**Name: Aarya Tiwari Roll No.: 16010421119 Batch: B2**

**Experiment 5 Code:**

<?php

$username = "";

$email = "";

$message = "Choose an operation to perform.";

*// Database configuration*

$host = "localhost"; *// Change to your database host*

*// $dbname = "newphp"; // Change to your database name*

$usernameDB = "root"; *// Change to your database username*

$passwordDB = "root"; *// Change to your database password try* {

*// Create a PDO instance*

$pdo = new PDO("mysql:host=$host", $usernameDB,

$passwordDB);

*// $pdo = new PDO("mysql:host=$host;dbname=$dbname;charset=utf8", $usernameDB,*

*$passwordDB);*

$pdo->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

$databaseName = "exp5DB";

$createDatabaseSQL = "CREATE DATABASE IF NOT EXISTS

$databaseName";

*// Execute the SQL query*

$pdo->exec($createDatabaseSQL);

$pdo->exec("USE $databaseName");

$createTableSQL = "CREATE TABLE IF NOT EXISTS users ( id INT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(255) NOT NULL, email VARCHAR(255) NOT NULL

)";

$pdo->exec($createTableSQL);

*if* ($\_SERVER["REQUEST\_METHOD"] == "POST") {

*// Check which form action is being performed (insert, update, or delete)*

*if* (isset($\_POST["action"])) {

*if* ($\_POST["action"] == "insert") {

*// Handle insertion*

$username = $\_POST["username"];

$email = $\_POST["email"];

$id = $\_POST["userid"];

$insertSQL="";

*// Perform the insertion into the "users" table if* (empty($username) || empty($email)) {

$message = "Enter Username or Email to

insert"; empty($id)) {

} *elseif*(!empty($username) && !empty($email) &&

$insertSQL = "INSERT INTO users (username,

email) VALUES (:username, :email)";

} *else* {

$insertSQL = "INSERT INTO users (id, username, email) VALUES (:id, :username, :email)";

}

*if* (!empty($insertSQL)) {

$stmt = $pdo->prepare($insertSQL);

*if* (!empty($id)) {

$stmt->bindParam(':id', $id);

}

*if* (!empty($email)) {

$stmt->bindParam(':email', $email);

}

*if* (!empty($username)) {

$username);

$stmt->bindParam(':username',

}

$stmt->execute();

$message = "Data inserted successfully!";

}

} *elseif* ($\_POST["action"] == "update") {

*// Handle update*

$id = $\_POST["userid"];

$username = $\_POST["username"];

$email = $\_POST["email"];

$updateSQL="";

*if* (!empty($id) && !empty($username) &&

!empty($email)) {

$updateSQL = "UPDATE users SET email =

:email,username = :username WHERE id = :id";

} *elseif*(!empty($id) && !empty($username) && empty($email)) {

$updateSQL = "UPDATE users SET username =

:username WHERE id = :id";

} *elseif*(!empty($id) && empty($username) &&

!empty($email)) {

$updateSQL = "UPDATE users SET email =

:email WHERE id = :id";

} *elseif*(!empty($id) && empty($username) && empty($email)) {

update";

$message = "Enter Username or Email to

} *elseif*(empty($id) && !empty($username) &&

!empty($email)) {

$updateSQL = "UPDATE users SET email =

:email WHERE username = :username";

} *elseif*(empty($id) && empty($username) &&

!empty($email)) {

$message = "Enter Username or ID to update";

} *else* {

$message = "Enter Email to update";

}

*// Perform the update in the "users" table if* ($updateSQL !== "") {

$stmt = $pdo->prepare($updateSQL);

*if* (!empty($id)) {

$stmt->bindParam(':id', $id);

}

*if* (!empty($email)) {

$stmt->bindParam(':email', $email);

$username);

}

*if* (!empty($username)) {

$stmt->bindParam(':username',

}

$stmt->execute();

$message = "Data updated successfully!";

}

} *elseif* ($\_POST["action"] == "delete") {

*// Handle deletion*

$username = $\_POST["username"];

$id = $\_POST["userid"];

$deleteSQL="";

*// Perform the deletion from the "users" table if* (!empty($username) && !empty($id) ||

empty($username) && !empty($id)) {

$deleteSQL = "DELETE FROM users WHERE id =

:id";

} *elseif*(!empty($username) && empty($id)) {

$deleteSQL = "DELETE FROM users WHERE username = :username";

} *else* {

$message = "Enter Username or ID to delete";

