Server side Programming - PHP

Objectives

- Set up a local web server which supports PHP
- Create a web page to display specific data from the database
- Create a HTML form to add new data into the database and add HTML buttons to delete data from the database
- You can pair program for this lab exercise. (optional)

Pre-lab work

1. Make sure you have a working copy of the database instance used in Lab5. You need not install the MySQL server again for this lab.
2. Install a web server that provides support for PHP. You can pick any web server with PHP support. Following are some popularly used software stack:
   Windows:
   a. XAMPP
   b. WampServer
   c. PhpMyAdmin
   Ubuntu:
   d. LAMP
   OS X:
   e. Default Apache

For this lab, we are going to use Apache web server on Ubuntu VMs to run the PHP scripts. We are going to write PHP scripts which connect and retrieve data from our Lab5 MySQL database tables. These PHP Scripts are run on apache web server and the result is displayed on our browser as HTML pages. To get the required server setup working, we need to do the following:

1. Install Apache web server

Command to install apache web server is:
```
sudo apt-get install apache2
```

Command to check if your installed apache server is up and running:
```
service apache2 status
```

Command to check the apache error logs when there are some bugs in the php programs:
```
tail -f /var/log/apache2/error.log
```

To check if your apache installation was successful access this link on your browser:

http://127.0.0.1/

Note: This link should result in a web page with the heading: *Apache2 Ubuntu Default Page*
More information about apache web server is available here.

2. Install PHP

Before installing PHP, you need to update the libraries on your machine by running the following command:

```
sudo apt-get update
```

Command to install PHP:

```
sudo apt-get install php
```

Command to install packages that help us to access MySQL database server using PHP:

```
sudo apt-get install php-mysql libapache2-mod-php
```

3. Check your existing MySQL installation

In Lab5, we installed MySQL database server and created a database and populated it with some tables. In this lab today, we are going to use the *store* table from the database we created previously.

Try to connect to the mysql instance on your machine:
Command: `sudo mysql -u root -p`

If your mysql instance is not up and running, you will not be able to connect to mysql. Check if your mysql instance is running.

Command: `mysqladmin -u root -p status`

If your instance is not up, please start the mysql instance by running the following command:

```
sudo service mysql start
```

After connecting to the database instance, run the commands - `show databases;` `use <yourdatabasename>;` and `show tables;` to see if your database has the *store* table.

Note: Please restart your apache server after running all the above commands before you start the exercise. The command to restart the apache server is: `sudo service apache2 restart`

Exercise

The goal of this exercise is to write server side scripts using PHP to create web pages with dynamic content. PHP is extensively used for developing social media websites, email management, content management software, chat forums and ecommerce stores. In today's lab more specifically, you will create a web application that not only displays the items from your *store table* to the web page, but also allows users to add or delete items into the *store* table from the web page.

Part 1 – Displaying data from database on HTML page

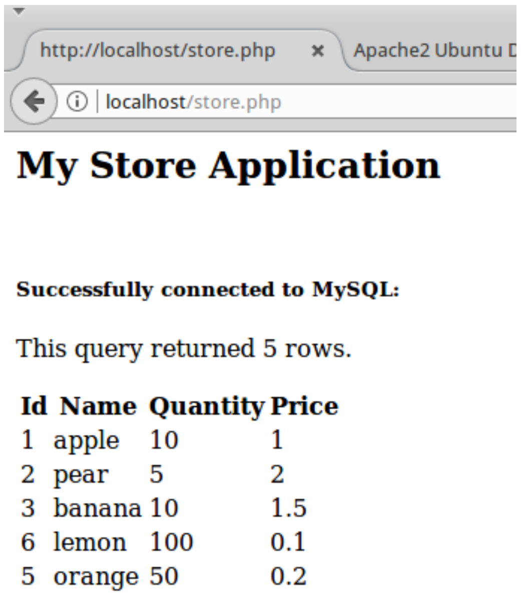
While checking for the successful installation of apache2 server, we accessed the URL http://127.0.0.1/. This rendered a default web page on your browser. Where did we get this web page from? Why did we access that URL?

