Lab 09: Working with Databases – PHP and MySQL

Task 1: Create a new database to store test scores

(a) Setup database:

i. Create a new MySQL database named "`lab9`".
ii. Create two new tables named "`students`" and "`scores`" in "`lab9`" database to store the data for students and scores.
iii. Insert some records into "`students`" table.
iv. Create a new user with the username ('`user1`') and password ('`user1abc`'). Grant the user to have privilege to assess the "`lab9`" database.

NOTE: Refer to the `task1.sql` file for the SQL statements to create database, create tables and insert new records into table.

(b) Create a new PHP file named "`config.php`" file and write the PHP codes with **PDO** statements to connect to MySQL database and handle connection errors. The database information is below:

i. Database Host Name = '`localhost`';
ii. Database Name = '`lab9`';
iii. Username = '`user1`';
iv. Password = '`user1abc`';

(c) Modify the HTML form that you have written in **Lab08 – Task 1: "addScores.php"**:

i. Add a new `<input>` element with option values in a `<datalist>` to allow users to select a **Student ID** to enter the test scores.
ii. Include the `config.php` file to connect to the database and handle connection errors.
iii. Write the PHP codes with **PDO** statements to:
   - Retrieve all the Student IDs from the "`students`" table using **SELECT** SQL statement.
   - Write a while loop to print all the retrieved Student IDs as the `<option value>` in the `<datalist>`. Figure 1 shows the sample screenshots for the list of Student IDs in a `<datalist>` in "addScores.php".

```
<!DOCTYPE html>
<html>
<body>
<h1>Enter Scores:</h1>
<form method="POST" action="grader.php" >
      <input list="students">
      <datalist id="students">
      //Write php codes to generate list from 'students'table.
        <option value="StudentID">
      </datalist>
      Score 1 <input type="text" name="score1"/><br/>
      Score 2 <input type="text" name="score2"/><br/>
      Score 3 <input type="text" name="score3"/><br/>
      Score 4 <input type="text" name="score4"/><br/>
      Score 5 <input type="text" name="score5"/><br/>
      Score 6 <input type="text" name="score6"/><br/><br/>
      <input type="submit">
</form>
</body>
```

```
</html>
```

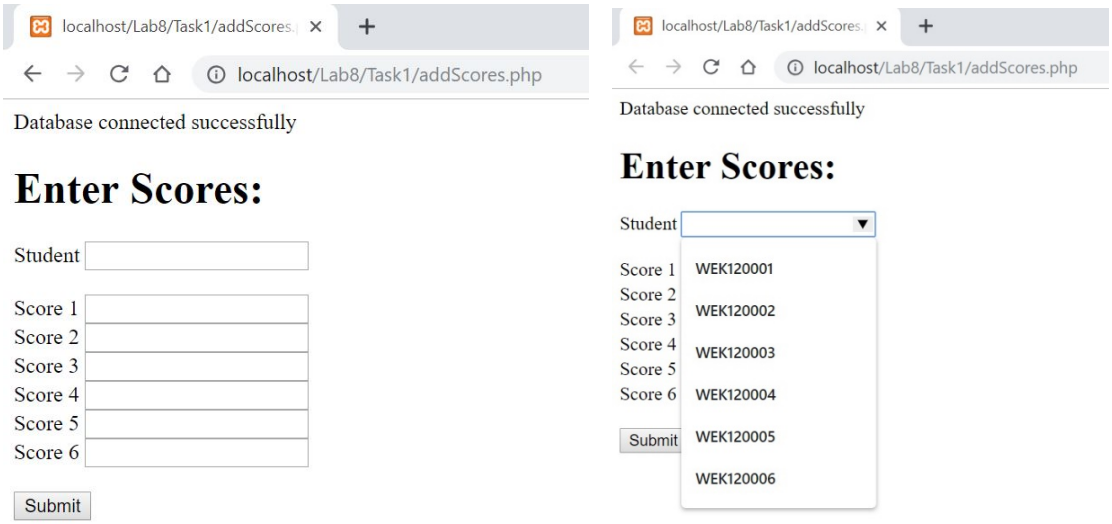


Figure 1: Screenshots of addScores.php

(d) Figure 2 shows the sample screenshot for "grader.php". Modify the PHP program that you have written in Lab08 – Task 1: "grader.php" to do the following tasks:

i. Define a new variable named `$studentID` to store the Student ID sent from the form with the POST method in the "`addScores.php`" page.

