Name: Ngoc Phuong Trang Duọng
ID: 223990468

Task 9.2D

This assignment requires me to develop a PHP script that checks if the employee name exists in the database before adding a new one. If the name is duplicated, the system will display a message "Name already exists" and stop the data adding process. If the name does not exist, the system will continue inserting the data into the database.

Before I start, I need to understand the assignment requirement. The assignment requires me to check if the employee name is duplicated in the database. If yes, the system will give an error message "Name already exists". If not, the data will be added to the database.

The first step in performing the task is to create a MySQL database. I created a database with a table named employee, which has fields employeeName, street, and city. The employeeName field is the primary key and is the field I need to check when data is entered.

After that, to solve the problem of checking if an employee's name is already in the database, I followed these steps:

1. Check the Data Submission Method (POST):

First, I used the if ($_SERVER["REQUEST_METHOD"] == "POST") statement to check if the form was submitted via the POST method. This ensures that the code only runs when the user actually submits the form.

2. Retrieve Data from the Form:

I used $_POST to get the values the user entered into the form, including employeeName (employee name), street (address), and city (city).

3. Connect to the Database:

To connect to the database, I used the mysqli object in PHP. The connection parameters include the server address, username, password, and the database name. After successfully connecting, I check for any connection errors.

4. Check for Duplicate Employee Name:

I used an SQL SELECT statement to check if the employee name already exists in the database. To avoid SQL injection, I used prepared statements with the bind_param method to protect user input.
If the name exists in the database (i.e., there is at least one record), I throw an exception (Exception) with the message “The name already exists”.

5. Insert New Data If Name Does Not Exist:

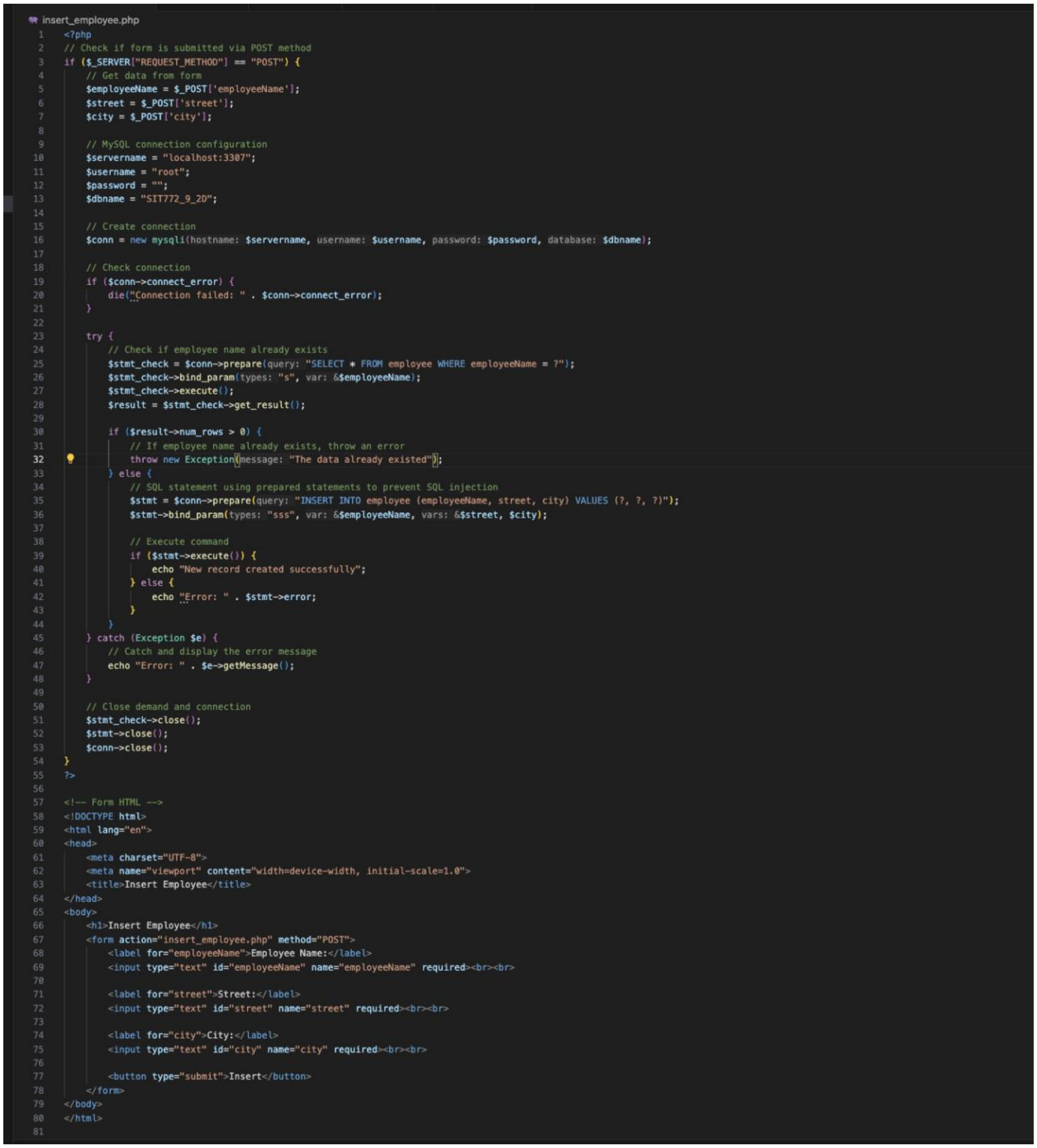
If the employee name does not exist, I proceed to execute the SQL INSERT statement to add the employee to the database. I also used prepared statements to protect the data from SQL injection. After executing the INSERT statement, I check if the operation was successful. If successful, I display the message "New record created successfully." If there’s an error, I display the error message.

