

UNIT 5

Advanced Server Side Issues

(9 Hours) — Web Technology (CACS205)

5.1 Database Connectivity

Most web applications store data in a relational database. **Database connectivity** is the ability of a server-side program to connect to a database, send SQL commands and process the results. The program uses a *driver* that talks to the database engine.



Connect → Prepare SQL → Execute → Fetch result set → Close connection

Figure 5.1 — Web application to database connectivity

Technology	Platform / Language	Description
JDBC	Java (servlets, JSP)	Java Database Connectivity API; uses a vendor driver (e.g. MySQL Connector/J).
ODBC	Windows / C / ASP	Open Database Connectivity; a generic driver interface.
ADO.NET	.NET / ASP.NET	Data-access classes: SqlConnection, SqlCommand, DataReader.
mysqli / PDO	PHP	PHP extensions to access MySQL and other databases.

5.1.1 Steps in Connecting to a Database

- **Load the driver** (JDBC: `Class.forName("com.mysql.cj.jdbc.Driver")`).
- **Open a connection** using a URL, user name and password.
- **Create a statement** (Statement / PreparedStatement) holding the SQL.
- **Execute it**: `executeQuery()` for SELECT; `executeUpdate()` for INSERT, UPDATE, DELETE.

- **Process the ResultSet** row by row.
- **Close** the ResultSet, statement and connection (release resources).

JDBC connection and query

```
import java.sql.*;

public class DbDemo {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/college";
        try (Connection con = DriverManager.getConnection(url, "root",
"secret");
            Statement st = con.createStatement();
            ResultSet rs = st.executeQuery("SELECT id, name FROM student")) {

            while (rs.next()) {
                System.out.println(rs.getInt("id") + " " +
rs.getString("name"));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

The same in PHP (mysqli)

```
<?php
$conn = new mysqli("localhost", "root", "secret", "college");
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
$result = $conn->query("SELECT id, name FROM student");
while ($row = $result->fetch_assoc()) {
    echo $row["id"] . " - " . $row["name"] . "<br>";
}
$conn->close();
?>
```

5.2 Creating an SQL Statement

SQL (Structured Query Language) is the standard language for relational databases. The four basic data-manipulation statements are shown below for a table `student(id, name, age, city)`.

Statement	Syntax	Purpose
SELECT	SELECT col1, col2 FROM table WHERE cond ORDER BY col;	Retrieve rows.
INSERT	INSERT INTO table (c1, c2) VALUES (v1, v2);	Add a new row.

Statement	Syntax	Purpose
UPDATE	UPDATE table SET c1 = v1 WHERE cond;	Modify existing rows.
DELETE	DELETE FROM table WHERE cond;	Remove rows.

SQL statement examples

```
-- SELECT
SELECT name, age FROM student WHERE city = 'Kathmandu' ORDER BY name;
SELECT COUNT(*) FROM student;

-- INSERT
INSERT INTO student (id, name, age, city) VALUES (101, 'Sita', 20, 'Pokhara');

-- UPDATE
UPDATE student SET city = 'Lalitpur' WHERE id = 101;

-- DELETE
DELETE FROM student WHERE id = 101;
```

Note: Always include a WHERE clause with UPDATE and DELETE; without it every row of the table is changed or removed.

5.2.1 Executing SQL from a Web Program

SQL statements are usually built from user input. Concatenating raw input into SQL causes **SQL injection** — an attacker enters text such as ' OR '1'='1 to change the meaning of the query. The remedy is **prepared (parameterised) statements**, which send the SQL and the values separately.

Prepared statements prevent SQL injection

```
// UNSAFE: string concatenation
String sql = "SELECT * FROM users WHERE name='" + name + "' AND pass='" + pass + "'";

// SAFE: PreparedStatement with placeholders
PreparedStatement ps = con.prepareStatement(
    "SELECT * FROM users WHERE name = ? AND pass = ?");
ps.setString(1, name);
ps.setString(2, pass);
ResultSet rs = ps.executeQuery();

