FSD LAB 4

Aim: Write server-side script in PHP to perform form validation and create database application using PHP and MySQL to perform insert, update, delete and search operations.

Objectives:

To understand Server-side Scripting.

To learn database connectivity using PHP-MySQL.

To perform insert, update, delete and search operations on database.

Theory:

1. **PHP Architecture**

PHP (Hypertext Preprocessor) is a server-side scripting language commonly used for web development. Its architecture follows a request-response model. When a client sends an HTTP request to a server, PHP processes the script embedded in the requested page, interacts with databases, performs business logic, and generates dynamic content. PHP can seamlessly integrate with various databases, and its modular structure supports the use of extensions for added functionalities. The processed output is then sent back to the client as an HTTP response. PHP is often deployed alongside web servers like Apache or Nginx, forming a robust and widely-used technology stack.

2. Steps for Database connectivity in PHP.

Connecting PHP to a database involves several steps. Below are the general steps for establishing database connectivity in PHP using MySQL as an example. Keep in mind that the specific details may vary depending on the database management system you are using (MySQL, PostgreSQL, SQLite, etc.).

1. **Install a Database:**
   - Choose a database management system and install it on your server. For example, you might use MySQL, MariaDB, PostgreSQL, SQLite, etc.
2. **Create a Database:**
   - Using tools like phpMyAdmin or command-line interfaces, create a database and define its structure.
3. **Get Database Connection Information:**
   - Note down the database host (usually "localhost" for local development), database name, username, and password. You might need these details for the PHP script.
4. **Choose a PHP Extension for Database Interaction:**
   - PHP provides several extensions for interacting with databases. The commonly used ones are mysqli (improved MySQL) and PDO (PHP Data Objects).
5. **Connect to the Database:**

- Use the appropriate PHP extension to establish a connection to the database. Here's an example using mysqli:

```
<?php
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$dbname = "your_database";

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

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

FAQ:

1. What are the advantages of Server-side Scripting?

Server-side scripting refers to the execution of scripts on a web server to generate dynamic web pages. PHP, Python, Ruby, and ASP.NET are examples of server-side scripting languages. Here are several advantages of using server-side scripting in web development:

1. **Dynamic Content Generation:**

- Server-side scripting allows the creation of dynamic content that can change based on various factors such as user input, database queries, or external data sources. This dynamic nature enhances the user experience by providing personalized and up-to-date information.

2. **Database Interaction:**

- Server-side scripting languages can easily connect to databases, enabling the storage and retrieval of data. This interaction is crucial for applications that require persistent data storage, retrieval, and manipulation.

3. **Enhanced Security:**

- Server-side scripts execute on the web server, and only the output (HTML, CSS, JavaScript) is sent to the client's browser. This helps in keeping sensitive business logic and database interactions secure, as they are not exposed to the client-side code.

4. **Code Protection:**

- Server-side code remains on the server and is not visible to users, providing a layer of protection for the application's logic and algorithms. This helps prevent unauthorized access to or modification of critical code.

5. **Centralized Codebase:**

- With server-side scripting, the application's logic is centralized on the server, making it easier to manage and update. Changes made on the server are immediately reflected for all users, ensuring consistency across the application.

6. **Scalability:**

- Server-side scripting allows for better scalability, as most of the processing is done on the server. This reduces the load on client devices and ensures that the application can handle a larger number of concurrent users without compromising performance.

7. **Cross-Browser Compatibility:**

- Server-side scripting helps address cross-browser compatibility issues, as the server generates HTML, CSS, and JavaScript that is sent to the client. This reduces the need for complex client-side code to handle variations between different web browsers.

8. **SEO-Friendly:**

- Server-side scripting allows for server-rendered content, which is often more favorable for search engine optimization (SEO). Search engines can easily crawl and

index content generated on the server, potentially improving the website's visibility in search results.

9. **User Authentication and Authorization:**

- Server-side scripting facilitates the implementation of robust user authentication and authorization mechanisms. User credentials and access permissions are typically managed on the server, reducing the risk of security vulnerabilities.

10. **Session Management:**

- Server-side scripting enables the management of user sessions, allowing the server to keep track of user-specific information across multiple requests. This is essential for maintaining stateful interactions and user-specific data.

In summary, server-side scripting offers several advantages, including dynamic content generation, enhanced security, centralized code management, scalability, and better support for database interactions. These benefits make server-side scripting an integral part of building robust and secure web applications.

2. What is XAMPP and phpMyAdmin?

XAMPP and phpMyAdmin are both tools commonly used in web development, especially for local development environments.

1. **XAMPP:**

XAMPP is a free and open-source cross-platform web server solution stack package developed by Apache Friends. The name "XAMPP" is an acronym, which stands for:

- **X:** Stands for any of the operating systems (Windows, Linux, macOS, or Solaris).

- **A:** Apache, the web server.

- **M:** MySQL, the database management system.

- **P:** PHP, Perl, or Python, the scripting languages used for web development.

Additionally, XAMPP includes other software components like phpMyAdmin, OpenSSL, and FileZilla.

XAMPP simplifies the process of setting up a local development environment for web developers. It provides an easy-to-install package that includes Apache for serving web pages, MySQL for database management, and PHP for server-side scripting. This combination allows developers to test and develop web applications locally before deploying them to a live server.

2. **phpMyAdmin:**

phpMyAdmin is a free and open-source web-based tool written in PHP that provides a graphical interface for managing MySQL databases. With phpMyAdmin, users can perform various database tasks, such as creating, modifying, and deleting databases, tables, and records. It also allows for running SQL queries, importing and exporting data, and managing user permissions.

Key features of phpMyAdmin include an intuitive web interface, support for multiple database servers, the ability to execute SQL commands, and support for common database operations. It is widely used by developers and administrators to interact with MySQL databases without needing to use the command-line interface.

phpMyAdmin is often included as part of the XAMPP package, making it convenient for developers who use XAMPP for local development. However, phpMyAdmin can also be installed separately on other web servers that support PHP and MySQL.

3. What are the two ways to connect to a database in PHP?

In PHP, there are primarily two ways to connect to a database: using the MySQLi extension and using the PDO (PHP Data Objects) extension. Both approaches provide a way to interact with various database management systems, not just MySQL.

1. **MySQLi Extension:**

MySQLi (MySQL Improved) is a PHP extension specifically designed for working with MySQL databases. It offers both procedural and object-oriented interfaces for interacting with databases.

Procedural Approach:

