WAMP Apps

Overview

This lab walks you through using Windows, Apache, MySQL and PHP (WAMP) to create simple, yet very powerful PHP applications connected to a MySQL database. For developers using Linux, the acronym becomes LAMP (Windows (W) is replaced by Linux (L)). The basics of inserting, updating, deleting and selecting from MySQL using PHP forms will be provided. Some "bad" security practices that lead to SQL injection vulnerabilities will be exposed as well as some techniques to mitigate these issues. Injection, and particularly, SQL injection continues to plague web applications that use any programming language and a database. Caution and best practices must be used starting in the initial design of an application to eliminate these significant software vulnerabilities.

Learning Outcomes:

At the completion of the lab you should be able to:

1. Create least privilege MySQL accounts for the Web connection
2. Insert data into a MySQL database using PHP forms
3. Query existing data in a MySQL database using PHP forms
4. Delete data from a MySQL database using PHP forms
5. Update data in a MySQL database using PHP forms
6. Identify and use best practices to eliminate SQL injection

Lab Submission Requirements:

After completing this lab, you will submit a word (or PDF) document that meets all of the requirements in the description at the end of this document. In addition, your associated files should be submitted. You should submit multiple files in a zip file.

Virtual Machine Account Information

Your Virtual Machine has been preconfigured with all of the software you will need for this class. You have connected to this machine in the previous labs. Reconnect again using the Remote Desktop connection, your Administrator username and password.

We will first use a technique very susceptible to SQL injection and then a better approach using prepared statements. Note, the first technique is what **NOT** to do. It is provided so you can easily identify this issue in future code.

In addition, when connecting to a database through a Web interface, the connection credentials should never be from the root account. If your Web connection becomes compromised and the account had root privileges to your database, the hacker now has root privileges to the database as well. The **least privilege** rule should always be adhered to when assigning user privileges.

To make sure the root account is not used for web connections, we need to create a couple of new MySQL users and a new database. One user will be the owner of the new database and have most

privileges. The other user will be a restricted user with access to only the tables and the privileges for those tables that are needed. Other courses (e.g. SDEV 350) discuss database roles and privileges in more depth. For this course, just a few statements will be used so that you can follow and model for additional activities for this lab.

1. Assuming you have already launched and logged into your SDEV AMI from your remote desktop, As shown in figure 1, open your command prompt and change to the Bitnami\WampStack-7.1.16-0\mysql\bin directory and enter the following command:

   ```
   mysql -u root -p
   ```

   When prompted enter the following mysql root password:

   ```
   sdev300vm99
   ```

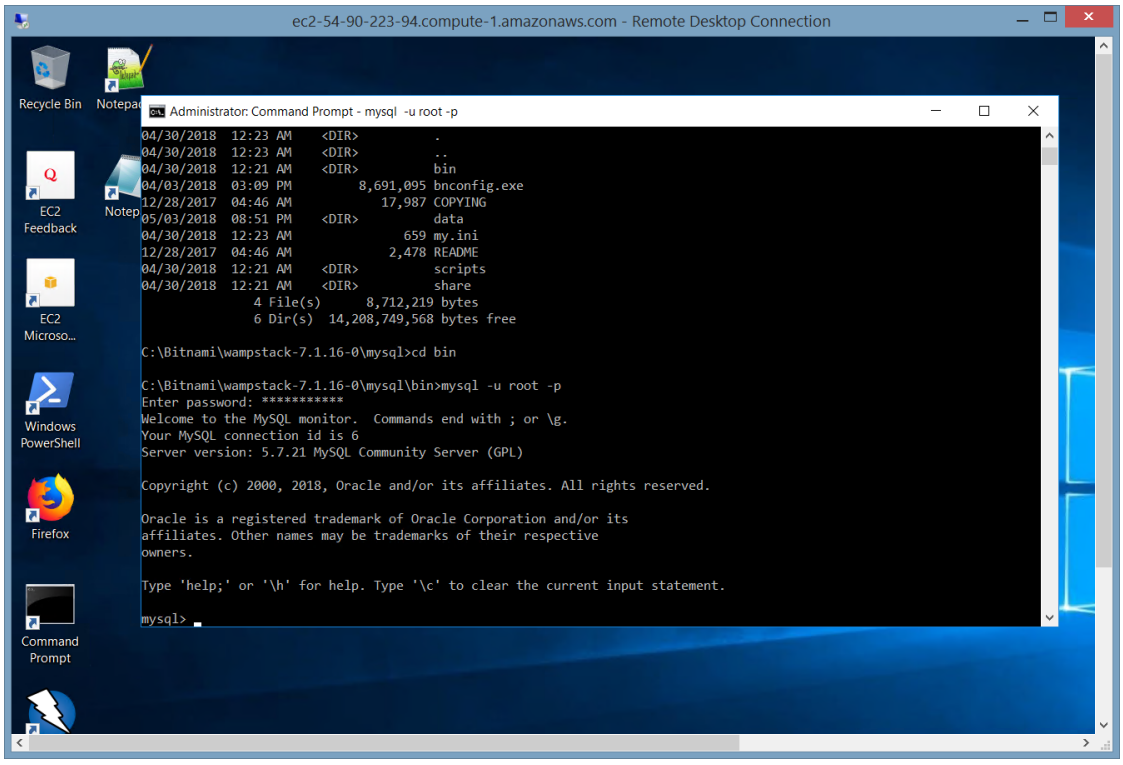

Figure 1 Launch a MySQL connection

2. To display the available databases type the following at the mysql prompt:

   ```
   show databases;
   ```

The following SQL statements can be run at the MySQL command prompt to create a database named wamp along with a privileged and non-privileged user. We will be adding more grant statements as additional functionality is required for the non-privileged user.

```
create database wamp;
-- create a user called wamp_owner for the host machine
-- Grant all privileges to this user
-- select password('wamp4umuc') Note: Remove this!
-- Need to grant *.* or would not have the FILE and other global permissions
GRANT ALL PRIVILEGES ON *.* TO wamp_owner@localhost IDENTIFIED BY PASSWORD
'*D7168872D493E342B59B692FF863431330897967';

-- select password('user4wamp') Note: Remove this!
GRANT SELECT ON wamp.students TO wamp_user@localhost IDENTIFIED BY PASSWORD
'*E090926B474658359CB983C65B768047DF3A55BE';
```

Several comments are warranted to better understand these SQL statements:

a. These statements should be run as the root user. The root user has all privileges and will be able to create databases and users.
b. You can create any database with the statement `create database databasename`;
c. Privileges are grants to specific database and tables using the database.tablename syntax. For example `wamp.students` grants privileges to the students table. Using `wamp.*` grants the privilege to all tables in the wamp database.
d. Common privileges to grant include `SELECT, INSERT, UPDATE and DELETE`. Only grant what is needed.
e. Privileges are granted to a user at a specific host (or machine). Since we are running on the localhost (which by the way is **not** a recommended best practice), the `wamp_user@localhost` will work. However; if the database is on another machine, you would use that host or IP name. For example, `wamp_user@192.1.1.12`.
f. The passwords are encrypted using the `select password ('')` syntax. For example, typing `select password ('user4wamp')` will return the encrypted version displayed above.

Figure 2 shows running of the script listed above to create the database and users on the SDEV AMI.