}

*if* ($deleteSQL !== "") {

$stmt = $pdo->prepare($deleteSQL);

*if* (!empty($id)) {

$stmt->bindParam(':id', $id);

$username);

}

*if* (!empty($username) && empty($id)) {

$stmt->bindParam(':username',

}

$stmt->execute();

$message = "Data deleted successfully!";

}

}

}

}

} *catch* (PDOException $e) {

echo "Error: " . $e->getMessage();

}

?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<style>

*@import* url('https://fonts.googleapis.com/css2?family=Noto+Sans:ital,wgh t@0,700;1,300&family=Poppins:wght@300;400;600&display=swap');

\* {

margin: 0;

padding: 0;

*/\* font-family: 'Noto Sans', sans-serif; \*/*

font-family: 'Poppins', sans-serif;

}

.container {

height: 100vh; display: flex; flex-wrap: wrap;

}

.quadrant {

flex: 1;

min-width: 50%;

min-height: 50%;

box-sizing: border-box; border: 1px solid #ccc; overflow:hidden; position: relative;

}

.options {

position: absolute;

top: 0;

left: 0;

width: 100%; padding: 10px;

box-sizing: border-box; background-color: #fff; text-align: center; font-size: 1.5rem; display:flex;

justify-content: center; gap: 10px;

align-items: center;

}

.updateTablebtn { position: absolute; bottom: 0;

left: 0;

width: 100%; padding: 10px;

box-sizing: border-box;

*/\* background-color: #fff; \*/*

text-align: center;

}

.updateTablebtn button { padding: 10px; cursor: pointer; border-radius:10px; font-size: 1.2rem;

}

.content {

position: absolute; top: 50%;

left: 50%;

transform: translate(-50%, -50%);

width: 80%; padding: 10px;

box-sizing: border-box; background-color: #fff; text-align: left;

font-size: 1.2rem;

}

.content input { width: 100%; padding: 6px;

box-sizing: border-box; margin-bottom: 6px;

}

table {

*/\* top: 0;*

*left: 0; \*/*

width: 100%; padding: 10px;

*/\* width: 100%; \*/*

border-collapse: collapse;

*/\* margin-top:5px; \*/*

margin-right:20px

}

table, th, td {

border: 1px solid #ddd;

}

th, td {

padding: 10px; text-align: left; font-size: 1.5rem;

}

thead {

background-color: #f2f2f2;

}

tbody tr:nth-child(even) { background-color: #f2f2f2;

}

tbody tr:hover {

background-color: #ddd;

}

</style>

<body>

<div class="container">

<div class="quadrant quad3">

<div class="options" >

<input type="radio" name="operation" id="update"

value="update">

value="insert">

value="delete">

<label for="update">Update</label>

<input type="radio" name="operation" id="insert"

<label for="insert">Insert</label>

<input type="radio" name="operation" id="delete"

<label for="delete">Delete</label>

<div id="dbtype"></div>

</div>

<div class="content">

<?php *if* (!empty($message)) : ?>

<?php echo $message; ?>

<?php *endif*; ?>

</div>

<div class="updateTablebtn">

<button id="showUpdatedTable" >Show Updated

Table</button>

</div>

</div>

<div class="quadrant quad4">

<table>

<thead>

<tr>

<th>ID</th>

<th>Username</th>

<th>Email</th>

</tr>

</thead>

<tbody>

<?php

$selectSQL = "SELECT \* FROM users";

$stmt = $pdo->prepare($selectSQL);

$stmt->execute();

$users = $stmt->fetchAll();

*foreach* ($users as $user) { echo "<tr>";

echo "<td>" . $user["id"] . "</td>"; echo "<td>" . $user["username"] .

"</td>";

echo "<td>" . $user["email"] . "</td>"; echo "</tr>";

}

?>

</tbody>

</table>

</div>

</div>

</body>

<script>

const radioButtons = document.querySelectorAll('input[name="operation"]');

const contentDiv = document.querySelector('.content');

const insertContent = ` method="post">

<form action="exp5.php"

<label

for="userid">User-Id</label>

<input type="text" name="userid"

id="userid" placeholder="Enter Userid">

<label

for="username">Username</label>

<input type="text"

name="username" id="username" placeholder="Enter Username">

<label for="email">Email</label>

<input type="email" name="email" id="email" placeholder="Enter Email">

<input type="submit"

value="Insert">

name="action" value="insert">

<input type="hidden"

`; const deleteContent = `

method="post">

</form>

<form action="exp5.php"

