Dept. of Computing and Information Systems

Comprehensive Solution Guide: Quiz 03 — Spring 2026

Course: Web Engineering, CIS324

SET A: Solutions & Explanations

Q1: Cookies vs. Sessions and Overcoming "Web Amnesia" [7 Marks]

1. Fundamental Differences:

- **Location:** A Cookie is stored on the client-side (the user's browser). A Session is stored on the server-side (the website's physical server).
- **Security:** Cookies have extremely low security because they live on the user's computer; anyone can open their browser developer tools and view or edit them. Sessions have high security because the raw data never leaves the server, making it inaccessible and tamper-proof to the user.
- **Data Type (The Analogy):** Think of a Cookie as an everyday *Identification Card*. It holds non-sensitive data like a theme preference (dark/light mode). Think of a Session as a *Secure Locker*. It holds highly sensitive data that must be protected, like a login status or a shopping cart.

2. Overcoming Stateless HTTP: By default, the web suffers from "amnesia." Because HTTP is a stateless protocol, every time a user clicks a link, the server treats them like a brand-new stranger. Cookies and Sessions work together using a "Key and Locker" system to fix this:

1. When a user visits, the server creates an empty, secure Session (a locker) for them.
2. The server generates a unique, random string called a **Session ID** (the key to the locker).
3. The server places this Session ID inside a Cookie and sends it to the user's browser.
4. On every future click, the browser automatically shows the Cookie containing the ID. The server reads the ID, finds the matching locker, and instantly "remembers" the user.

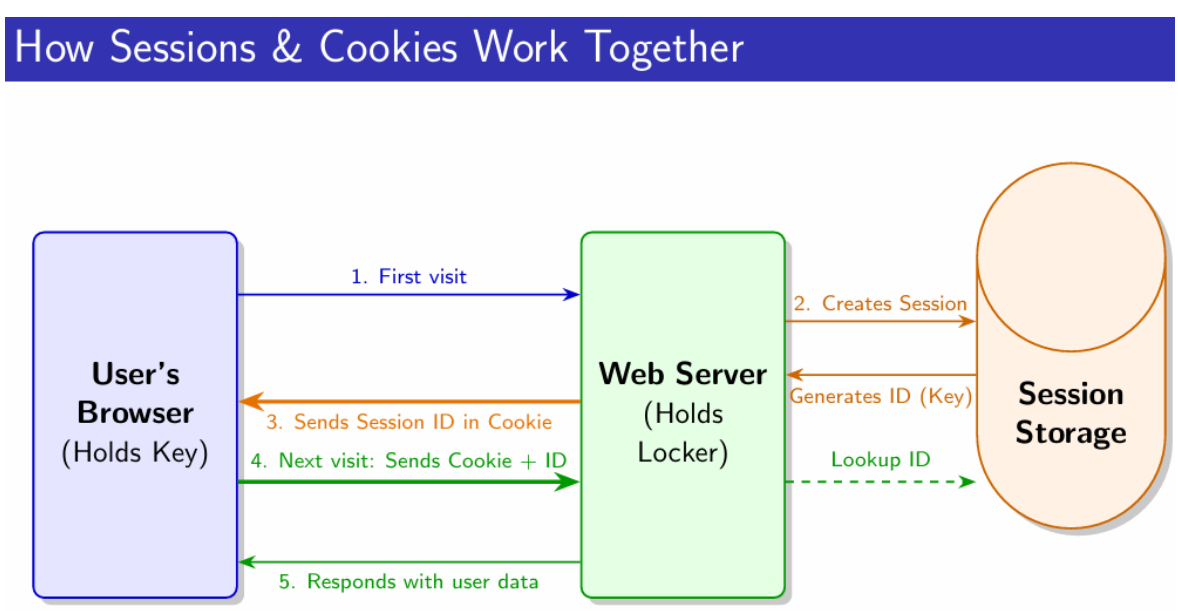

Figure 1: How Sessions & Cookies Work Together.

Q2: Database Connectivity Steps and PHP Script [8 Marks]

1. The 5 Essential Steps:

1. **Set Login Details:** Define variables for the server, username, password, and database.
2. **Try to Connect:** Call the `mysqli_connect()` function.
3. **Check for Errors:** Verify if the connection variable is empty/false. If it failed, stop the script and print the error using `die()`.
4. **Execute & Confirm:** Run the SQL queries using `mysqli_query()`.
5. **Close Connection:** Terminate the connection at the end using `mysqli_close()`.

2. Complete PHP Script:

```
<?php
// Step 1: Set Login Details
$servername = "localhost";
$username = "root";
$password = "";
$dbname = "myFirstDB";

// Step 2: Try to Connect
$conn = mysqli_connect($servername, $username, $password, $dbname);

// Step 3: Check for Errors
if (!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}

// Step 4: Execute Query
$sql = "SELECT id, firstname, lastname FROM Guests";
$result = mysqli_query($conn, $sql);

// Think of $result as a stack of papers. We use a while loop
// to pick up one paper ($row) at a time until the stack is empty.
while ($row = mysqli_fetch_assoc($result)) {
    echo "ID: " . $row["id"] . " - Name: " . $row["firstname"] . " " . $row["
       lastname"] . "<br>";
}

// Step 5: Close Connection
mysqli_close($conn);
?>
```

SET B: Solutions & Explanations

Q1: Session Hijacking and the "Remember Me" Mechanism [7 Marks]

1. Session Hijacking: While the session locker on the server is practically impenetrable, the "Key" to that locker (the Session ID) travels back and forth across the internet inside a Cookie.