ii. Include the `config.php` file to connect to the database and handle connection errors.

iii. Write PHP codes with **PDO statements** to:
   - insert a new record into "`scores`" table using the **INSERT** SQL statement;
   - get ScoreID of last inserted record in the "`scores`" table and define a new variable named `$last_id` to store the id.
   - print a statement to show that a new record was created successfully;
   - free resources and closing database connection;

iv. Provide a link to allow navigation to "View Score" and send the last inserted ScoreID (`$last_id`) to `viewScores.php` using the GET method:
```
<a href='viewScores.php?id=$last_id'>View Scores</a>
```

v. Provide a link to allow navigation to "Add New Score":
```
<a href='addScores.php'>Add New Scores</a>
```

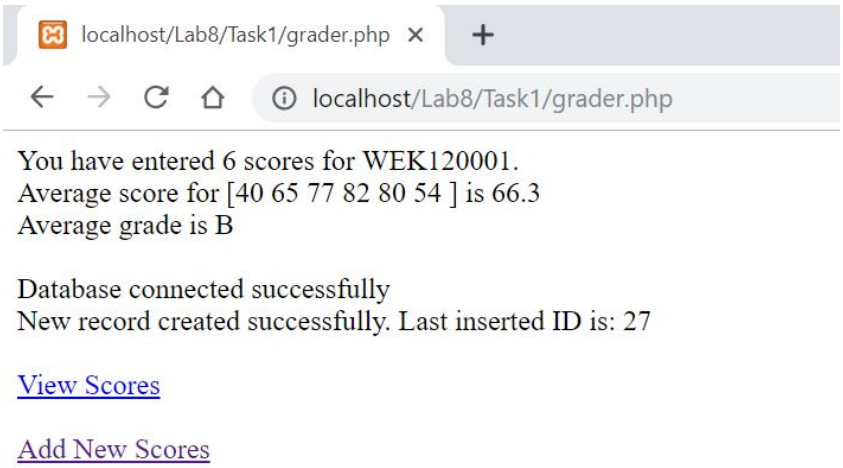


Figure 2: Screenshot of grader.php

(e) Create a new PHP file named "**viewScores.php**" to allow users to **view the scores and choose to update or delete** the record. Figure 3 shows the sample screenshot for "viewScores.php". Write PHP codes to do the following tasks:

i. Include the `config.php` file to connect to the database and handle connection errors.

ii. Define a new variable named `$id` to store the **ScoreID** sent from the form with the POST method in the "addScores.php" page.

i. Write PHP codes with **PDO statements** to:

- Select all the data associated with this particular ScoreID in the "`scores`" table using the **SELECT** SQL statement.
- Define variables to store all the data retrieved from the database.
- Create an html form with the following parameters:
  `<form method="post" action="editScores.php">` and inputs:
  - 6 text inputs for the 6 scores,
  - 2 hidden inputs for ScoreID and StudentID.
  - 2 submit buttons: **Update** and **Delete**,
  `<input type="submit" name="update" value="Update">`
  `<input type="submit" name="delete" value="Delete">`
- free resources and closing database connection.

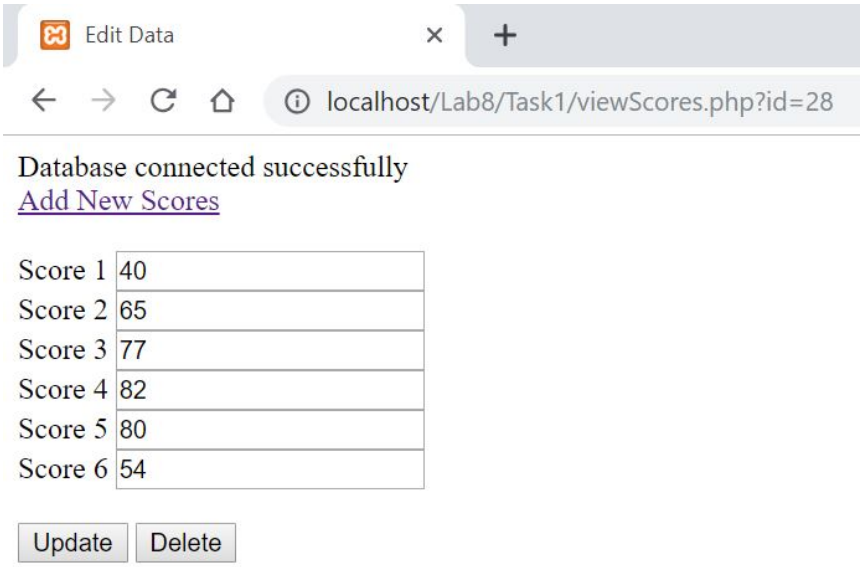


Figure 3: Screenshot of viewScores.php

(f) Create a new file named "**editScores.php**" to handle either the update or delete request from users. Write PHP codes with if-else statements to handle the two requests:

i. **Update** request `if(isset($_POST['update']))`: Write PHP codes with **PDO statements** to get the ScoreID, StudentID and updated score values sent from the `viewScores.php` form and update the record in the database using the **UPDATE** SQL statement. Print a message to show whether the update is successful or failed.

Figure 4 shows the sample screenshot for "editScores.php" that handle the update request when the user click the **Update** button.

ii. **Delete Request** else `if(isset($_POST['delete']))`: Write PHP codes with **PDO statements** to get the ScoreID sent from the viewScores.php form and delete the record in the database using the **DELETE** SQL statement. Print a message to show whether the update is successful or failed.

Figure 5 shows the sample screenshot for "editScores.php" that handle the delete request when the user clicks the **Delete** button.

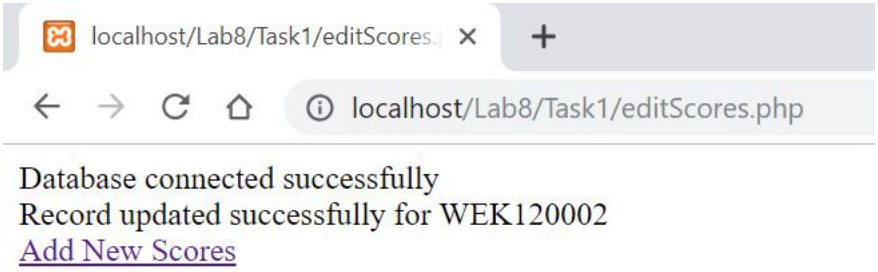


Figure 4: Screenshot of editScores.php (Handle Update request)

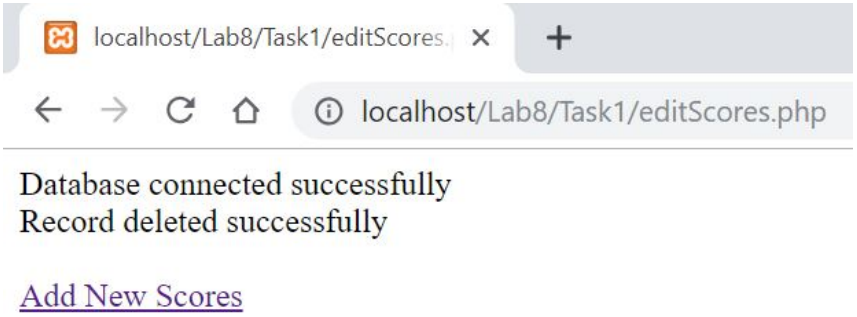


Figure 5: Screenshot of editScores.php (Handle Delete request)