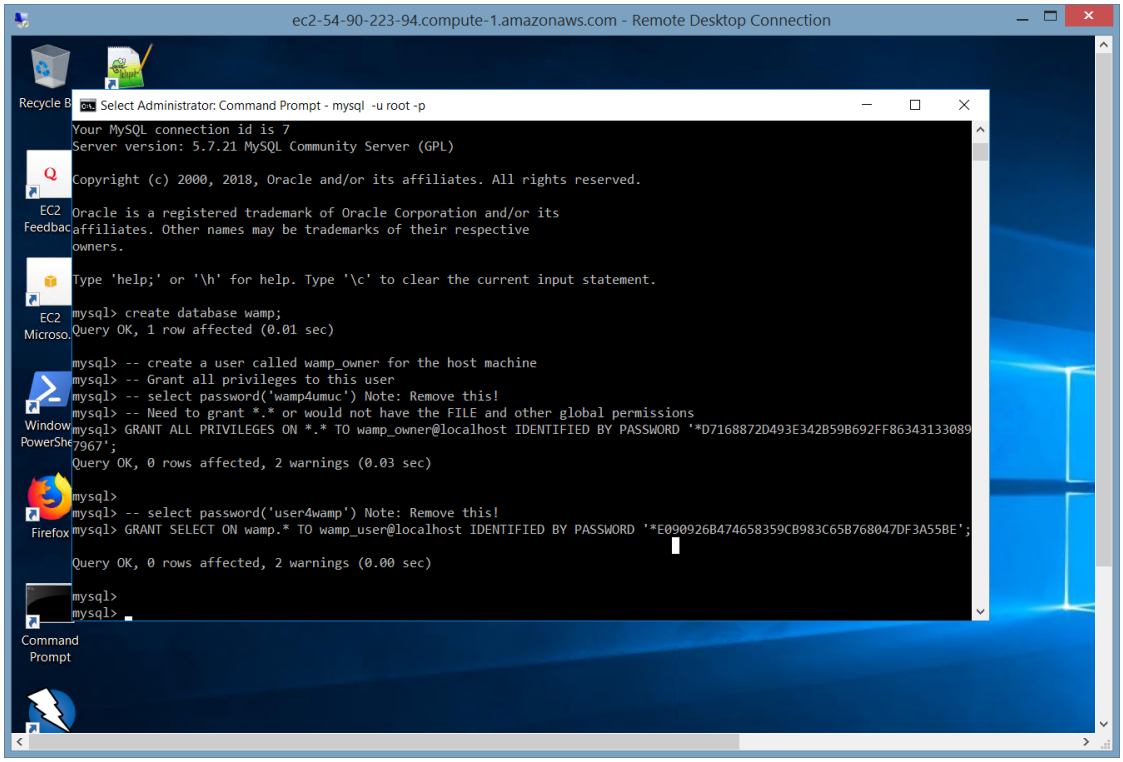

Figure 2 Create a new database and users

Now, that the new database and users have been created, you should connect as the wamp_owner to create the tables and connect as the wamp_user to test the connectivity for the non-privileged user.

For example, to connect at the wamp_owner, (assuming the MySQL console is not running), enter the following statement

```
mysql –u wamp_owner –p
```

When prompted enter the wamp4umuc password as shown in figure 3.

Recall you can type “exit” to exit the root session so you can login back in as the wamp_owner.

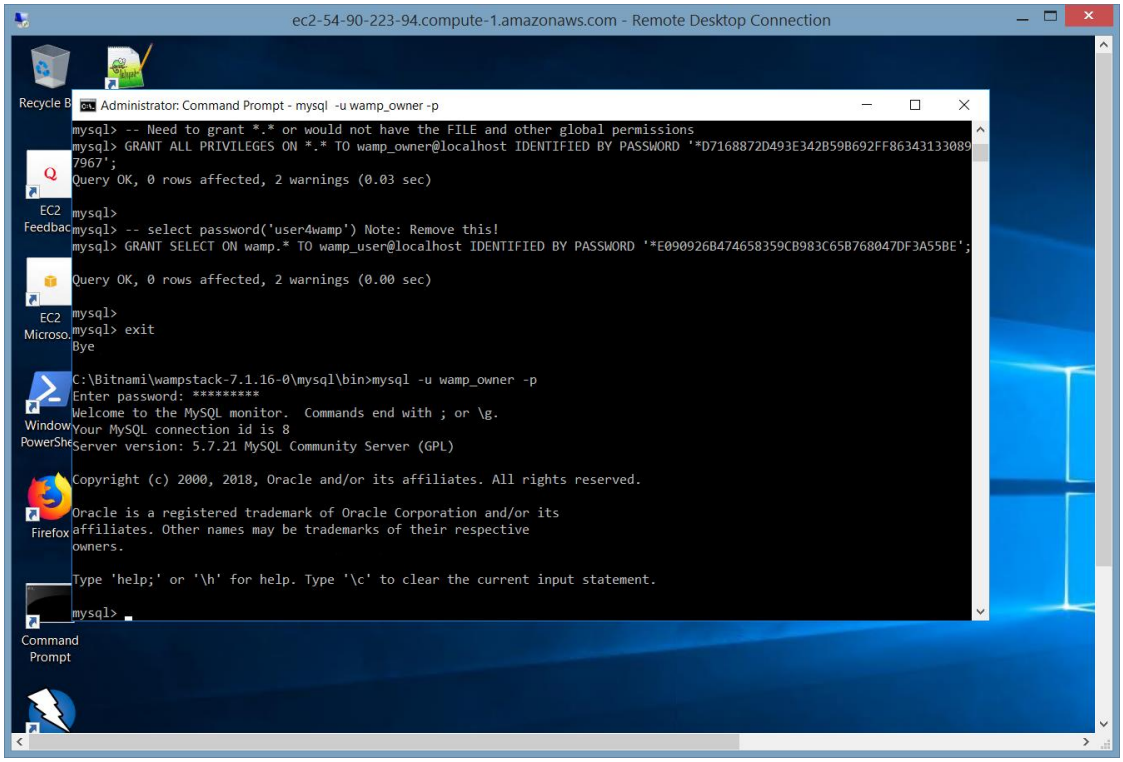

Figure 3 Logging in as the wamp_owner

3. To use the newly created wamp database, type `use wamp`; followed by `show tables;` as shown in figure 4.

   You may already have some tables in your database. If so, the names of those tables would be displayed. If not, you would see Empty set.

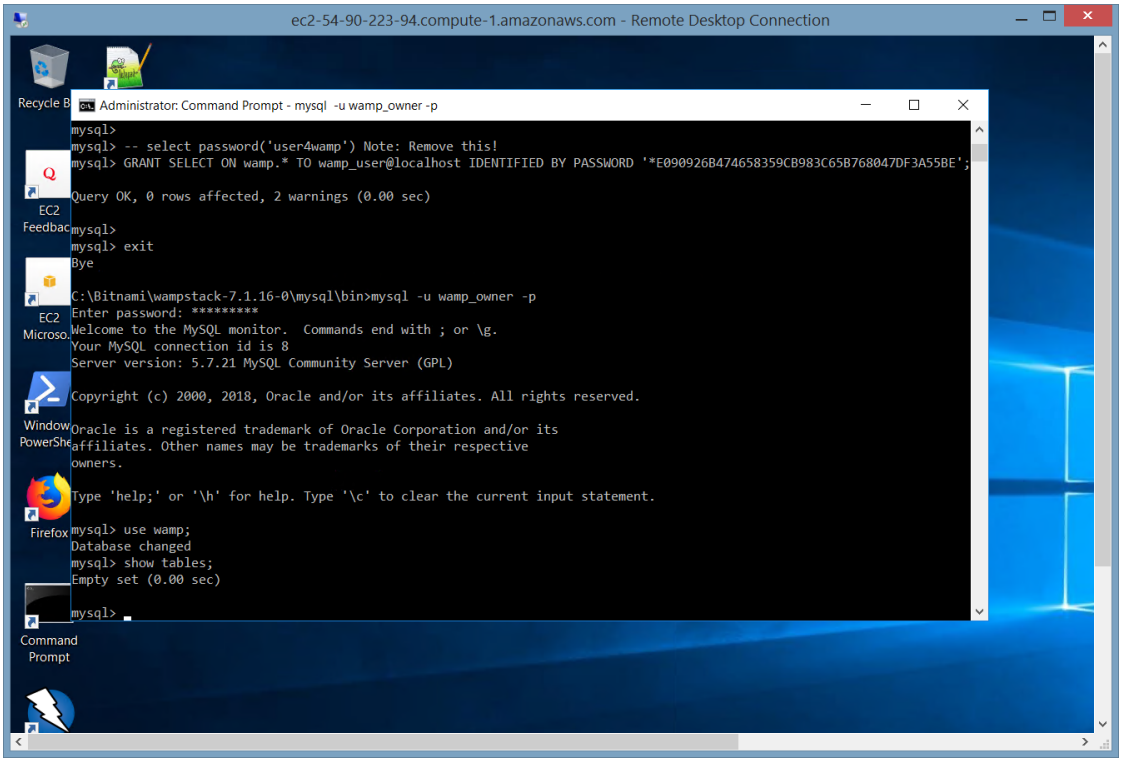

Figure 4 show the tables in the wamp database

4. Create a Students table in the wamp database, if one does not already exist:

```
-- Create a student table
CREATE TABLE Students (
PSUsername varchar(30) primary key,
FirstName varchar(30),
LastName varchar(30),
EMail varchar(60)
);
```

If you already have a Students table and it doesn't match this one, you can delete the current one and then add the new one. Recall to delete a table you type:

```
drop table table_name;
```

Substitute the name of the table you want to drop with table_name. For example:

```
drop table Students;
```

Figure 5 shows the results of creating the Students table in the wamp database.