6. Error Handling:

In the case where the employee name already exists or if any error occurs during execution, I used try-catch to catch the error and display a suitable error message to the user.

7. Close the Connection:

Finally, after all operations are complete, I close the database connection objects to clean up resources and prepare for the next request.

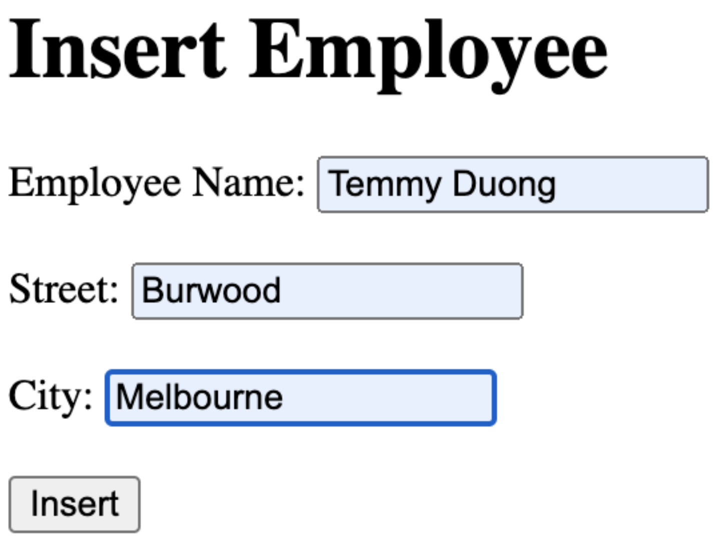


insert_employee.php

```php
<?php
// Check if form is submitted via POST method
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Get data from form
    $employeeName = $_POST['employeeName'];
    $street = $_POST['street'];
    $city = $_POST['city'];

    // MySQL connection configuration
    $servername = "localhost:3307";
    $username = "root";
    $password = "";
    $dbname = "SIT772_9_2D";

    // Create connection
    $conn = new mysqli(hostname: $servername, username: $username, password: $password, database: $dbname);

    // Check connection
    if ($conn->connect_error) {
        die("Connection failed: " . $conn->connect_error);
    }

    try {
        // Check if employee name already exists
        $stmt_check = $conn->prepare(query: "SELECT * FROM employee WHERE employeeName = ?");
        $stmt_check->bind_param(types: "s", var: &$employeeName);
        $stmt_check->execute();
        $result = $stmt_check->get_result();

        if ($result->num_rows > 0) {
            // If employee name already exists, throw an error
            throw new Exception(message: "The data already existed");
        } else {
            // SQL statement using prepared statements to prevent SQL injection
            $stmt = $conn->prepare(query: "INSERT INTO employee (employeeName, street, city) VALUES (?, ?, ?)");
            $stmt->bind_param(types: "sss", var: &$employeeName, vars: &$street, $city);

            // Execute command
            if ($stmt->execute()) {
                echo "New record created successfully";
            } else {
                echo "Error: " . $stmt->error;
            }
        }
    } catch (Exception $e) {
        // Catch and display the error message
        echo "Error: " . $e->getMessage();
    }

    // Close demand and connection
    $stmt_check->close();
    $stmt->close();
    $conn->close();
}
?>

<!-- Form HTML -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Insert Employee</title>
</head>
<body>
    <h1>Insert Employee</h1>
    <form action="insert_employee.php" method="POST">
        <label for="employeeName">Employee Name:</label>
        <input type="text" id="employeeName" name="employeeName" required><br><br>

        <label for="street">Street:</label>
        <input type="text" id="street" name="street" required><br><br>

        <label for="city">City:</label>
        <input type="text" id="city" name="city" required><br><br>

        <button type="submit">Insert</button>
    </form>
</body>
</html>
```