1. Apache server on installation, by default runs on port 80. When you access any url without providing a port number, the default port number the browser considers is 80. For example, http://127.0.0.1/ is same as http://127.0.0.1:80/
2. Apache server on receiving a request responds with a default page named index.html present in the following directory: /var/www/html/
3. If you want to access any other web page in your browser, you can create corresponding html pages or php pages in this folder and access it from the browser. For example, I can create a page Yogitha.html in the directory /var/www/html/ and then I can open the browser and access the html page by typing the url: http://127.0.0.1/Yogitha.html.

In this part of the exercise, we are going to write a PHP script which will

- Connect to the database,
- Get all the records from the *store* table
- Displays the records as HTML on web browser

At the end of part1, you should have something similar to this:



My Store Application

Successfully connected to MySQL:

This query returned 5 rows.

Id	Name	Quantity	Price
1	apple	10	1
2	pear	5	2
3	banana	10	1.5
6	lemon	100	0.1
5	orange	50	0.2

Connecting to the database:

You need the following information to connect to your database using PHP.

1. **MySQL Server address:** You can connect to any MySQL server instance which is reachable from your machine. If the database is on the same machine as the apache web server then the MySQL server address which you could use is **localhost** or **127.0.0.1.** If your MySQL server instance is on a different machine, you need to use that machine's IP address.
2. **Username:** The user name which can be used to connect to the MySQL server instance.
3. **Password:** The password given during the installation of the MySQL server instance.
4. **Database name:** The name of the database which you are trying to connect to.

Once you have all this information you can connect to your database from PHP using the following code snippet:

```
<?php

// Obtain a connection object by connecting to the db

$connection = @mysqli_connect (MySQL server address, username,
password, database name); //  please fill these parameters with the actual data

if(mysqli_connect_errno())
{
   echo "<h4>Failed to connect to MySQL: </h4>".mysqli_connect_error();
 }
else
{
   echo "<h4>Successfully connected to MySQL: </h4>";
}
?>
```

Note: Create a store.php file in the /var/www/html/ directory with the above code and try to access this php file from the browser. You should see the message: *Successfully connected to MySQL:*

Retrieving records from the store table:

Once we have a connection object, we can use this *$connection* object to query your database.

1. First we need to formulate the query to be executed on the database. What is the query/SQL statement to select everything from the *store table* in your database?

2. Create a variable called query (Variables usually start with $ in PHP) and assign the SQL statement which retrieves everything from the *store table*.
   For example: `$query = "Select ---------;";`

3. Once the query variable has been created, we need to execute this query on the database. To do this, we need to use a function named *mysqli_query()*
   For example: `$resultset =  mysqli_query($connection,$query);`
   The results(output) of the query is stored in the variable $resultset.

4. We retrieved all the records from the *store table* and they are currently stored in the variable $resultset.

Displaying the records on the HTML Web page:

Once we have all the records in a PHP variable, we can write a loop to iterate over each record of the *store table*. (iterate over the variable $resultset). To iterate over the $resultset variable, we could use a function named: *mysqli_fetch_array()*
For example:

```
        while ($row = mysqli_fetch_array($resultset, MYSQLI_NUM)) {
                echo $row[0]." ".$row[1]." ".$row[2]." ".$row[3]."<br>";
        }
```

Add the above code after retrieving all the records from the *store table* into the $resultset variable. Save the php file and try to access the file from browser: http://127.0.0.1/store.php

You should be able to see all the records from the *store table* displayed on your web page in the browser.

Note: If you want them to be displayed in a tabular format, like shown in the above picture please use html <table> tags in the echo statement.

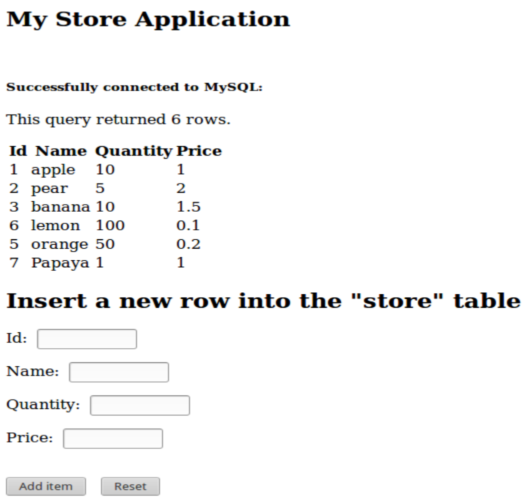
Congratulations! Now you know how to retrieve data from a database table and display it as a HTML web page using PHP.