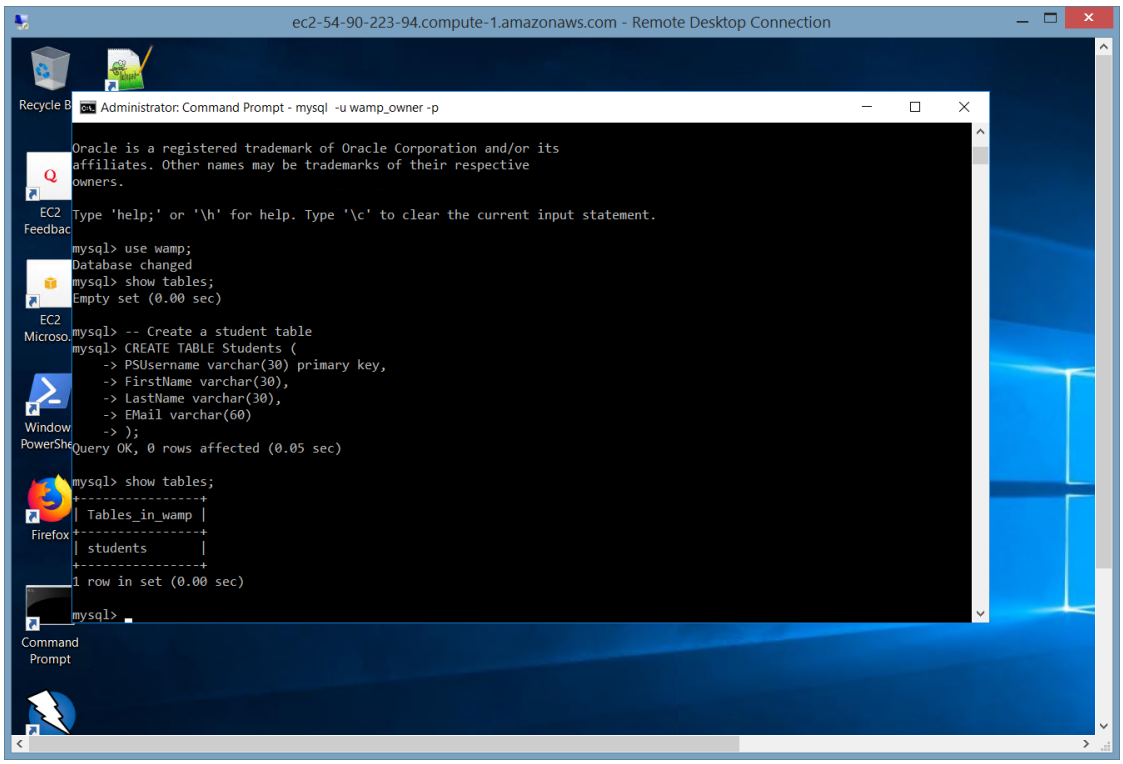

Figure 5 Creating the Students table

5. Next, we will insert a couple of records into the Students table from the MySQL command shell. These records will be used to demonstrate we are actually correctly connecting with the MySQL database table later in this lab.

To insert a two records into the students table, copy and paste the following insert statements and run them at the MySQL command shell. Be sure you are still connected to MySQL and are using the wamp database. (See figure 6.)

```
insert into Students values ('mjones14', 'Mary',
'Jones','mary.jones@student.umuc.edu');

insert into Students values ('gsmith294', 'George',
'Smith','george.smith@student.umuc.edu');
```

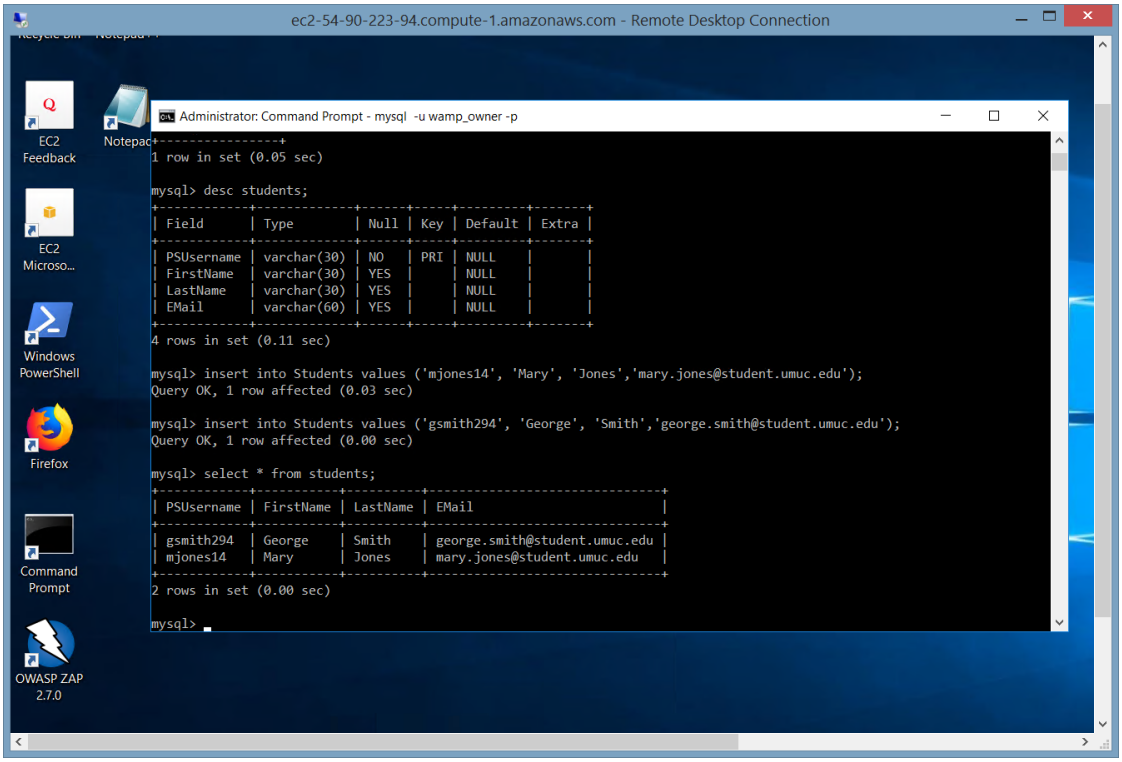

Figure 6 Insert Records into the Students table

6. Next we will create the PHP code that will display the data in the Students table.

Several files are part of this project including StudentApp.html, SelectStudent.php, DBClasses.php, DBConnect.php, DBQueries.php, and Utils.php. These files are found in the wamp.zip attachment. Copy the file to the SDEV AMI, unzip and place the wamp folder into your htdocs folder. You should review and tinker with all aspects of the code to become comfortable with the functionality.

The location and overall function of each of the files is as follows:

a. StudentApp.html – Landing page for the application allowing a user to select from one of many options. Location : wamp/
b. SelectStudent.php – PHP script providing display of all records in the Students table. Location: wamp/
c. DBClasses.php – PHP script supporting the Students and Parameters objects. Location: wamp/Includes
d. DBConnect.php – PHP script providing connection to the database. Location: wamp/Includes
e. DBQueries.php – PHP script providing the SQL query functions. Location: wamp/Includes
f. Utils.php – PHP script providing file utility and other needed functions. Location: wamp/Includes

As we add on the additional functionality, each PHP file will contain similar functionality yet specific to the database query. For example, when we add the Insert functionality, the Insert SQL functions will be placed in the DBQueries.php file. If we need to add additional classes, they would be added to the DBClasses.php file. This design allows for queries and similar functionality to be easily located for debugging and documentation.

Also, notice the use of the `require_once()` PHP code. For example, `require_once('Includes/DBConnect.php');` in the DBQueries.php file allows access to the functions found in the DBConnect.php file. This is useful for code reuse but can be tricky as you need to make sure you include the correct and all files needed.