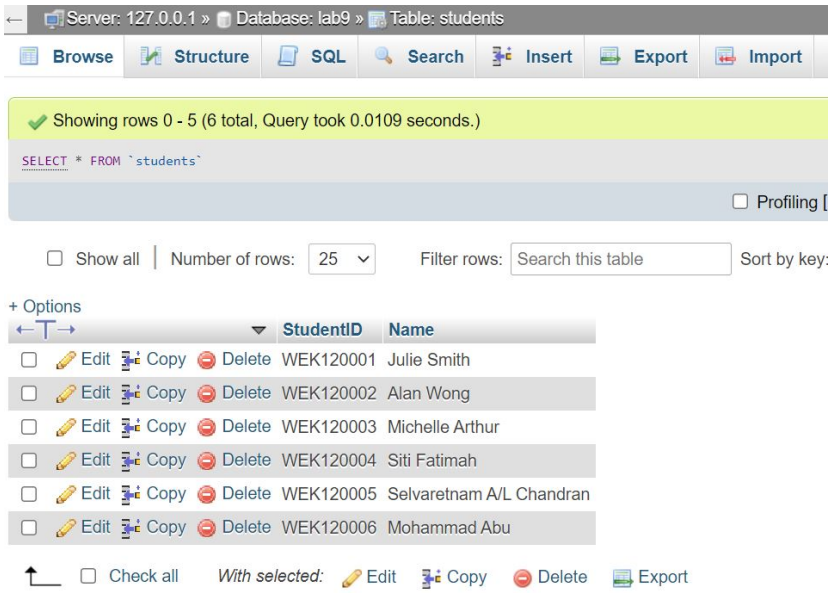
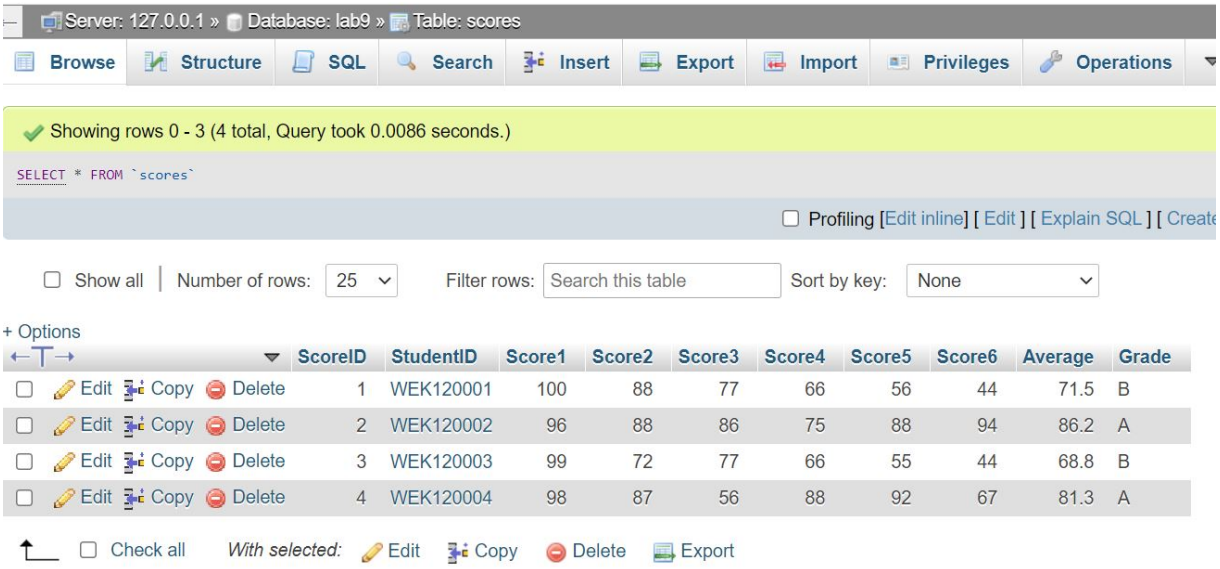


Figure 6: Screenshot of '`lab9`' database – '`students`' table



ScoreID	StudentID	Score1	Score2	Score3	Score4	Score5	Score6	Average	Grade
1	WEK120001	100	88	77	66	56	44	71.5	B
2	WEK120002	96	88	86	75	88	94	86.2	A
3	WEK120003	99	72	77	66	55	44	68.8	B
4	WEK120004	98	87	56	88	92	67	81.3	A

Figure 7: Screenshot of '`lab9`' database – '`scores`' table

References:

https://www.w3schools.com/tags/tag_datalist.asp

https://www.w3schools.com/php/php_mysql_insert_lastid.asp

Task 2: Using PHP in JavaJam Coffee House website:

NOTE: Create new folder and files

- Create a new folder on your "C:\xampp\htdocs" folder called "**javajam7**". Copy **all** the files from your Lab 8's folder (javajam6) into the "**javajam7"** folder.
- Create a new PHP file named "**config.php**".

a) Setup database: refer to the `javajam.sql` file to create new database and table
   - Create a new MySQL database named "`javajam`".
   - Create a new table named "`members`" in the "`javajam`" database to store the new member information.

b) Open the "**config.php**" file and write the PHP code using **MySQLi statements** to connect to database and handle connection errors. The database information is below:
   - Database Host Name = '`localhost`';
   - Database Name = '`javajam`';
   - Username = '`user`';
   - Password = '`userpwd`';

When the user fills out the form in "signup.html" and clicks the submit button, the form data is sent for processing to a PHP file named "processSignup.php". The form data is sent with the HTTP POST method.