Part 2 – Inserting a new record into the database table from the UI – HTML web page.

In this part of the exercise, we are going to edit the store.php script to add a html form where user can provide input data and then we create a new PHP script which serves as an action handler to insert the data received from HTML form (user input) into the database table.

- Adding HTML form elements to the existing store.php page
- Creating a new SQLInsertHandler Script.
- Displaying the updated records on the HTML web page

At the end of part2, we should have something similar to this:



My Store Application

Successfully connected to MySQL:

This query returned 6 rows.

Id	Name	Quantity	Price
1	apple	10	1
2	pear	5	2
3	banana	10	1.5
6	lemon	100	0.1
5	orange	50	0.2
7	Papaya	1	1

Insert a new row into the "store" table

Id:

Name:

Quantity:

Price:

Adding HTML form elements to the existing store.php page:

Everything written in between the php open tag (<?php) and php close tag (?>) is treated as php code by the web server. So, if you write HTML in between these tags, PHP throws undefined errors. Instead, we could embed html code in PHP using php *echo* statements.
For example, you could write HTML code inside php in the following way:

```
<?php
echo “<html><head><title>My Store Web Application
</title></head><body> Hello World </body></html>”;
?>
```

The other way of doing it is writing html code outside the php open and close tags.

```
<html>
<head>
<title> My Store Web Application </title>
</head>
<body>
<?php

// Your PHP Code

?>
</body>
</html>
```

Depending on which section of the HTML code you are writing you choose one of the above mentioned ways. For this section, we choose the second way. Now that we know how to embed HTML in PHP, we can proceed to add HTML form elements to our store.php code.

Following is the HTML form code that you need to add to the store.php file:

```
<form enctype="multipart/form-data"
action="http://localhost/SQLInsertHandler.php">
<p>Id:&nbsp <input type="text" name="Id" size="10" maxlength="11"
/></p>
<p>Name:&nbsp   <input type="text" name="Name" size="10"
maxlength="20" /></p>
<p>Quantity:&nbsp       <input type="text" name="Quantity"
size="10" maxlength="30" /></p>
<p>Price:&nbsp  <input type="text" name="Price" size="10"
maxlength="10" /></p>
<br>
<input type="submit" value="Add item" /> &nbsp
<input type="reset" />
</form>
```

After adding the HTML form code to your store.php, try to access the store.php in your browser and check if your form is displayed correctly.

Notes:

1. <p> - stands for paragraph element in HTML
2. <input> tag is used for HTML elements which take some user input. For example, using input tag you can create a textbox, textarea, radio button, submit button, reset button etc.,
3. <form> tag is used to create a HTML form element.
4. All the above html elements have some attributes and attributes are included inside the tags. For example, form tag has two attributes *enctype* and *action*.
5. The <input> tag has an attribute ***type****,* which can be used to specify the type of the element we are trying to create. If type is '*submit',* then a button element is created and when the button is clicked, the form is submitted. If type is '*reset',* then on click of the button the form is reset.
6. **action** attribute for form tag specifies the URL to which the data input from the user (filled form fields) is sent once the user clicks on the '*submit*' button.
7. Here we want the user input data to be sent to a program named SQLInsertHandler.php. (We currently don't have this php file, but we will create this php file to handle the input data from the form – form filled by the user, in the next step).

To read more about these HTML elements please check this link.

Creating a new SQLInsertHandler script to insert the data into the database:

Once the form is displayed, a user can type in the input in the form fields and click on the *submit* button (Add item). Once the button is clicked, we want to insert this data (filled by the user) into our database table – *store*. In this section we are going to create a new PHP script named: SQLInsertHandler.php to handle this data filled by the user and to insert this data in the database table.