Once unzipped, and placed correctly, you will see the wamp folder in your htdocs folder as shown in figure 7.

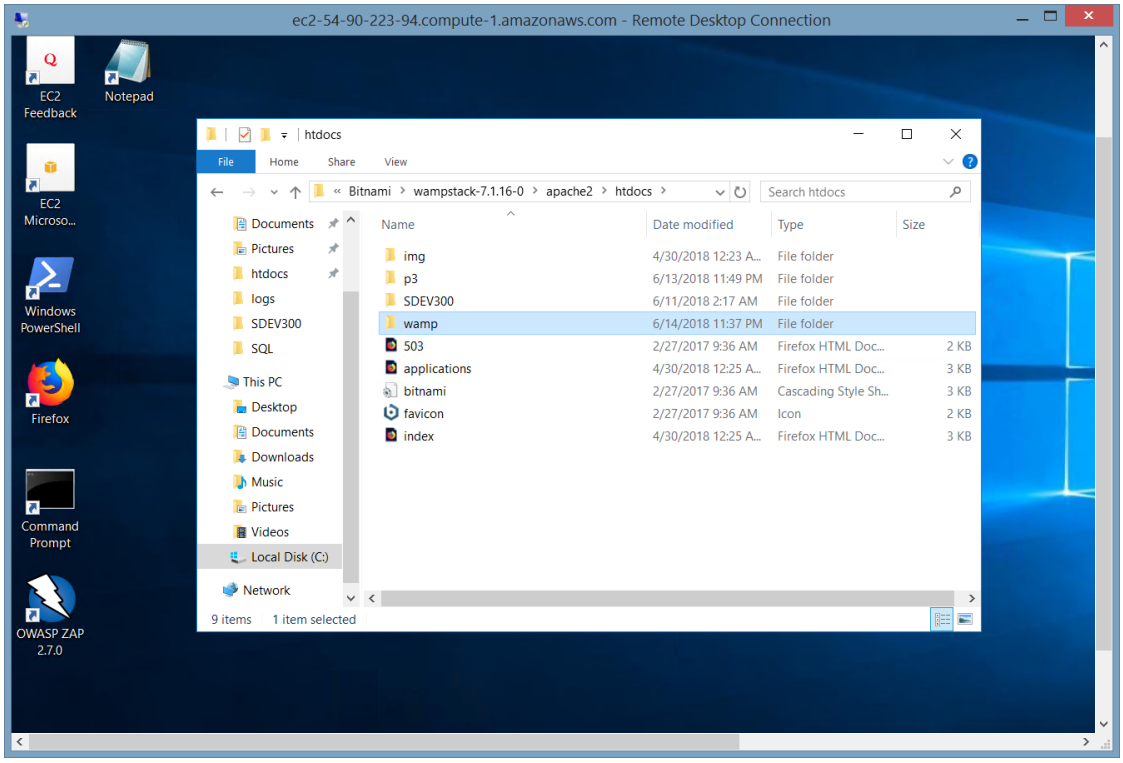

Figure 7 The wamp application resides in the htdocs folder

Verify you have the correct folder structure or your application will not run as expected. The wamp folder should be directly under the htdocs. As shown in figure, if you click on the wamp folder you should see 2 folders: Includes and parms and two files: StudentApp.html and SelectStudent.php. We will be adding to these folders throughout this lab. But for now, that is the structure and the contents.

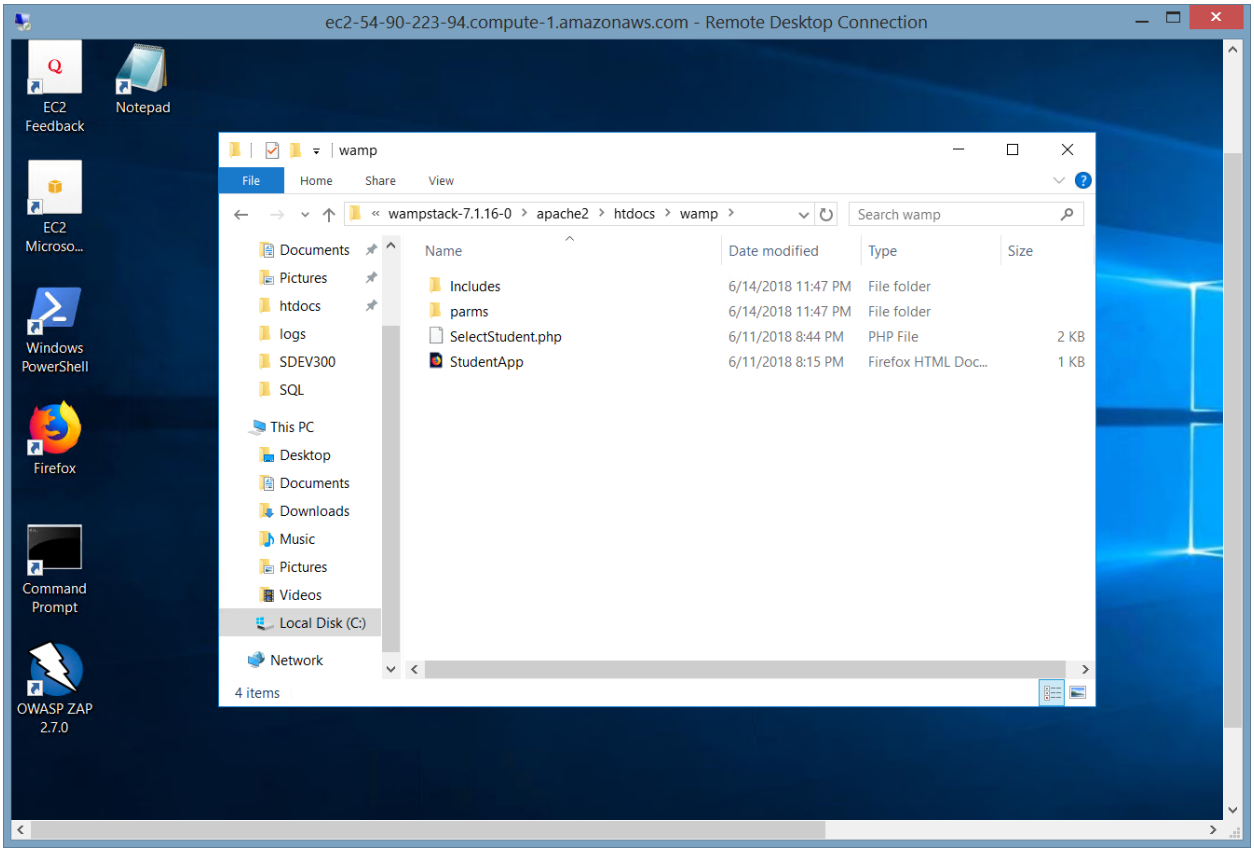

Figure 8 Initial Contents of the wamp folder

To run the application, open the Firefox browser on your SDEV AMI and type in localhost/wamp/StudentApp.html

As shown in figures 8, the html page will provide several options for PHP links. Note at this point, only the DisplayStudent.php is functional.