// INSERT / UPDATE / DELETE return the number of rows affected
PreparedStatement ins = con.prepareStatement(
    "INSERT INTO student(id,name,age) VALUES (?, ?, ?)");
ins.setInt(1, 102); ins.setString(2, "Ram"); ins.setInt(3, 21);
int rows = ins.executeUpdate();
```

Example 5.1 — Web form to database

A form posts fields `id`, `name` and `age` to `save.jsp`. Describe how the data is stored in the student table.

Solution

`save.jsp` reads the fields using `request.getParameter("id")` etc., opens a connection, prepares `INSERT INTO student(id,name,age) VALUES (?, ?, ?)`, sets the three parameters (converting `id` and `age` to `int`), calls `executeUpdate()`, checks that 1 row was affected, closes the connection and redirects the user to a confirmation page.

5.3 Authentication

Authentication is the process of verifying *who* a user is; **authorisation** then decides *what* that user may do. A web server can restrict access to resources using several methods, most of them configured in the web server (such as Microsoft IIS).

Method	How it works	Remarks
Anonymous access	No user name or password is requested; the server runs the request as a default guest account (e.g. IUSR in IIS).	Used for public web sites.
Authentication by IP address and domain	Access is allowed or denied depending on the client's IP address, IP range or domain name.	Simple, but IPs can be shared or spoofed; domain checks need reverse DNS lookups.
Basic authentication	Browser prompts for credentials, sends them Base64-encoded in the Authorization header.	Not encrypted — use only over HTTPS.
Digest authentication	Server sends a challenge; client returns a hash of the credentials.	Password not sent in clear text.
Integrated Windows Authentication	Uses the user's Windows login (NTLM or Kerberos) automatically; no prompt in Internet Explorer/Edge on a domain.	Best for intranets; requires Windows clients on the same domain.
Form-based authentication	Application shows its own login form and stores the login state in a session.	Most common on the Internet; must use HTTPS and hashed passwords.

Restricting a URL pattern to a role

```
<!-- Java EE: web.xml form-based / basic authentication -->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Admin</web-resource-name>
    <url-pattern>/admin/*</url-pattern>
  </web-resource-collection>
```

```
<auth-constraint><role-name>admin</role-name></auth-constraint>
</security-constraint>
<login-config><auth-method>BASIC</auth-method></login-config>
```

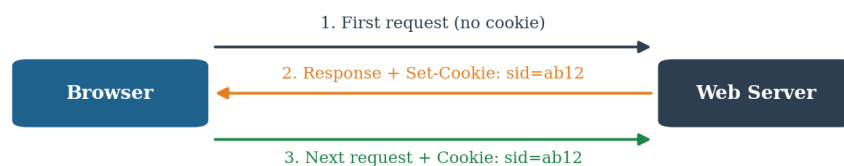
Form-based login (PHP)

```
<?php
// Simple form-based login using a session
session_start();
$stmt = $conn->prepare("SELECT pass_hash FROM users WHERE name = ?");
$stmt->bind_param("s", $_POST["user"]);
$stmt->execute();
$stmt->bind_result($hash);
if ($stmt->fetch() && password_verify($_POST["pass"], $hash)) {
    $_SESSION["user"] = $_POST["user"]; // login succeeded
    header("Location: dashboard.php");
} else {
    echo "Invalid user name or password";
}
?>
```

Note: Never store passwords in plain text; store a salted hash (bcrypt, Argon2) and protect the login page with HTTPS.

5.4 Cookies

A **cookie** is a small piece of text (name = value) that the server asks the browser to store and return with every later request to the same site. Cookies overcome HTTP's statelessness and are used for sessions, personalisation, shopping carts and tracking.



Server uses the cookie value to recognise the same user across requests

Figure 5.2 — Cookie exchange between browser and server

Cookie attribute	Meaning
name=value	The data being stored (limit about 4 KB per cookie).
Expires / Max-Age	When the cookie is deleted. Without it: a session cookie that is lost when the browser closes. With it: a persistent cookie .

Cookie attribute	Meaning
Domain / Path	Which host and URL paths the browser sends the cookie back to.
Secure	Send only over HTTPS.
HttpOnly	Not accessible to JavaScript — protects against theft by XSS.
SameSite	Limits sending with cross-site requests (helps prevent CSRF).

Cookies in PHP

```
<?php
// PHP: set a cookie valid for 7 days, then read and delete it
setcookie("theme", "dark", time() + 7*24*3600, "/");
echo $_COOKIE["theme"] ?? "no cookie";
setcookie("theme", "", time() - 3600, "/"); // delete
?>
```

Cookies in a Java servlet

```
// Java servlet: create a cookie
Cookie c = new Cookie("user", "ram");
c.setMaxAge(60 * 60 * 24); // one day
c.setHttpOnly(true);
response.addCookie(c);

// read cookies from the request
Cookie[] all = request.getCookies();
if (all != null)
    for (Cookie k : all)
        if (k.getName().equals("user")) out.println(k.getValue());
```

Cookies	Sessions
Stored on the client browser	Data stored on the server (only the id is in a cookie)
Less secure; user can edit them	More secure
Small size limit	Limited by server memory
Can persist for days or years	Ends on logout or timeout

5.5 File Handling

Server-side programs often need to **read, write, append, upload and delete files** — for logs, reports, user uploads and simple data storage. The usual steps are: open the file in a suitable *mode*, read or write data, and close the file.

Mode (PHP fopen)	Meaning
r	Read only; file pointer at start.

Mode (PHP fopen)	Meaning
r+	Read and write.
w	Write only; creates the file or erases existing contents.
a	Append; writes at the end, creating the file if missing.
x	Create new file for writing; fails if it already exists.

File handling in PHP

```
<?php
// Write (append) a line to a log file
$fp = fopen("log.txt", "a");
fwrite($fp, date("Y-m-d H:i") . " - user visited\n");
fclose($fp);