```php
<?php
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$dbname = "your_database";

// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);

// Check connection
if (!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";
// Perform database operations here
// Close connection when done: mysqli_close($conn);
?>
```

Object-Oriented Approach:

```php
```

```
<?php
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$dbname = "your_database";

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

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
// Perform database operations here
// Close connection when done: $conn->close();
?>
```

2. **PDO (PHP Data Objects) Extension:**

PDO is a database access layer providing a uniform method of access to multiple databases. It supports a wide range of database management systems, including MySQL, PostgreSQL, SQLite, and more. PDO provides a consistent interface for database access regardless of the specific database being used.

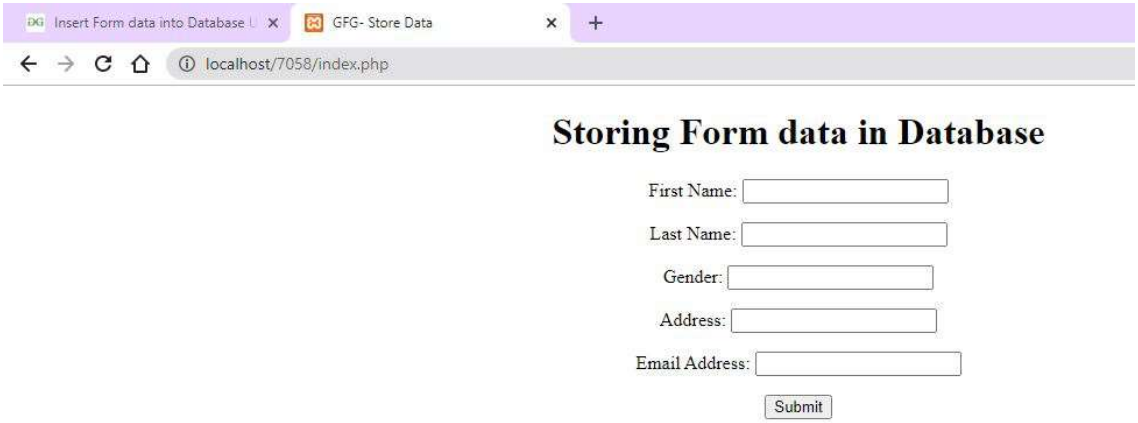
```php
<?php
$servername = "localhost";
$username = "your_username";
```

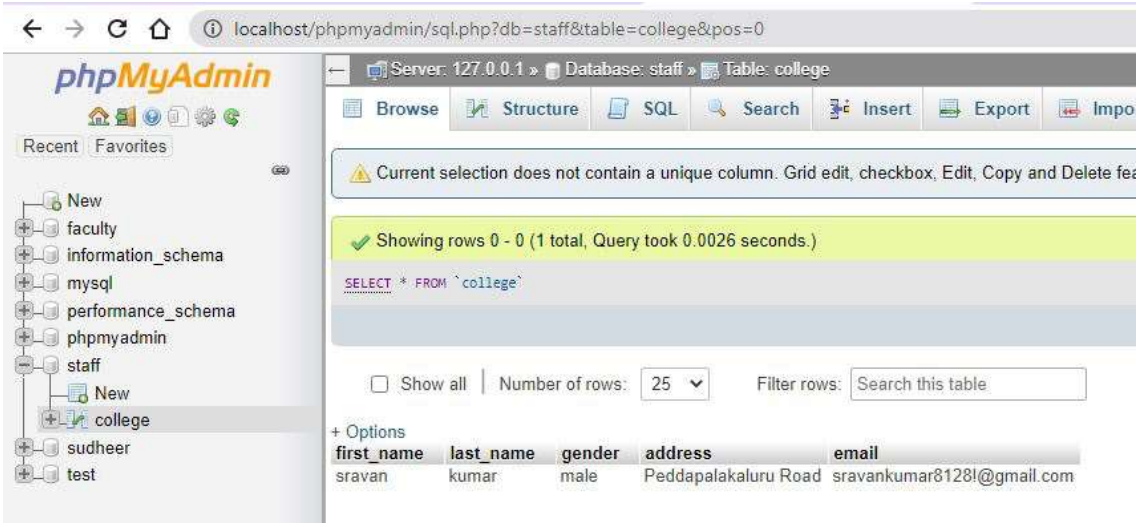
```
$password = "your_password";
$dbname = "your_database";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    // Set the PDO error mode to exception
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
    // Perform database operations here
} catch (PDOException $e) {
    echo "Connection failed: " . $e->getMessage();
}
// Close connection when done: $conn = null;
?>
```

Both MySQLi and PDO are effective ways to connect to databases in PHP. The choice between them often comes down to personal preference or specific project requirements. PDO is generally favored for its broader database support and consistent interface, while MySQLi may be preferred for projects primarily working with MySQL due to its MySQL-specific features.

Screenshots –





CODE –

INDEX.PHP

