num_rows ) {

} else {

}
```

   If no rows have been returned, print out an error message.

```
if ( !$result->num_rows ) {
  echo '<h2>The requested page could not be found.</h2>';
} else {
```

   Test the page by setting the URL's view value to a random word:

6. If the query result contains 1 or more rows, the **else** statement will be executed instead. Even with a single row, we can iterate through the result set. Add a **while** loop within **else** statement – this will be very similar to the loop used in **list.php**:

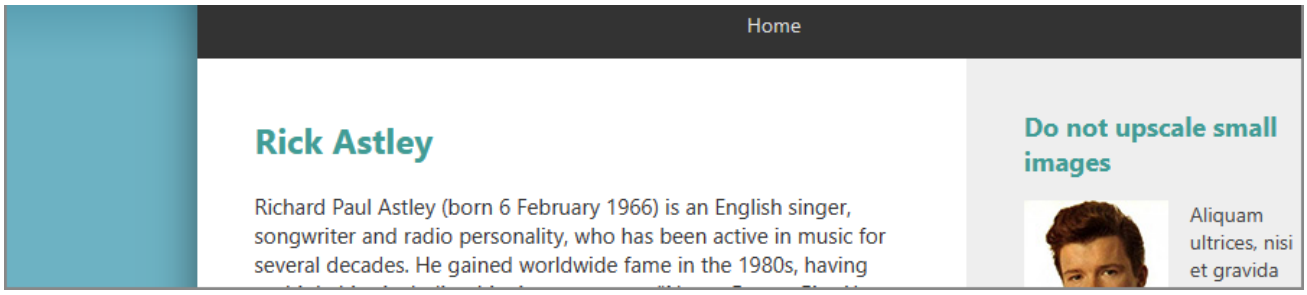
```
} else {
  while ( $row = $result->fetch_assoc() ) {

  }
}
```

7. Within the **while** loop, the **title** can be printed as an **h2** element, and the **content** as a paragraph:

```
while ( $row = $result->fetch_assoc() ) {
  echo "<h2>$row[title]</h2>";
  echo "<p>$row[content]</p>";
}
```

   Test the page by viewing the **home** URL:

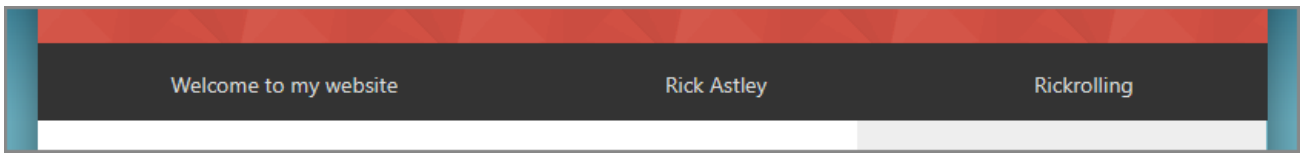
Go back to **http://localhost/week11/list.php** and try viewing the pages for other records in your database. The single **index.php** template file is used to display every page of the website.



Challenge tasks!

We now have a set of documents that make it possible to create, view and display pages in the website, but there's still ways to improve it:

1. Using the technique from **list.php**, query the database and automatically generate working hyperlinks in the navigation menu for each database record.



2. **Summernote** is a third-party JavaScript that adds formatting options to textarea elements. Implement Summernote in **add.php** to allow HTML formatted content to be saved to the database.

    - Summernote homepage: **https://summernote.org/**
    - Documentation: **https://summernote.org/getting-started/**