Result:

With then name “Temmy Duong”, street “Burwood”, city “Melbourne”

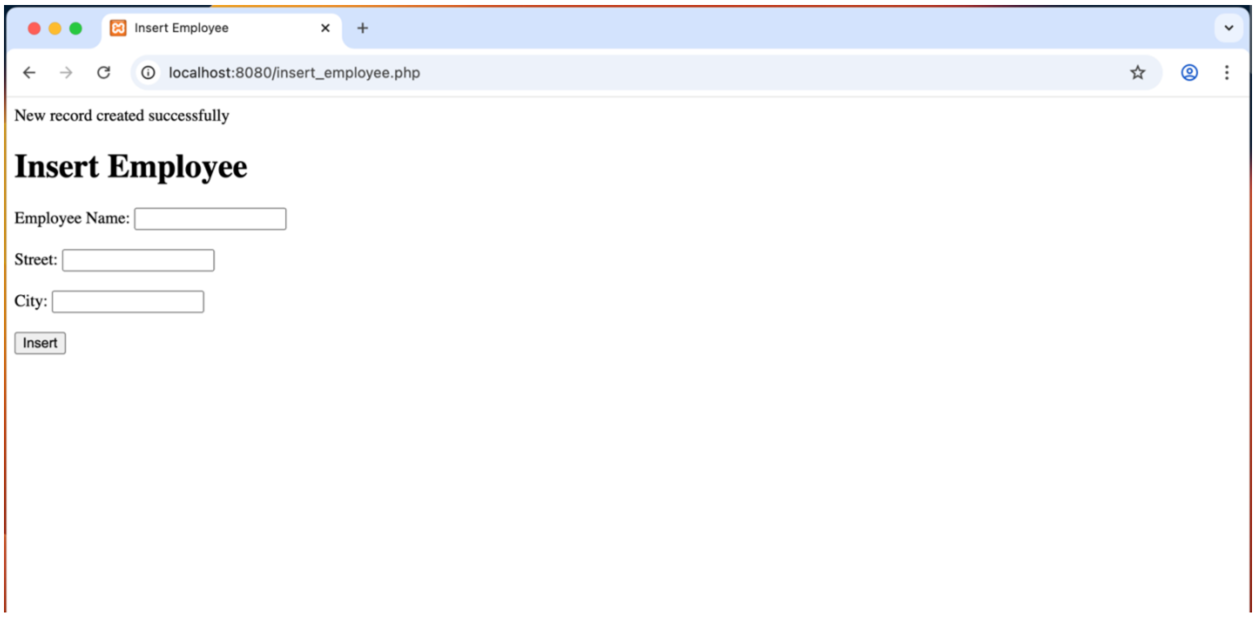


Insert Employee

Employee Name: Temmy Duong

Street: Burwood

City: Melbourne



New record created successfully

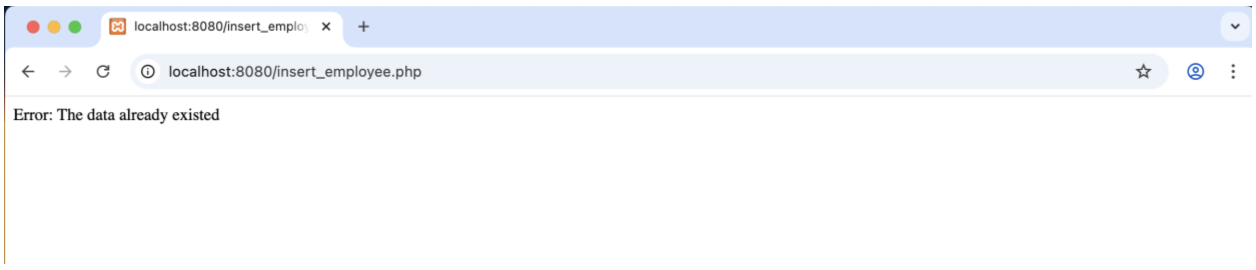
Insert Employee

Employee Name:

Street:

City:

And then, I inserted the name, street and city, the result is:


localhost:8080/insert_emplo
localhost:8080/insert_employee.php

Error: The data already existed

Through this task, I learned how to check if an employee's name already exists in the database to avoid duplicates before inserting new data. I also used prepared statements to protect the application from SQL injection, and learned how to handle errors with try-catch to provide clearer error messages to users. I reinforced my skills in managing database connections, from connecting to the database to closing the connection after the task is completed. This task made me more aware of the importance of securing user data and improving user experience through detailed error messages. Finally, I also strengthened my ability to handle data from HTML forms and use it in SQL queries.