This new php file should first retrieve the data filled by the user in the form created by the HTML code embedded in the store.php file and then, it should insert this data into your database. Also to perform any operations like insert, update, select or delete data from the database tables, we first need to connect to the database.
Retrieving the data filled by the user:

1. The data typed by the users is sent to the SQLInsertHandler.php because we mentioned this php file name in the **action** attribute of the form.
2. The SQLInsertHandler.php can access each of the form fields filled by the user, by using a variable named ***$_REQUEST***
3. If we need to access a form element say id, you can access it using $_REQUEST('Id')
   For example, the following code retrieves the *Id* field value typed by the user and it saves it to a php variable named $Id:
   ```
   $Id = $_REQUEST['Id'];
   ```

   Note: What is `'Id'` in `$_REQUEST['Id']`? Id is the name you gave to your HTML form element. Our HTML form element for the Id input text box of the web page is:
   ```
   <input type="text" name="Id" size="10" maxlength="11" />
   ```
   You can see that the name attribute for the input element has a value **`Id`**
   Similarly, you can retrieve all the fields like Id, Name, Quantity and Price.

Connecting to the database and running the insert query:

4. Now that SQLInsertHandler.php file has the data provided as an input by user, we can connect to the database and run a query to insert this data into the database - *store* table.
    1. Write PHP code to connect to the database
    2. Run an ***Insert*** query on the database to insert a row with the user input data into the *store* table.

    Connecting to the database and running a *select* query concepts have been covered in Part1. Please refer to those sections for hints on how to do the above two steps. The only difference here is the SQL statement, instead of SELECT, you would write an INSERT INTO sql statement.
5. After running the query, add a php *echo* statement to print "Inserted successfully into the database" on your web page.
6. Access the store.php in your browser, type some data in the form and click on submit. You should see the statement displaying a successful Insert.
7. Now connect to your MySQL instance, switch to your database and check if the record has been successfully inserted into the *store* table.

Displays the updated records on the HTML web page:

Instead of just displaying the success message on the HTML web page, we could redirect from SQLInsertHandler.php page to the initial store.php page which retrieves all the records from the database along with the newly inserted record. Following php code does this redirection:

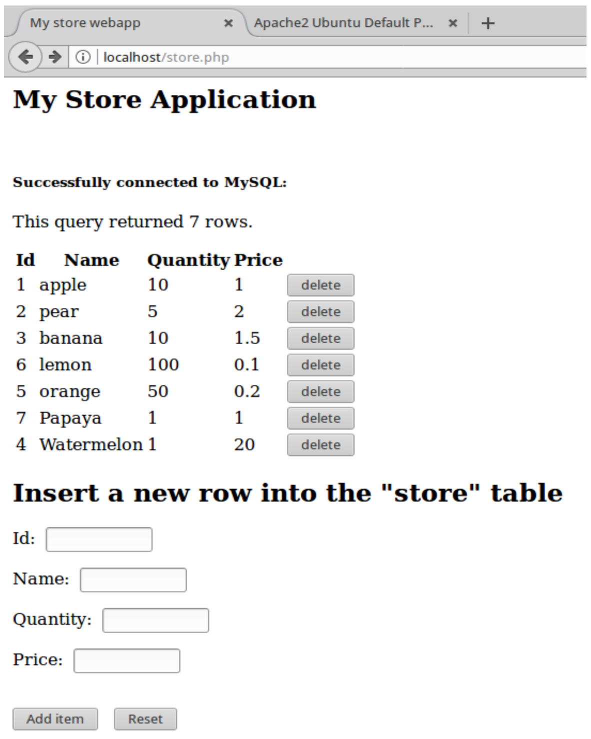
```
    include 'store.php';  // You can include this after the insert query in SQLInsertHandler.php
```

Congratulations! Now you know how to use PHP for adding data to your database tables from UI (web page) consisting of HTML form elements.

Part 3 – Deleting a record from the database store table from the UI.