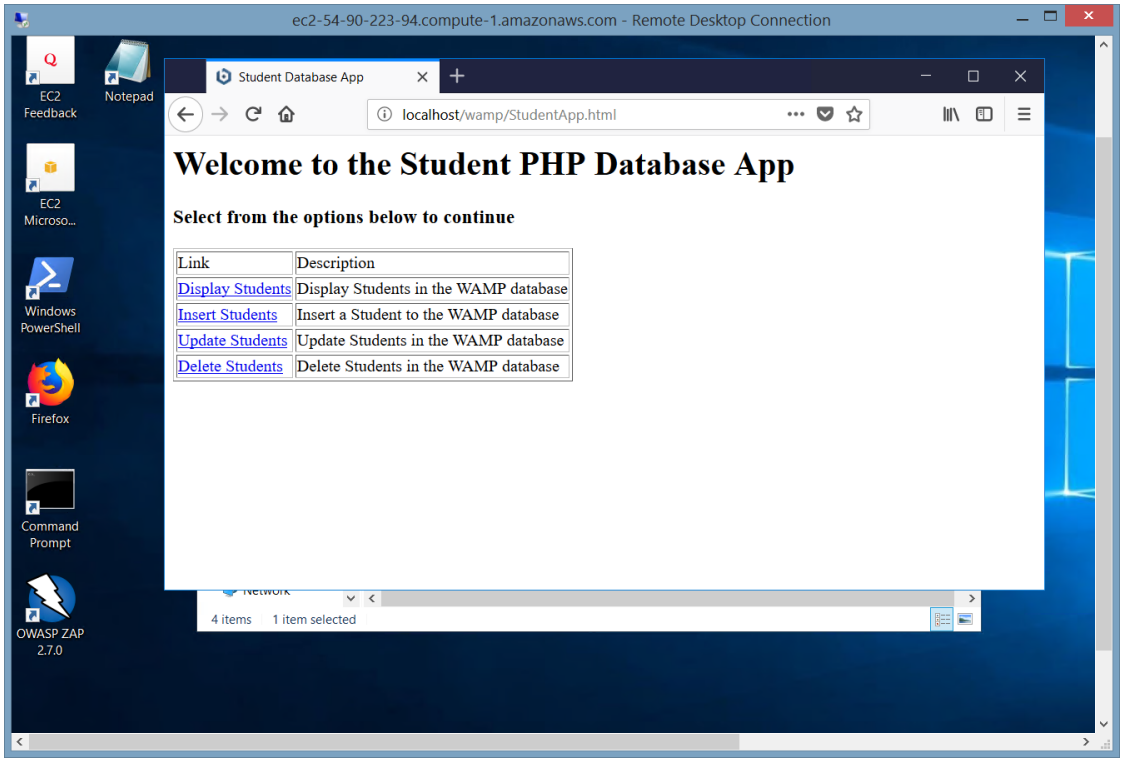

Figure 9 StudentApp Landing Page

Clicking on the Display Students link will show the two students in our database. (See figure 10).

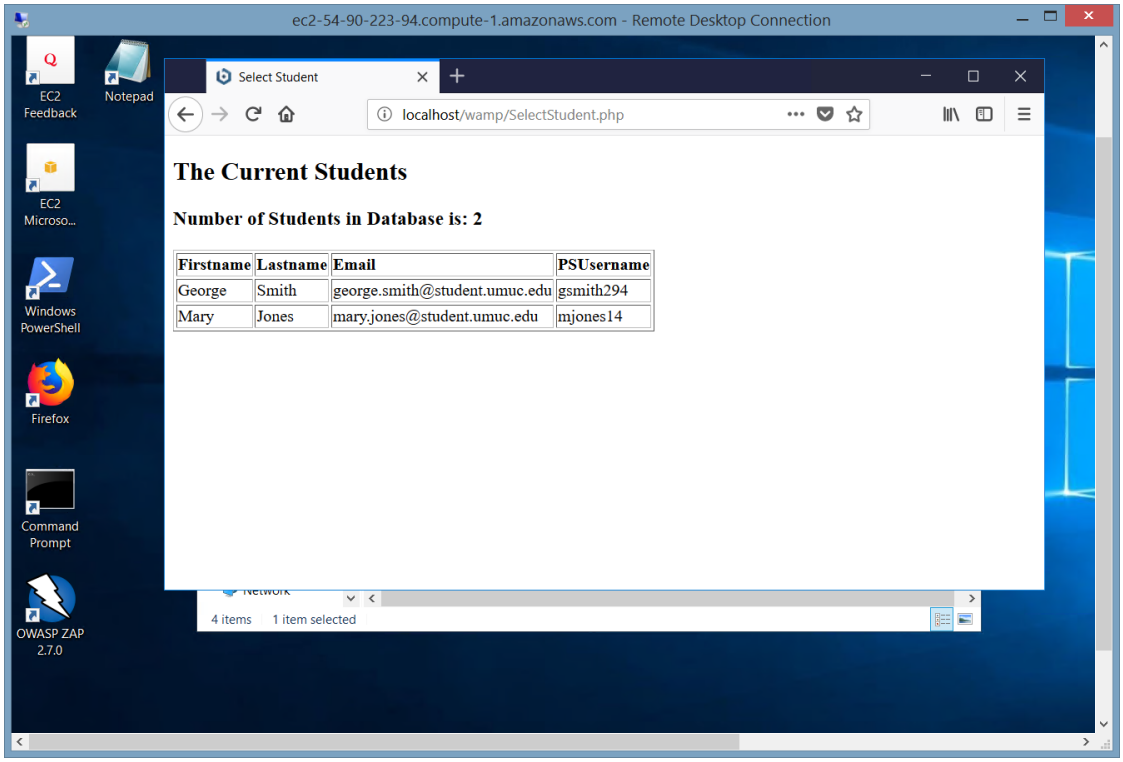

Figure 10 Student Table Contents Display

If you are receiving a database connectivity error message, you probably didn't set-up the database properly. Go back to Create database steps shown on page to verify each of those steps were performed.

7. Next, we will create the PHP code that will provide a form and response for entering data into the database table. **Note, this code has SQL injection issues and should not be used as a model for any programming project.** We will correct the code soon, but it is important to see what incorrect, or unsafe code looks like.

   To make our code more efficient, we made several additions to the other files to allow for the additional SQL functions while still using the existing classes and database connectivity.

   In addition, we have to update the MySQL database permissions to allow the wamp_user to be able to Insert data into the Students table. **Without this permission, the user will not be able to update the table through the Web Interface.**

   The SQL statement granting this permission must be run as the root user at the MySQL command prompt:

```
GRANT INSERT ON wamp.Students TO wamp_user@localhost IDENTIFIED BY
PASSWORD '*E090926B474658359CB983C65B768047DF3A55BE';
```

The following functions were added to the DBQueries.php file:

```
function insertStudent ($student)
  {

            // Connect to the database
   $mysqli = connectdb();

       $firstname = $student->getFirstname();
   $lastname = $student->getLastname();
   $wsname = $student->getPsusername();
   $email = $student->getEmail();

      // Now we can insert
      $Query = "INSERT INTO Students
                (Psusername, firstName,lastName,eMail)
                 VALUES ('$wsname','$firstname', '$lastname', '$email' )";

      $Success=false;
      if ($result = $mysqli->query($Query)) {
        $Success=true;
      }
      $mysqli->close();

            return $Success;
      }

function countStudent ($student)
  {
      // Connect to the database
   $mysqli = connectdb();
```

```
$firstname = $student->getFirstname();
$lastname = $student->getLastname();
$wsname = $student-> getPsusername();
$email = $student->getEmail();


   // Define the Query
   // For Windows MYSQL String is case insensitive
    $Myquery = "SELECT count(*) as count from Students
            where Psusername='$wsname'";

    if ($result = $mysqli->query($Myquery))
    {
       /* Fetch the results of the query */
       while( $row = $result->fetch_assoc() )
       {
           $count=$row["count"];
       }

       /* Destroy the result set and free the memory used for it */
       $result->close();
}

   $mysqli->close();

   return $count;

}
```

The revised files that include the revisions to the DBQueries.php file and the StudentInsert.php are included in an attachment named wamp2.zip. You should copy that file to the SDEV AMI and test the insert functionality.

Figures 11, 12 and 13 show the results of testing the wamp2 zip file on the SDEV AMI. Notice, once a new record is entered in the database, it can be successfully displayed with the “Display Students” option.