- **The Attack:** If a hacker sits on the same public Wi-Fi network as the user, they can intercept this Cookie. The hacker can then send that exact same key to the server. The server, seeing a valid key, assumes the hacker is the legitimate user and opens the secure locker for them.
- **Prevention:**
  1. Use **HTTPS** to encrypt the cookie while it travels.
  2. Provide functional **Logout buttons** that execute `session_destroy()`. This physically destroys the locker on the server, rendering any stolen keys instantly useless.

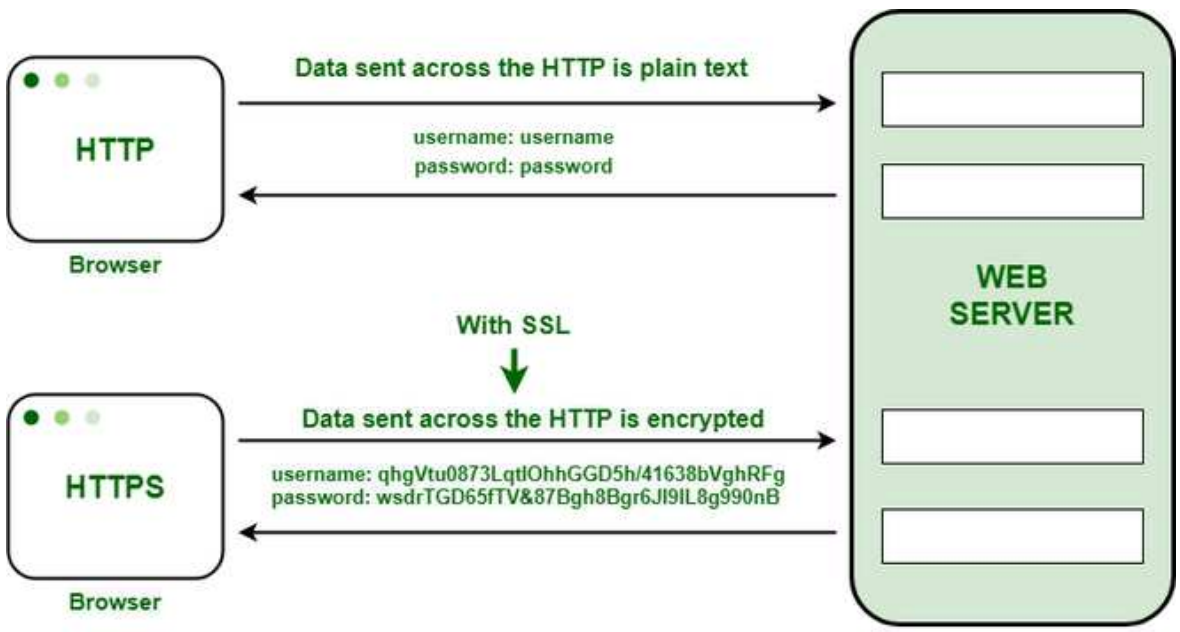

Figure 2: Working of HTTPS.

2. The "Remember Me" Mechanism: If a session is destroyed the moment a user closes their browser, how do platforms like Google keep you logged in for months?

- They do *not* keep your original session open forever. Your session still dies when you close the tab.
- Instead, when you check the "Remember Me" box, the server creates a special, highly-encrypted **Long-Term Cookie** (think of it as a VIP Pass) set to expire in 30 days or more.
- When you return to the site the next day, your browser shows this VIP Cookie. The server verifies it and instantly builds a **brand new session** in the background. It feels like you never logged out, but a fresh session was actually generated specifically for your new visit.

Q2: MySQLi vs. PDO and Cookie Error Handling [8 Marks]

1. MySQLi vs. PDO:

- **MySQLi (Improved):** This extension is simpler but strictly supports *only* MySQL databases. If you ever need to change your database system, you must rewrite all your procedural code.

- **PDO (PHP Data Objects):** This extension is incredibly flexible because it can talk to 12 different kinds of databases (including PostgreSQL, SQLite, Oracle). If you switch databases later, you only have to change the connection string, not your entire codebase.

2. The "Headers Already Sent" Error: This error is triggered when a developer makes a critical architectural mistake: placing HTML tags (like `<html>`, `<head>`, or `<body>`), plain text, or even empty white space *before* calling the `setcookie()` function. In PHP, you MUST issue the cookie command before a single piece of visual output or HTML is sent to the browser screen.

3. Correct PHP Script:

```
<?php
// The cookie MUST be set at the very top of the document,
// strictly before any HTML tags or blank spaces are printed!
$chosen_theme = "dark";

// Save their choice in a cookie named 'my_theme' for 30 days
setcookie("my_theme", $chosen_theme, time() + (86400 * 30));
?>
<!DOCTYPE html>
<html>
<head>
    <title>Theme Saver</title>
</head>
<body style="background-color: #333; color: white;">
    <h2>Your theme has been saved securely!</h2>
</body>
</html>
```