<label

for="userid">User-Id</label>

<input type="text" name="userid" id="userid" placeholder="Enter Userid">

<label

for="username">Username</label>

<input type="text"

name="username" id="username" placeholder="Enter Username">

<input type="submit"

value="Delete">

name="action" value="delete">

<input type="hidden"

`; const updateContent = `

method="post">

</form>

<form action="exp5.php"

<label

for="userid">User-Id</label>

<input type="text" name="userid"

id="userid" placeholder="Enter Userid">

<label

for="username">Username</label>

<input type="text"

name="username" id="username" placeholder="Enter Username">

<label for="email">Email</label>

<input type="email" name="email" id="email" placeholder="Enter Email">

<input type="submit"

value="Update">

name="action" value="update">

<input type="hidden"

</form>

`;

radioButtons.forEach(*radioButton* => { radioButton.addEventListener('change', function () {

*if* (this.checked) {

const selectedValue = this.value;

*if* (selectedValue === 'insert') { contentDiv.innerHTML = insertContent ;

} *else if* (selectedValue === 'delete' ) { contentDiv.innerHTML = deleteContent ;

} *else if* (selectedValue === 'update') { contentDiv.innerHTML = updateContent ;

}

}

});

});

showUpdatedTable.addEventListener("click", function () { fetch('refreshexp5.php')

.then(*response* => response.text())

.then(*data* => {

document.querySelector("table tbody").innerHTML

= data;

})