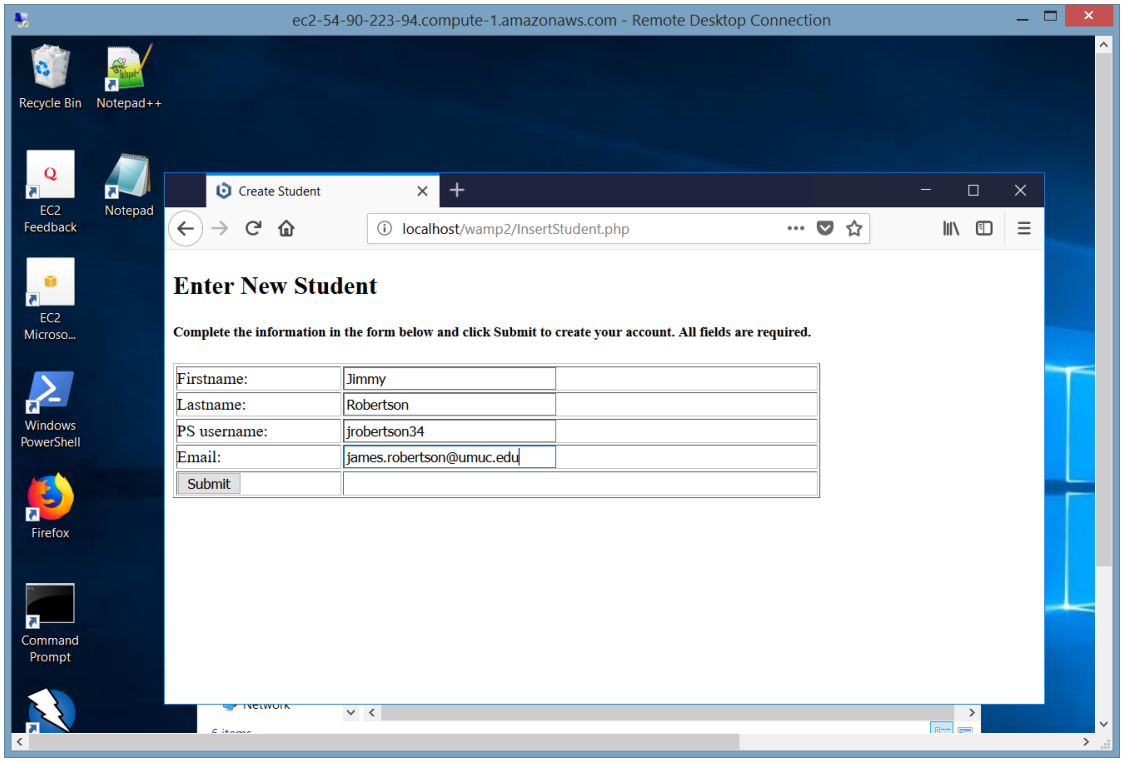

Figure 11 Inserting a new Student record

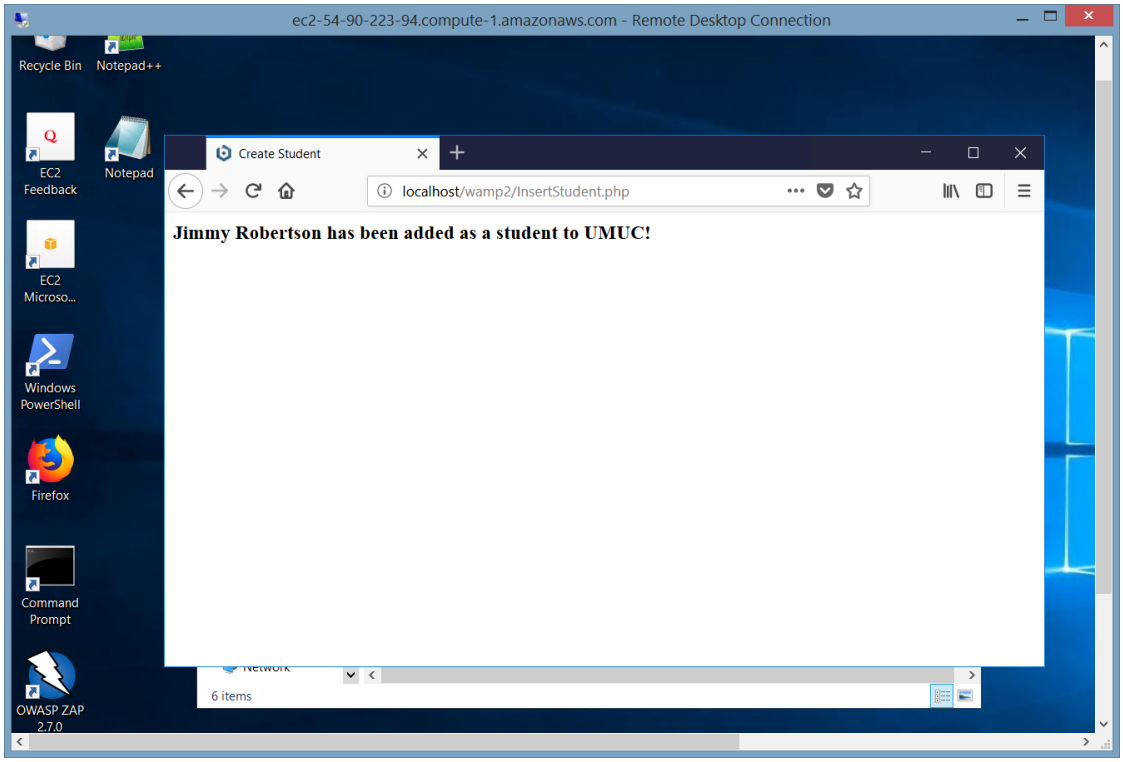

Figure 12 Successful Reply after Adding Record

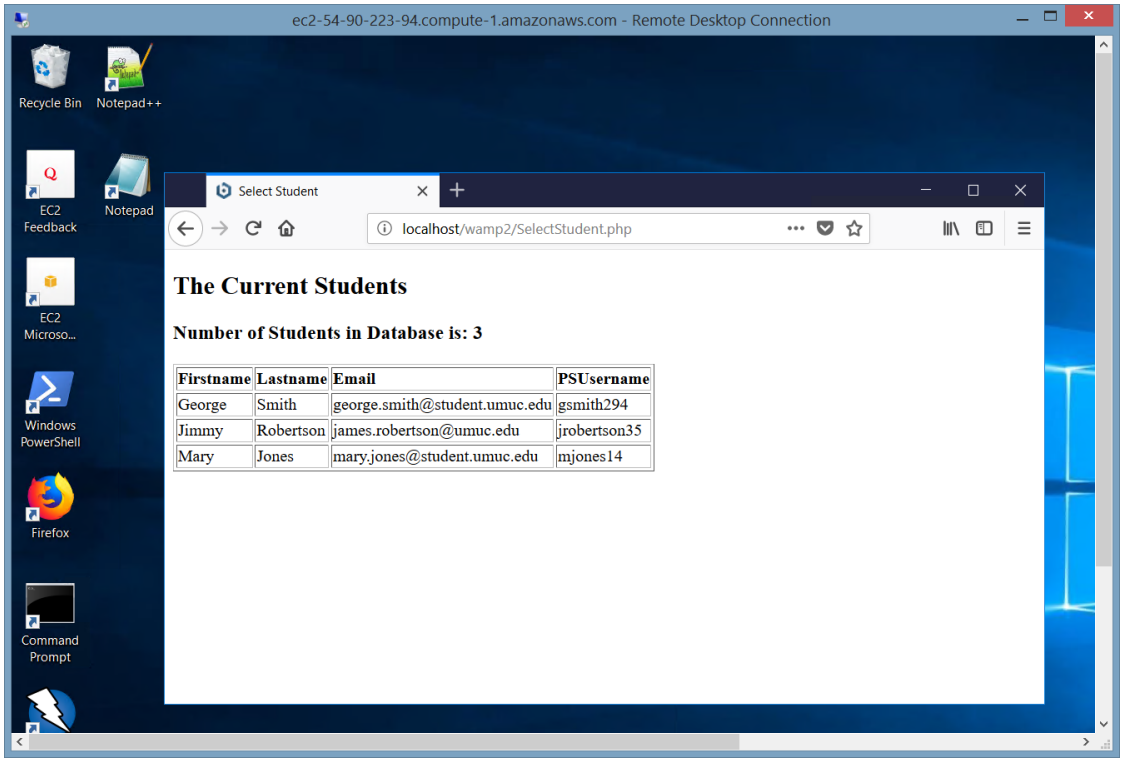

Figure 13 Displaying the new Student Record

Note the following code in the DBQueries.php file assumes you have honest users.

```
$Query = "INSERT INTO Students
                (Psusername, firstName,lastName,eMail)
                 VALUES ('$wsname','$firstname', '$lastname', '$email' )";
```

8. Next, we replace this statement with a prepared statements to help mitigate the SQL injection in the insertStudent function:

```
function insertStudent ($student)
  {

            // Connect to the database
   $mysqli = connectdb();

       $firstname = $student->getFirstname();
   $lastname = $student->getLastname();
   $wsname = $student->getPsusername();
   $email = $student->getEmail();

            // Add Prepared Statement
            $Query = "INSERT INTO Students
                (Psusername,firstName,lastName,eMail)
                 VALUES (?,?,?,?)";
```

```
            $stmt = $mysqli->prepare($Query);

$stmt->bind_param("ssss", $wsname,$firstname,$lastname,$email);$stmt-
>execute();


      $stmt->close();
      $mysqli->close();

            return true;
      }
```

Note the bind statement is using “ssss” representing 4 strings. Other options include i for integer and d for double. We will use the prepared statement in the remaining examples to mitigate SQL Injection. This essentially prevents the system from processing extra characters if a user attempts to inject those into your query.