```
<!DOCTYPE html>
<html lang="en">
<head>
        <title>GFG- Store Data</title>
```

```
</head>
<body>
	<center>
		<h1>Storing Form data in Database</h1>
		<form action="insert.php" method="post">

<p>
			<label for="firstName">First Name:</label>
			<input type="text" name="first_name" id="firstName">
			</p>


<p>
			<label for="lastName">Last Name:</label>
			<input type="text" name="last_name" id="lastName">
			</p>


<p>
			<label for="Gender">Gender:</label>
			<input type="text" name="gender" id="Gender">
			</p>


<p>
			<label for="Address">Address:</label>
			<input type="text" name="address" id="Address">
			</p>
```

```
<p>
                        <label for="emailAddress">Email Address:</label>
                        <input type="text" name="email" id="emailAddress">
                        </p>
<!DOCTYPE html>
<html>

<head>
        <title>Insert Page page</title>
</head>

<body>
        <center>
                <?php

                // servername => localhost
                // username => root
                // password => empty
                // database name => staff
                $conn = mysqli_connect("localhost", "root", "", "staff");

                // Check connection
                if($conn === false){
                        die("ERROR: Could not connect. "
                                . mysqli_connect_error());
                }

                // Taking all 5 values from the form data(input)
```

```
            $first_name = $_REQUEST['first_name'];
            $last_name = $_REQUEST['last_name'];
            $gender = $_REQUEST['gender'];
            $address = $_REQUEST['address'];
            $email = $_REQUEST['email'];

            // Performing insert query execution
            // here our table name is college
            $sql = "INSERT INTO college VALUES ('$first_name',
                '$last_name','$gender','$address','$email')";

            if(mysqli_query($conn, $sql)){
                echo "<h3>data stored in a database successfully."
                    . " Please browse your localhost php my admin"
                    . " to view the updated data</h3>";

                echo nl2br("\n$first_name\n $last_name\n "
                    . "$gender\n $address\n $email");
            } else{
                echo "ERROR: Hush! Sorry $sql. "
                    . mysqli_error($conn);
            }

            // Close connection
            mysqli_close($conn);
            ?>
        </center>
</body>
```

```
</html>                         <input type="submit" value="Submit">
                    </form>
          </center>
</body>
</html>
```

INSERT.PHP