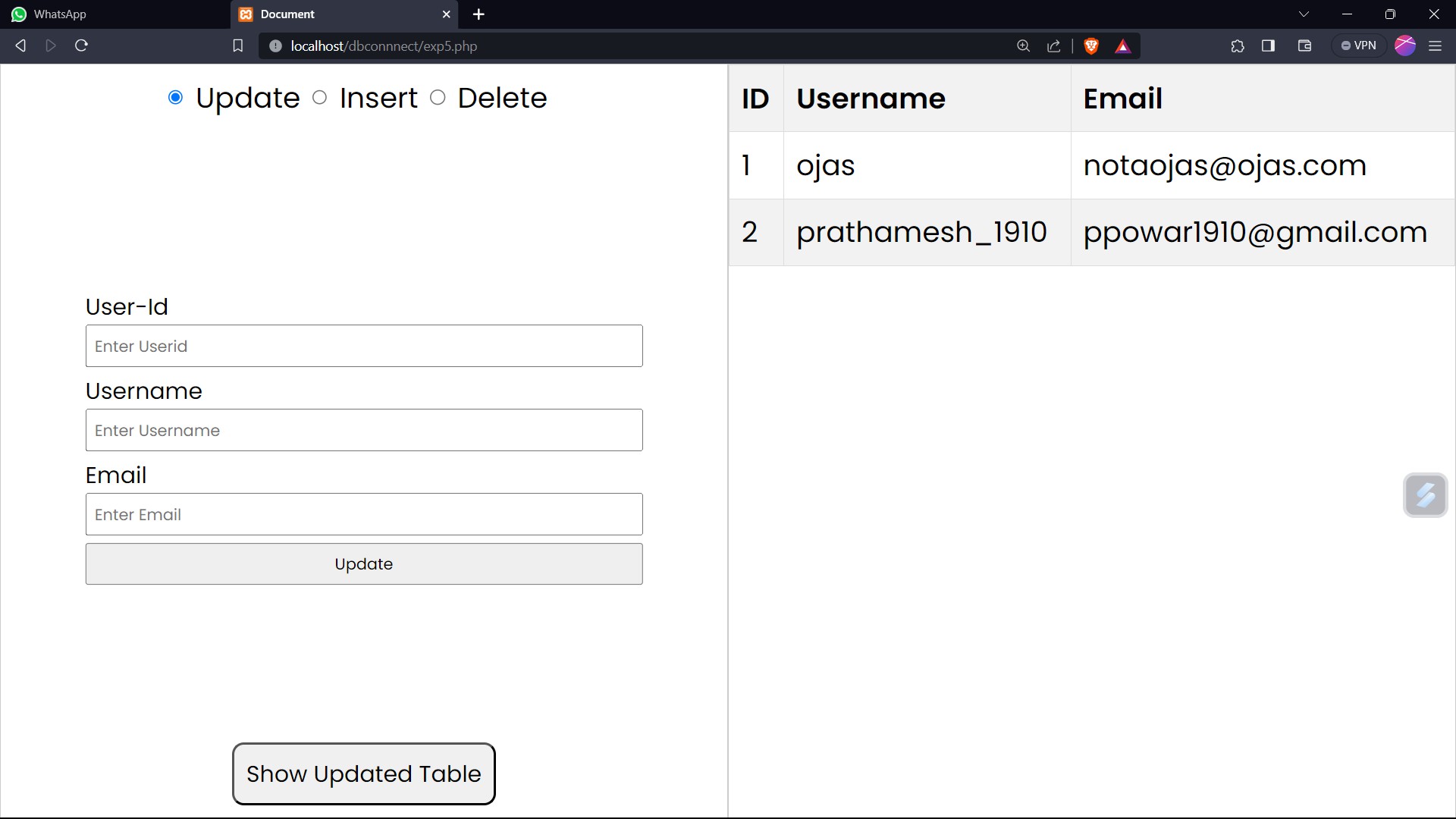
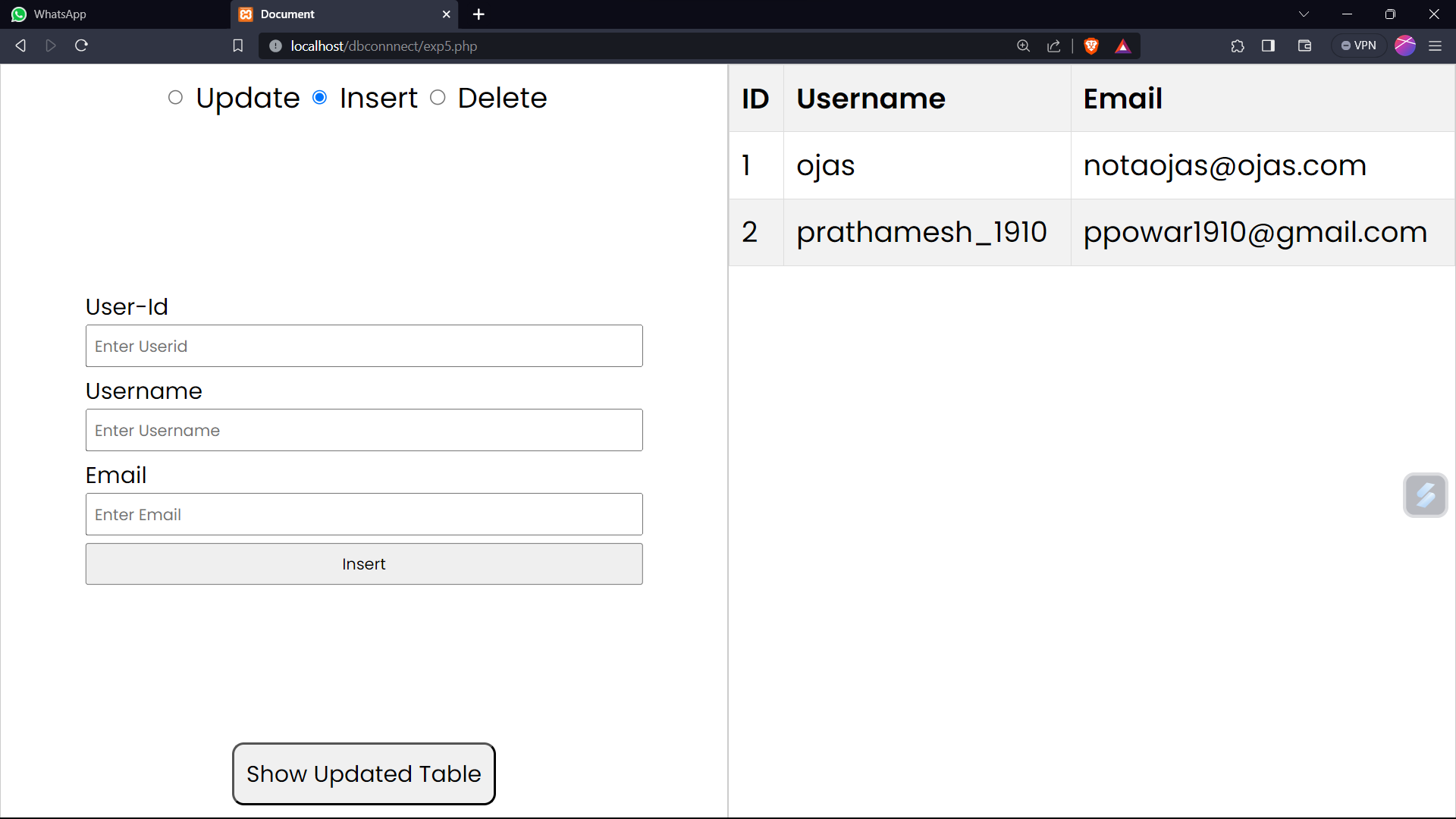
.catch(*error* => console.error(error));