Note the functionality doesn’t change as a user can still insert new student records and display them as before.

Now that we have a form to Insert and Select data, we can continue to expand and add the delete and update functionality. The attached wamp3.zip contains the new functions for DeleteStudent and UpdateStudent function calls within the DBQueries.php file. Note, the other files are unchanged and further demonstrate the importance of both good and secure design for your code.

9. To prepare for the delete and update functionality, we need to add the privileges to the wamp.Students table. Enter the following command at the MySQL prompt on your SDEV AMI to ensure the proper permissions are grants to the Students table. Be sure to use the root user of the MySQL account when granting this permission.

```
GRANT Delete,Update ON wamp.Students TO wamp_user@localhost IDENTIFIED BY
PASSWORD '*E090926B474658359CB983C65B768047DF3A55BE';
```

10. Once the permissions are set, copy the wamp3.zip folder to your SDEV AMI and unzip and place the contents in a wamp3 folder in the htdocs folder. Be sure the directory structure is correct as previously discussed when you copied the wamp and wamp2.zip folders to the SDEV AMI.

As always, you should review and tinker with all aspects of the code to become comfortable with the functionality. The pattern of use and functionality is the same. Note that the DBQueries.php file make use of prepared statements for all queries. Some additional hyperlinks were also added to the code to easily return to the StudentApp from the response screens.

Envision how you could modify this template for adding additional tables and queries to those tables for an application of your choice.

Figures 14 and 15 shows the results of using the Update functionality on the SDEV AMI.

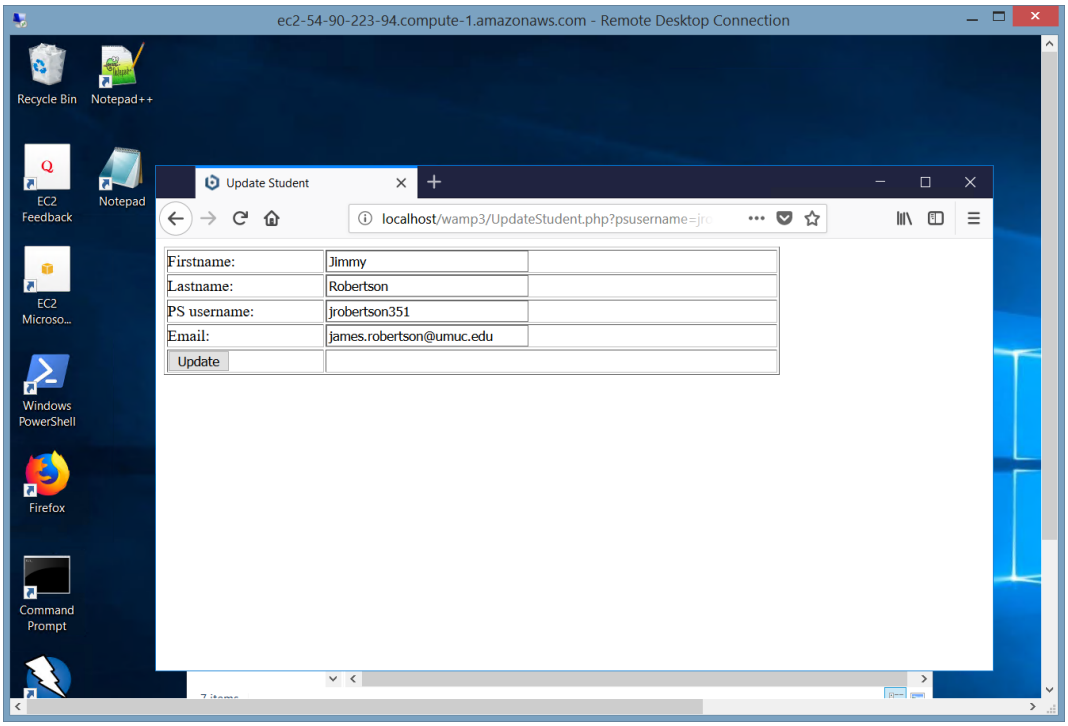

Figure 14 Updating a record

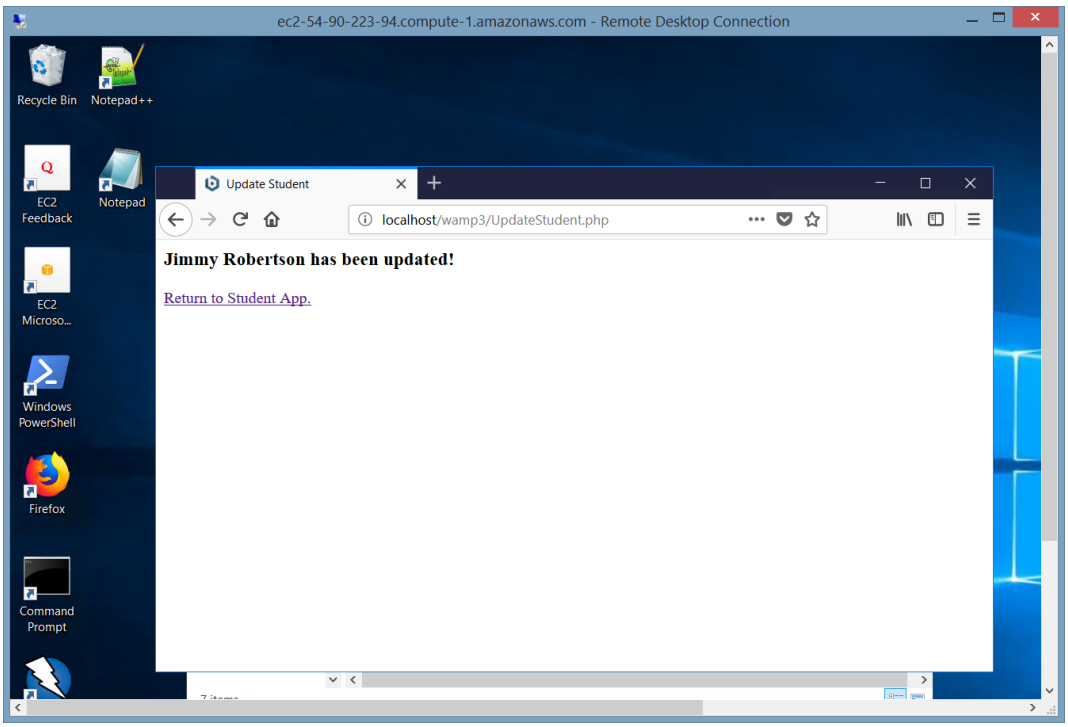

Figure 15 Response after Updating Record

Figures 16 and 17 show the results of using the Delete functionality on the SDEV AMI.