// Read the whole file / line by line
echo file_get_contents("log.txt");
$fp = fopen("log.txt", "r");
while (($line = fgets($fp)) !== false) { echo $line . "<br>"; }
fclose($fp);

// Other useful functions
file_exists("log.txt"); filesize("log.txt"); unlink("old.txt");
copy("a.txt", "b.txt");
?>
```

File handling in Java

```
import java.io.*;

try (BufferedWriter w = new BufferedWriter(new FileWriter("log.txt", true))) {
    w.write("user visited"); w.newLine();
}
try (BufferedReader r = new BufferedReader(new FileReader("log.txt"))) {
    String line;
    while ((line = r.readLine()) != null) System.out.println(line);
}
```

5.5.1 File Upload

To upload a file the form must use `method="post"` and `enctype="multipart/form-data"` with an `<input type="file">` field. The server stores the upload in a temporary location and the script moves it to a permanent folder after checking it.

Uploading a file

```
<!-- upload.html -->
<form action="upload.php" method="post" enctype="multipart/form-data">
    <input type="file" name="photo">
    <input type="submit" value="Upload">
</form>
```

```
<?php // upload.php
$f = $_FILES["photo"];
$ok = ["image/jpeg", "image/png"];
if ($f["error"] === 0 && in_array($f["type"], $ok) && $f["size"] < 2000000) {
    move_uploaded_file($f["tmp_name"], "uploads/" . basename($f["name"]));
    echo "Uploaded";
} else { echo "Invalid file"; }
?>
```

Note: Security: validate file type, size and name; store uploads outside the web root or rename them; never trust the client-supplied file name or MIME type.

5.6 Form Handling

Form handling is the server-side processing of data submitted from an HTML form: receive, validate, process (store, e-mail, compute) and respond. The values arrive as name–value pairs; the way the program reads them depends on the technology.

Technology	GET data	POST data
PHP	<code>\$_GET["name"]</code>	<code>\$_POST["name"]</code>
Servlet / JSP	<code>request.getParameter("name")</code>	<code>request.getParameter("name")</code>
ASP.NET	<code>Request.QueryString["name"]</code>	<code>Request.Form["name"]</code>
Multi-valued (checkbox)	<code>request.getParameterValues("i")</code>	<code>\$_POST["i"] (name="i[]")</code>

Reading and validating form data

```
<form action="process.php" method="post">
  Name: <input type="text" name="name">
  Email: <input type="text" name="email">
  Age: <input type="text" name="age">
  <input type="submit" value="Send">
</form>

<?php // process.php
$errors = [];
$name = trim($_POST["name"] ?? "");
$email = trim($_POST["email"] ?? "");
$age = $_POST["age"] ?? "";

if ($name === "") $errors[] = "Name is required";
if (!filter_var($email, FILTER_VALIDATE_EMAIL)) $errors[] = "Invalid email";
if (!ctype_digit($age) || $age < 1 || $age > 120) $errors[] = "Invalid age";

if ($errors) {
    foreach ($errors as $e) echo "<p style='color:red'>" . htmlspecialchars($e)
    . "</p>";
} else {
```

```
    echo "Thank you, " . htmlspecialchars($name) . "!"; // escape output
}
?>
```

- **Client-side validation** (JavaScript, HTML5 required/pattern) gives quick feedback but can be bypassed.
- **Server-side validation** is mandatory: check required fields, data type, length, range and format.
- **Escape output** (htmlspecialchars / c:out) to prevent cross-site scripting (XSS).
- Use **POST** for changing data or sensitive data; use **GET** for searches that can be bookmarked.
- Use **prepared statements** when saving form data to the database.
- After a successful POST, **redirect** (Post/Redirect/Get) so a page refresh does not resubmit the form.

Example 5.2 — Login flow

List the steps that combine form handling, database connectivity, sessions and cookies to build a login system.

Solution

1. Login form posts user name and password over HTTPS. 2. The server validates that fields are not empty. 3. A prepared SELECT fetches the stored password hash for that user. 4. The submitted password is verified against the hash. 5. On success, a session is started (session id is sent in a cookie) and the user is stored in the session; on failure an error is shown. 6. Protected pages check the session and redirect to login if absent. 7. Logout invalidates the session and deletes the cookie.

Key Takeaways — Unit 5

- ✓ Database connectivity: load driver → connect → create statement → execute → process result set → close.
- ✓ SELECT, INSERT, UPDATE and DELETE are the core SQL statements; use prepared statements to stop SQL injection.
- ✓ Authentication methods: anonymous, IP/domain based, basic/digest, integrated Windows, and form-based.
- ✓ Cookies store small pieces of data in the browser (session vs persistent); sessions keep data on the server.
- ✓ File handling uses open–read/write–close with modes r, w, a; uploads need multipart/form-data and careful validation.
- ✓ Form handling: read GET/POST parameters, validate on the server, escape output, and respond or redirect.

— *End of Unit 5* —