});

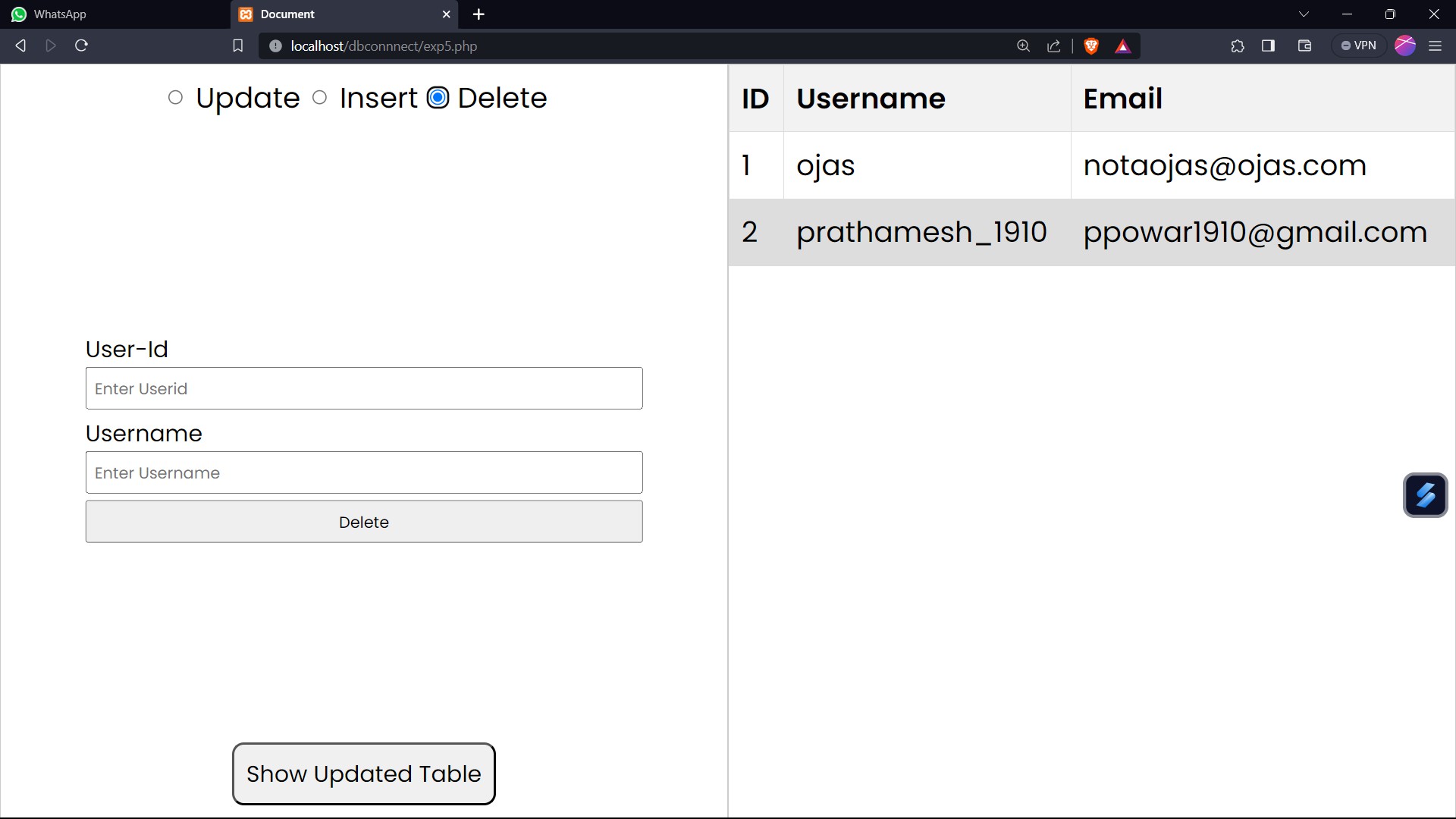
</script>

</html>

**Output:**



**Post Lab Questions:**



1. **Write in detail the importance of using Prepared Statements. Ans.**

**Security: Prepared statements help protect against SQL injection attacks.**

**They separate SQL code from user input, preventing malicious input from being directly executed as SQL code. Parameters are treated as data, not as executable code.**

