University of Anbar

College of Computer Sciences and

Information Technology

Web Development

Dr. Alaa Abdalqahar Jihad

Fourth Stage – Second Course

Lecture 7: Working with Databases (MySQL + PHP)

Objectives:

- Understand the role of databases in web applications.
- Set up and configure MySQL using tools like phpMyAdmin or command-line interfaces.
- Create databases and tables, and perform basic SQL operations.
- Connect to a MySQL database from PHP using the **mysqli** extension.
- Execute basic CRUD (Create, Read, Update, Delete) operations via PHP.

Introduction to Databases

- **What is a Database?**
  - A structured collection of data that is stored, managed, and retrieved electronically.
- **MySQL Overview:**
  - An open-source relational database management system (RDBMS).
  - Widely used in web applications due to its performance, reliability, and ease of integration with PHP.
- **Role in Web Development:**
  - Used for storing user information, product data, blog posts, etc.

Setting Up MySQL

- **Installation:** Using **XAMPP** which includes MySQL along with Apache and PHP.
- **Using phpMyAdmin:**
  - Access phpMyAdmin through http://localhost/phpmyadmin.
  - Overview of its interface: Creating databases, tables, and running queries.

Creating a Database and Tables

- **Database Creation via phpMyAdmin:** Create a database named web_app.
- **Table Creation:** Example: Creating a users table with fields such as id, username, email, and password.

SQL Example:

```
CREATE TABLE users (
   id INT AUTO_INCREMENT PRIMARY KEY,
   username VARCHAR(50) NOT NULL,
   email VARCHAR(100) NOT NULL,
   password VARCHAR(255) NOT NULL
);
```

- **Discussion:** Data types, primary keys, and auto-increment features.

Connecting PHP to MySQL

- **Using the MySQLi Extension:** Establish a connection to the MySQL server.

Code Example:

```
<?php
$servername = "localhost";
$username = "root";
$password = ""; // Default password for local servers (change if needed)
$dbname = "web_app";

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

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

Performing CRUD Operations

- **Create (INSERT):** Inserting a new record into the users table.

Code Example:

```
<?php
$stmt = $conn->prepare("INSERT INTO users (username, email,
password) VALUES (?, ?, ?)");
$stmt->bind_param("sss", $username, $email, $password);

$username = "Ali";
```

```
$email = "Ali@example.com";
$password = password_hash("secret123", PASSWORD_DEFAULT);

$stmt->execute();
echo "New record created successfully";
$stmt->close();
?>
```

- **Read (SELECT):** Fetching and displaying data from the database.

Code Example:

```
<?php
$sql = "SELECT id, username, email FROM users";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
   while ($row = $result->fetch_assoc()) {
      echo "ID: " . $row["id"] . " - Username: " . $row["username"] . " -
Email: " . $row["email"] . "<br>";
   }
} else {
   echo "0 results";
}
?>
```

- **Update (UPDATE):** Modifying an existing record.

Code Example:

```
<?php
```

```
$stmt = $conn->prepare("UPDATE users SET email = ? WHERE
username = ?");
$stmt->bind_param("ss", $newEmail, $username);

$newEmail = "Ali_new@example.com";
$username = "Ali";

$stmt->execute();
echo "Record updated successfully";
$stmt->close();
?>
```

- **Delete (DELETE):** Removing a record from the table.

Code Example:

```
<?php
$stmt = $conn->prepare("DELETE FROM users WHERE username =
?");
$stmt->bind_param("s", $username);

$username = "Ali";
$stmt->execute();
echo "Record deleted successfully";
$stmt->close();
?>
```

Best Practices and Security

- **Prepared Statements:** Prevent SQL injection by using prepared statements for all SQL queries.
- **Error Handling:** Always check for errors during database operations.
- **Closing Connections:** Ensure that database connections are properly closed after operations.

Task:

- Create a PHP script that:
    1. Connects to the web_app database.
    2. Inserts a new user record using prepared statements.
    3. Retrieves and displays all user records.

Summary

- **Recap Key Points:**
    - How to set up MySQL and use phpMyAdmin.
    - Creating databases and tables.
    - Connecting PHP to MySQL using the MySQLi extension.
    - Implementing CRUD operations securely with prepared statements.

Homework Assignment:

Create a PHP-based mini-application that:

1. Sets up a products table with fields such as id, product_name, description, and price.
2. Implements functions to add a new product, update product details, delete a product, and list all products.
3. Uses prepared statements for all SQL operations.
4. Provides clear feedback messages for each operation.

Lecture 8: Sessions and Cookies

Objectives:

- Understand the concept of user state in web applications.
- Distinguish between sessions and cookies.
- Create, read, update, and delete session data in PHP.
- Set, retrieve, and manage cookies securely.
- Implement basic user authentication features using sessions and cookies.

Introduction to User State

- **What is User State?**
  - Explanation of state management in stateless HTTP.
  - Importance of preserving user data across multiple pages (e.g., login status).
- **Overview:**
  - **Sessions:** Server-side storage of user data.
  - **Cookies:** Client-side data storage in the user’s browser.

Understanding Sessions

- **What Are Sessions?**
  - Sessions store data on the server and maintain state between HTTP requests.
- **How Sessions Work:**

    - PHP generates a unique session ID for each user.
    - Session data is stored on the server and referenced by the session ID.
- **Basic Session Management in PHP:**
    - Starting a session using session_start().
    - Storing and retrieving session variables.

Code Example:

```
<?php
// Start the session
session_start();

// Set session variables
$_SESSION["username"] = "Ali";
$_SESSION["role"] = "admin";

// Retrieve session data
echo "Welcome, " . $_SESSION["username"];
?>
```

- **Discussion:** Lifetime of a session, session storage, and session security.

Understanding Cookies

- **What Are Cookies?** Cookies are small pieces of data stored on the client-side (in the user's browser).
- **How Cookies Work:**

  - Set using PHP’s setcookie() function.
  - Automatically sent by the browser with every HTTP request to the domain.

- **Basic Cookie Operations:** Setting, accessing, and deleting cookies.

Code Example:

```
<?php
// Set a cookie that expires in 1 hour
setcookie("user", "Ali", time() + 3600, "/");

// Retrieve a cookie value
if(isset($_COOKIE["user"])) {
   echo "User is: " . $_COOKIE["user"];
} else {
   echo "User cookie is not set.";
}
?>
```

- **Security Considerations:** Cookie encryption, secure flags, HTTPOnly, and SameSite attributes.

Sessions vs. Cookies

Key Differences:

Aspect	**Sessions**	**Cookies**
Storage	Server-side	Client-side (browser)
Security	Generally more secure	Vulnerable to tampering if not secured
Capacity	Can store larger data amounts	Limited size (typically 4KB)

Lifespan	Lasts until session ends or times out	Set with an expiration time

- **When to Use Each:**
  - Use sessions for sensitive data (e.g., login status, user roles).
  - Use cookies for non-sensitive data that must persist between sessions (e.g., user preferences).

Implementing User Authentication

- **Using Sessions for Authentication:**
  - How to check if a user is logged in.
  - Creating login and logout scripts.
- **Example Workflow:**

1. User logs in → Session variable is set.
2. Subsequent pages check for the session variable.
3. User logs out → Session data is cleared.

Code Example for Logout:

```
<?php
session_start();
// Remove all session variables
session_unset();
// Destroy the session
session_destroy();
echo "You have been logged out.";
?>
```

Best Practices and Security

- **Securing Sessions:**
  - Use session_regenerate_id() to prevent session fixation.
  - Store session data in a secure location and use proper session timeout.
- **Securing Cookies:**
  - Set cookies with the HTTPOnly and Secure flags.
  - Consider using encryption for sensitive data stored in cookies.
- **Common Pitfalls:**
  - Avoid exposing session IDs.
  - Prevent cross-site scripting (XSS) attacks that target cookies.

Task:

- Create a PHP application that:
  1. Starts a session and stores user login information.
  2. Sets a cookie to remember the user's preferred language.
  3. Displays a welcome message using session data and shows the language preference from the cookie.
  4. Provides a logout button that clears the session.

Example:

- **login.php:**

```
<?php
session_start();
// Simulate a login action
$_SESSION["username"] = "Ali";
setcookie("language", "English", time() + 86400, "/"); // 1-day cookie
header("Location: welcome.php");
?>
```

- **welcome.php:**

```
<?php
session_start();
if (!isset($_SESSION["username"])) {
    header("Location: login.php");
    exit;
}
$username = $_SESSION["username"];
$language = isset($_COOKIE["language"]) ? $_COOKIE["language"] :
"not set";
echo "Welcome, $username!<br>";
echo "Your preferred language is: $language<br>";
echo "<a href='logout.php'>Logout</a>";
?>
```

- **logout.php:**

```
<?php
session_start();
```

```
session_destroy();
echo "You have been logged out.";
?>
```

Summary

- **Recap Key Points:**
  - Sessions store data on the server; cookies store data on the client.
  - Use sessions for sensitive information and cookies for preferences.
  - Secure session and cookie handling is crucial for web application security.

Homework Assignment:

Create a PHP script that:

1. Implements a simple user login system using sessions.
2. Sets a cookie to remember the user's theme preference (e.g., light or dark mode).
3. Provides functionality to update the theme preference and reflect it on subsequent visits.
4. Includes proper validation and security measures.