In this part of the exercise, you are first going to provide a delete button for each of the *store* table records. Once a user clicks on the *delete* button beside an item in the store, that particular item should be deleted from the database table '*store*' and also the store.php listing shouldn't have that entry anymore.

At the end of part3, we will have something similar to this:



My Store Application

Successfully connected to MySQL:

This query returned 7 rows.

Id	Name	Quantity	Price	
1	apple	10	1	delete
2	pear	5	2	delete
3	banana	10	1.5	delete
6	lemon	100	0.1	delete
5	orange	50	0.2	delete
7	Papaya	1	1	delete
4	Watermelon	1	20	delete

Insert a new row into the "store" table

Id:

Name:

Quantity:

Price:

To achieve this, we need to do the following:

1. First we need to add a delete button for each and every item in the store.php item listing.

   To add a delete button for each item, we need to include a button in the code which is listing the store items in store.php.

   In store.php file, we are using echo statement in the while loop to display the data from the *store* table. Now to add a delete button, we need to add the following html code to the same echo statement.

   ```
   echo "<input type=\"submit\" class=\"button\" name=\"".$row[0]."\"
   value=\"delete\"/>";
   ```

   Once we add this echo statement in the while loop you will see a delete button for each of the items listed in store.php file. After adding the above code, please reload the store.php in the browser and make sure you see the delete buttons after each item in the listing.

2. Now we need to handle what happens when a user clicks on the *delete* button. To call a php file on a button click, we need to use ajax and jQuery. (similar to what we have done in Lab6)

   For using ajax and jQuery functions, we need to include jQuery library in our HTML code using the following script tags. Please place the following script tag in the header section of the html code.

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.mi
n.js"></script>
```

   Now that we can use ajax and jQuery, write code to call the SQLDeleteHandler.php file which handles the deletion of the items from the database, on click of the delete button.

   Following is the code snippet to do this:

```
<script>
$(document).ready(function(){
   $('.button').click(function(){
    var clickBtnName = $(this).attr('name');
    var ajaxurl = 'http://localhost/SQLDeleteHandler.php';
    var data = {'id': clickBtnName};
    $.post(ajaxurl, data, function(response) {
     window.location.href="http://localhost/store.php";
});
});
});
</script>
```

   The above given javascript code transfers the control to the SQLDeleteHandler.php when a button with class name *'button'* is clicked. Along with transferring the control (sending a request), it also passes some data (name attribute of the button clicked – nothing but the id of the item that needs to be deleted) to the SQLDeleteHandler.php file.

3. The third and last step is to create the SQLDeleteHandler.php file. This php file should have code to get the *id* data sent by the store.php user action – button click and then it should also delete the record with this *id* from the *store* table.

   You can retrieve the id from the request in the same way as mentioned for SQLInsertHandler.php.

```
     $id = $_REQUEST['id'];
```

   Once the id is known, we can connect to the database using PHP and then execute a query to delete the record with this id.
   Connecting to the database using PHP and executing a select query is explained in part 1. You can refer to those concepts to get hints on how to finish this task. However, the only difference would be the SQL statement/ SQL Query. It was a SELECT statement there and here it is going to be a DELETE statement.

Once the record is deleted, the control is passed back to the store.php ajax call. We can now ask the browser to reload the store.php upon successful deletion, using the following code:

```
window.location.href="http://localhost/store.php";
```

(This code is already included in the above javascript code, you need not write it again.)

Once your SQLDeleteHandler.php code is ready, open the browser and load the store.php file. Click on the delete button beside one of the store items. On click of the delete button, the store.php page is reloaded and you can't see the deleted element anymore in the store items listing.

You could also confirm the deletion of the record by connecting to the MySQL database instance and looking at all the records of the *store* table.

This is it ☺ Congratulations now you know how to do CRUD operations on the MySQL database using PHP. Now, you have all the knowledge to build a working real-time website using PHP and MySQL.

Credit: To get credit for this lab exercise, please submit a zip file named YourFirstName_YourLastName_Lab8.zip on Moodle. The zip file should contain the store.php, SQLInsertHandler.php and SQLDeleteHandler.php files. Please include both the names as comments in all the files if you pair programmed on this lab exercise.