c) In the "**processSignup.php**" file, modify PHP codes to:
   i. **Remove the codes that** (a) using the PHP File functions (e.g. fopen, fwrite, fclose) and (b) redirect to Profile page. In the new program, new member records will be added to the "javajam" database and login is required to go to Profile page.
   ii. Include the database connection file: `include_once("config.php");`
   iii. **In Lab 8, variables were defined to store the input data** (Name, email, password), sent from the previous page, **Sign Up** form using the POST method. In this lab, write **PHP codes** with **MySQLi statements** to add new member record into "`javajam`" database. Validate the new registration account using email address (check this in **the "**`javajam`" **database**, "`members`" **table** using the **SELECT** SQL statement and **WHERE** clause):
      - **If the email already exists in the** "`members`" **table → registration failed, write codes to:** (1) Print a message to show that the account already exists (e.g. Email already exists). Provide links to allow navigation to "Sign Up" and "Log In" pages.
      - **Else if it is a new member → registration successful, write codes to:**
        (1) Insert a new record into **the "**`javajam`" **database**, "`members`" **table** using the **INSERT** SQL statement. Write the codes using Prepared Statements and bound parameters in MySQLi.
        (2) Print a message to show that the registration is successful. Provide a link to allow navigation to "`Log In`" page.
      - **Free** resources and **closing** database connection.
   iv. Run your web application and test whether the new record is inserted into "`members`" table successfully.

Figures 8 to 11 show the screenshots of the “signup.html”, “processSignup.php” (registration successful and failed) and database.

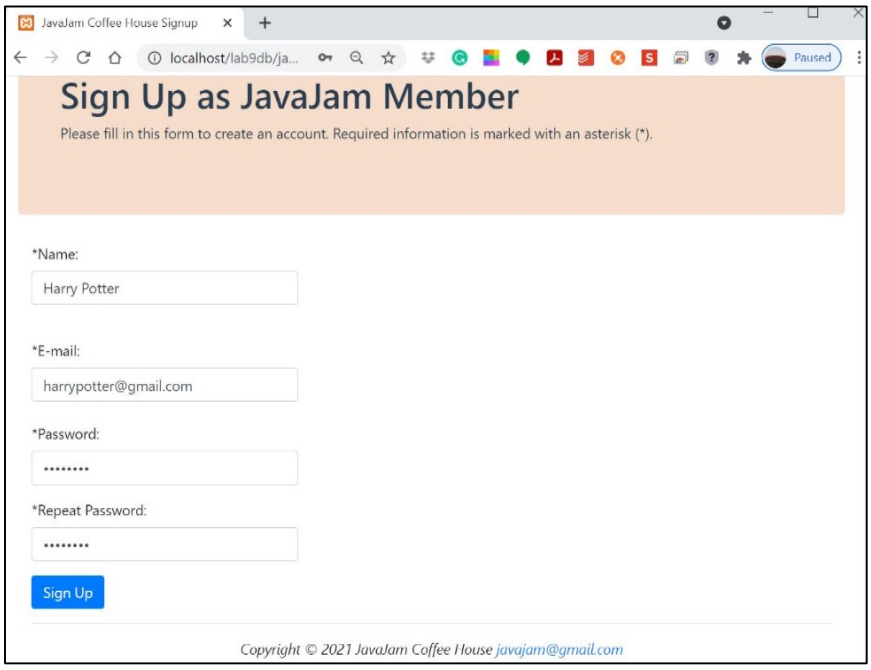


Figure 8: JavaJam Coffee House’s Sign Up page

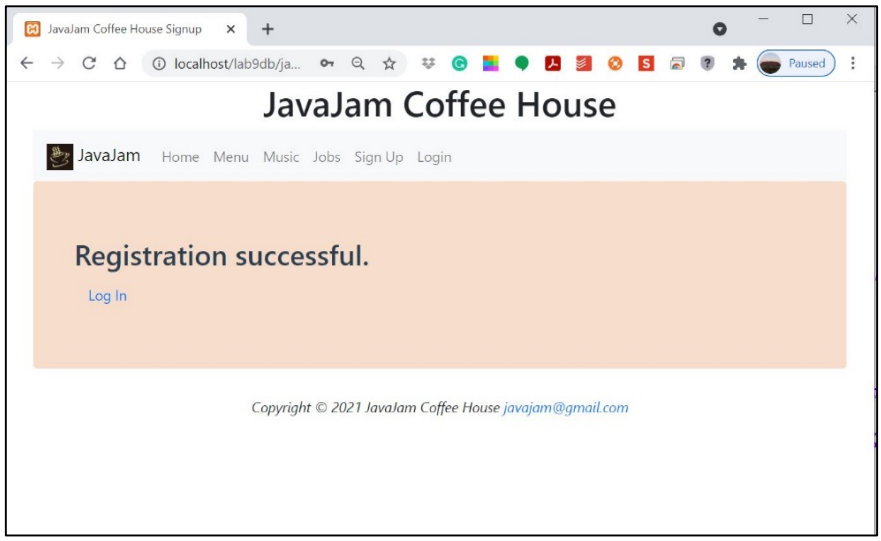


Figure 9: JavaJam Coffee House’s processSignup.php (Registration successful)