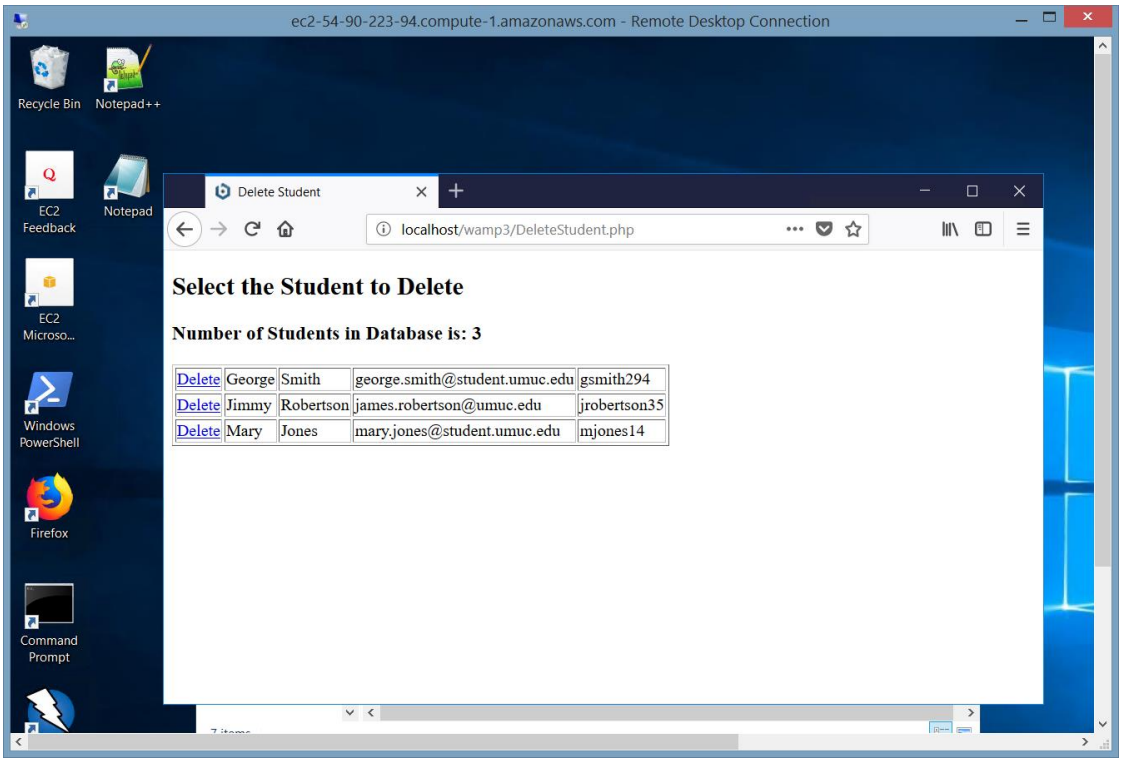

Figure 16 Deleting a record

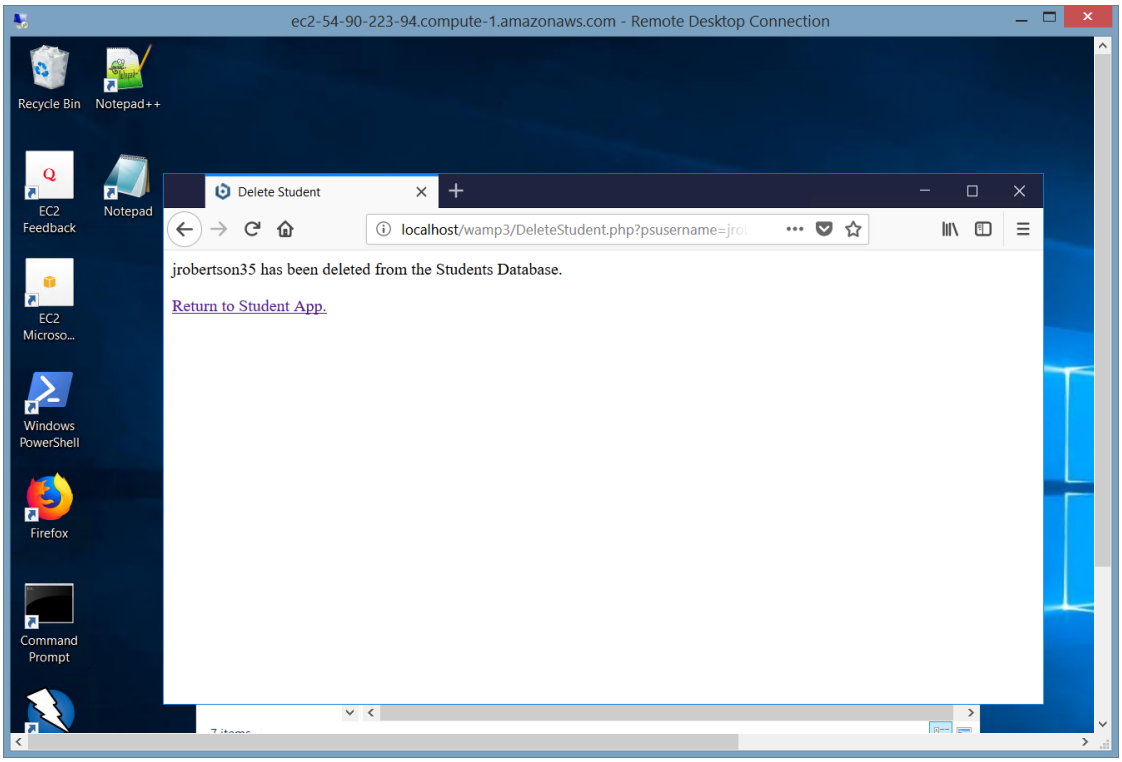

Figure 17 Response after deleting a record

Lab submission details:

For your lab for this week, you will create a WAMP application using the SDEV AMI. The application should satisfy the following requirements:

a. Allows a user to register and login to a Game application
b. When registering the user should provide a unique username, password, and 3 security questions. The password should be at least 8 characters in length and should include at least one upper case letter, one number and one special character. The three security questions should be from one of 10 prepared security questions that you design. For example, have them select from a list of 10 questions and provide their answers to 3 they select.
c. When logging into the Game application, if the username and password are correct, the application should ask one of the 3 security questions. The question should be randomly selected.
d. Access to the Game is granted if both the username and password are correct and the random security question was answered correctly.
e. The game shall be a simple mathematical formula in the form of a question from randomly generated numbers. For example, the game could be as simple as What is 102 * 203? You should select the formula used and the range of random numbers for each variable.
f. A scoring mechanism should be available that stores the high score of each user. You should design a points system for scoring the game.
g. The user with the highest score and their score should be displayed on the application after logging into the system.
h. The application should allow the user to submit the answer and provide feedback as to whether the answer was correct or not.
i. The user can quit the game at any time.
j. The user can logout of the game application at any time.
k. The application should allow the user to update their username, password and security questions at any time.
l. The application should allow the user to delete their account at any time.

Design requirements include the following:

a. Data must be stored and organized in MySQL tables. The required data should be the User data (username, password, security questions and answers), and high scores for each user. The mathematical formulas do not need to be stored in the database. Your database design should be relational with no redundant data in the tables and consist and primary and foreign keys as needed. (Hint: You should not put all of the data in one table)
b. Your design should be secure and at a minimum ensure no SQL injection is possible, passwords are encrypted in the database, passwords are encrypted in parameter tables, and security questions are encrypted in the database and only the least amount of privileges are given to database and web users for them to perform their functions.
c. Take the time to design the flow of your forms, such that they flow logically within your application and are presented in an attractive easy-to-use Web interface.

d. Create screen captures showing the successful running of your application from registering a user to playing the game to updating and deleting user accounts. Each screen capture should be fully described.

For your deliverables, you should submit a winzip file containing your word document (or PDF file) with screen captures and detailed descriptions of the application running successfully along with your SQL script file and all PHP files. (Hint: Use the templates provided to you but change the functionality as required)

Include your full name, class number and section, professor name and date in the document.

Grading Rubric:

Attribute	**Meets**
WAMP App	**80 points** Allows a user to register and login to a Game application. (5 points) When registering the user should provide a unique username, password, and 3 security questions. The password should be at least 8 characters in length and should include at least one upper case letter, one number and one special character. The three security questions should be from one of 10 prepared security questions that you design. (10 points) When logging into the Game application, if the username and password are correct, the application should ask one of the 3 security questions. The question should be randomly selected. (5 points) Access to the Game is granted if both the username and password are correct and the random security question was answered correctly. (5 points) The game shall be a simple mathematical formula in the form of a question from randomly generated numbers. (5 points) A scoring mechanism should be available that stores the high score of each user. (5 points) The user with the highest score and their score should be displayed on the application after logging in to the system. (5 points) The application should allow the user to submit the answer and provide feedback as to whether the answer was correct or not. (10 points) The user can quit the game at any time. (5 points) The user can logout of the game application at any time. (5 points)

	The application should allow the user to update their username, password and security questions at any time. (15 points) The application should allow the user to delete their account at any time. (5 points) **Does not include access.log (-100)** **Does not use the SDEV AMI. (-100)** **Does not use WAMP stack (-100)** **Does not employ required security components (-100)**
Documentation and submission	**20 points** Submits a winzip file containing your word document (or PDF file) with screen captures and detailed descriptions of the application running successfully along with your SQL script file and all PHP files. (15 points) Include your full name, class number and section, professor name and date in the document and Your report is well-organized and clearly written. (5 points)