**Performance: Prepared statements are precompiled by the database server, which can improve query execution time for queries that are executed multiple times. This is because the server can reuse the execution plan for the same query with different parameter values.**

**Readability: Prepared statements make SQL code more readable and maintainable. SQL queries with placeholders for parameters are often easier to understand than queries with concatenated values.**

**Compatibility: Prepared statements are supported by most modern database systems, making your code more portable. You can switch**

**between different database systems (e.g., MySQL, PostgreSQL, SQLite) with minimal code changes.**

1. **Write in detail about PDO constructor function. Ans.**

**The PDO constructor function is used to create a PDO (PHP Data Objects) instance, which is the primary way to connect to databases and interact with them using PDO. The constructor has the following syntax:**

**PDO::** **construct(string $dsn, string $username = null, string $password = null, array $options = [])**

**$dsn(Data Source Name): A string containing the information required to connect to the database, including the database type (e.g., "mysql", "pgsql"), hostname, port, and database name.**

**$username (Optional): The username to use when connecting to the database.**

**$password (Optional): The password associated with the username.**

**$options (Optional): An associative array of driver-speciﬁc connection options.**

**Example:**

**$dsn = "mysql:host=localhost;dbname=mydb";**

**$username = "user";**

**$password = "password";**

**try {**

**$pdo = new PDO($dsn, $username, $password);**

**// You now have a PDO instance to work with the database**

**} catch (PDOException $e) {**

**echo "Connection failed: " . $e->getMessage();**

**}**

1. **Write a php program to return the id of the last inserted row. Ans.**

**To retrieve the ID of the last inserted row in PDO, you can use the**

**`lastInsertId` method. Here's an example:**

**try {**

**$pdo = new PDO("mysql:host=localhost;dbname=mydb", "user", "password");**

**$sql = "INSERT INTO users (username, email) VALUES (:username,**

**:email)";**

**$stmt = $pdo->prepare($sql);**

**// Bind parameters and execute the query**

**$stmt->bindParam(':username', $username);**

**$stmt->bindParam(':email', $email);**

**$username = "john";**

**$email = "**[**john@example.com**](mailto:john@example.com)**";**

**$stmt->execute();**

**// Get the ID of the last inserted row**

**$lastInsertId = $pdo->lastInsertId(); echo "Last inserted ID: " . $lastInsertId;**

**} catch (PDOException $e) {**

**echo "Error: " . $e->getMessage();**

**}**

1. **What are the essential components of a PDO database connection string?**

**Ans.**

**A PDO database connection string typically consists of the following components:**

**Database Type: Speciﬁes the type of database you are connecting to (e.g., "mysql", "pgsql", "sqlite").**

**Hostname: The hostname or IP address of the database server.**

**Port: The port number where the database server is listening (optional, usually default values are used if not speciﬁed).**

**Database Name: The name of the speciﬁc database you want to connect to.**

**Username: The username used for authentication.**

**Password: The password associated with the username.**

1. **Explain the concept of database connection pooling and how it can be implemented with PDO.**

**Ans.**

**Database connection pooling is a technique used to efﬁciently manage and reuse database connections, reducing the overhead of creating new connections for each request. PDO itself does not provide built-in support for connection pooling. However, you can implement connection pooling in PHP using external libraries or frameworks, such as:**

**PDO Connection Pool Libraries: Some third-party libraries provide connection pooling for PDO. For example, "PDOPlus" is a PHP library that offers connection pooling for PDO.**

**Application Frameworks: Many PHP frameworks, such as Laravel, Symfony, and Yii, come with built-in support for database connection pooling. They manage connections for you, allowing you to focus on writing application logic.**

**Here's a high-level overview of how connection pooling can be implemented:**

* 1. **Maintain a pool of database connections that are created and initialized when the application starts.**
  2. **When a request comes in, retrieve a connection from the pool and use it for database operations.**
  3. **After completing the database operations, return the connection to the pool instead of closing it. This allows the connection to be reused for future requests.**
  4. **Implement logic to handle connection timeouts.**

**Outcome:**

**CO3: *CarrD ouĒ daĒabasg opgraĒions using ™H™***

**Conclusion:**

**Learned and implemented how PDO works with MySQL database and connected the mysql database with php and performed CRUD operations.**