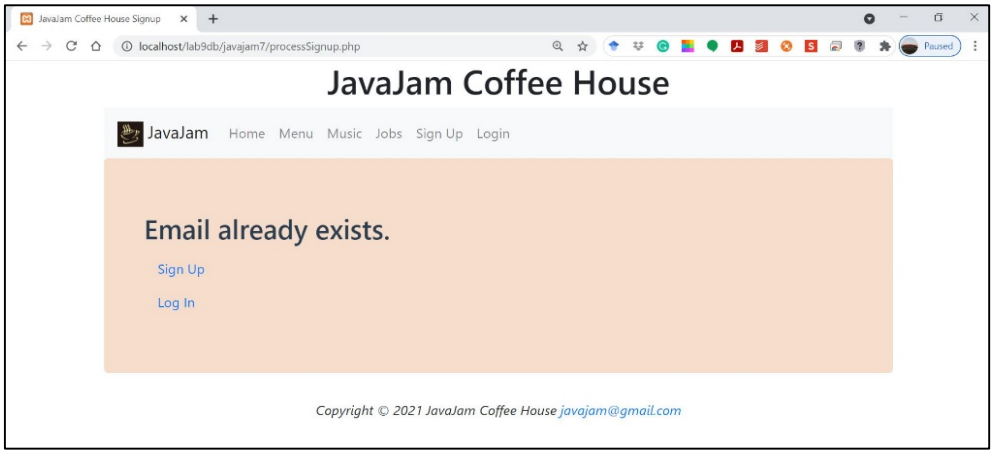


Figure 10: JavaJam Coffee House's processSignup.php (Registration failed – email already exists)

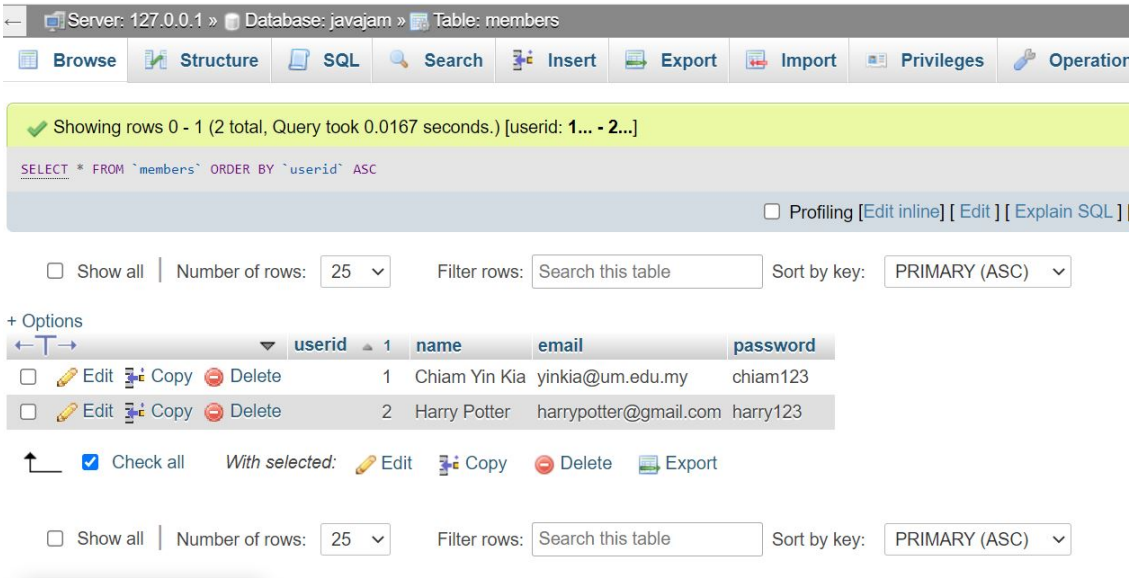


Figure 11: New record added to into "`members`" table in "`javajam`" database successfully (screenshot from phpdMyAdmin)