

Session 1

Basics of Servlet

A servlet is a Java class that answers HTTP requests inside a container such as Tomcat. This session covers the request and response objects, HTML forms, client information, session tracking and a first database-backed CRUD application.

Objectives

✕ Do not copy. Read for understanding and the viva

- Complete questions 1 to 5 of the manual: basics of servlet
- Prepare the deliverable before the lab and finish it during the session
- Be ready to explain every step in the viva

Questions Covered

✕ Do not copy. Read for understanding and the viva

Question	Requirement	Status
Q1	Write a Servlet Programme to print the current date and time along with the timestamp	Complete
Q2	Create an HTML form with the input of student information using HTTP Protocol and...	Complete
Q3	Write a servlet program to capture client IPs and display it	Complete
Q4	Write a servlet program for session management using HTTP Session along with tracking...	Complete
Q5	Write a CRUD (Create/Save, Read, Edit/Update, Delete) application using servlet...	Complete

Preparation

✗ Do not copy. Read for understanding and the viva

- Create a Dynamic Web Project (Eclipse) or Java Web application (NetBeans) targeting Tomcat 10; note that Tomcat 10 uses the `jakarta.servlet` package, not `javax.servlet`.
- Design the IGNOU database and the Student table on paper first: enrolment number, name, date of birth, email, mobile, address, programme, courses. Question 5 and most later sessions reuse it.
- Add the MySQL Connector/J jar to the project's library path.
- One Maven web project called `ServletLab` holds all five answers. Every servlet is mapped with `@WebServlet`, so `web.xml` only carries the welcome file and the session timeout. All Java files live in `src/main/java/ignou/`, static pages in `src/main/webapp/`.

Project Setup

✗ Do not copy. Read for understanding and the viva

Do this once; every question below drops files into the same project.

1. NetBeans: File, New Project, Java with Maven, Web Application, Next. Project Name `ServletLab`, Next. Server: Apache Tomcat 10.1 (click Add if it is not listed and browse to the extracted Tomcat folder, as in manual figure 2.24), Java EE Version: Jakarta EE 10 Web, Finish.
2. Eclipse: File, New, Dynamic Web Project, name `ServletLab`, Target runtime: Apache Tomcat v10.1, Dynamic web module version 6.0, Finish. Then right-click the project, Configure, Convert to Maven Project.
3. Replace the generated `pom.xml` dependencies with the fragment below and save; the IDE downloads the jars.
4. Put `web.xml` in `src/main/webapp/WEB-INF/`.
5. Run: right-click the project, Run (NetBeans) or Run As, Run on Server (Eclipse). The base URL is `http://localhost:8080/ServletLab/`.

■ Lab record: every tab is one file of the answer. Write all of them.

pom.xml

pom.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- Dependency fragment for a Maven WAR project on Tomcat 10.1 / JDK 17.
3     NetBeans: File > New Project > Java with Maven > Web Application create
4     the rest of this file; paste the <dependencies> block in.
5     Eclipse: right-click the Dynamic Web Project > Configure > Convert to
6     Maven Project, then paste the same block. -->
7 <project xmlns="http://maven.apache.org/POM/4.0.0"
8     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
10 <modelVersion>4.0.0</modelVersion>
11 <groupId>ignou</groupId>
12 <artifactId>ServletLab</artifactId>
13 <version>1.0</version>
14 <packaging>war</packaging>
15
16 <properties>
17     <maven.compiler.source>17</maven.compiler.source>
18     <maven.compiler.target>17</maven.compiler.target>
19     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
20     <failOnMissingWebXml>false</failOnMissingWebXml>
21 </properties>
22
23 <dependencies>
24     <!-- Servlet 6.0 API: Tomcat 10.1 already ships it, so scope is provided
25     <dependency>
26         <groupId>jakarta.servlet</groupId>
27         <artifactId>jakarta.servlet-api</artifactId>
28         <version>6.0.0</version>
29         <scope>provided</scope>
30     </dependency>
31     <!-- JSTL 3.0 (Session 2): API plus the Glassfish implementation, packed
32     <dependency>
33         <groupId>jakarta.servlet.jsp.jstl</groupId>
34         <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
35         <version>3.0.0</version>
36     </dependency>
37     <dependency>
38         <groupId>org.glassfish.web</groupId>
39         <artifactId>jakarta.servlet.jsp.jstl</artifactId>
40         <version>3.0.1</version>
41     </dependency>
42     <!-- MySQL JDBC driver, packed in the WAR -->
43     <dependency>
```

```
44     <groupId>com.mysql</groupId>
45     <artifactId>mysql-connector-j</artifactId>
46     <version>8.4.0</version>
47   </dependency>
48 </dependencies>
49
50 <build>
51   <finalName>ServletLab</finalName>
52 </build>
53 </project>
```

Question 1

Problem Statement

■ Write in lab record

Write a Servlet Programme to print the current date and time along with the timestamp.

Solution

■ Write in lab record

Steps

1. Right-click `Source Packages`, New, Servlet (NetBeans) or New, Servlet (Eclipse). Class name `DateTimeServlet`, package `ignou`. Untick "Add information to deployment descriptor" so the annotation does the mapping.
2. Replace the generated body with the listing.
3. Run the project and open `http://localhost:8080/ServletLab/datetime`.
4. Press F5 a few times: time and timestamp change, the date does not.

Program

DateTimeServlet.java

```
1 package ignou;
2
3 import jakarta.servlet.ServletException;
4 import jakarta.servlet.annotation.WebServlet;
5 import jakarta.servlet.http.HttpServlet;
6 import jakarta.servlet.http.HttpServletRequest;
7 import jakarta.servlet.http.HttpServletResponse;
8
9 import java.io.IOException;
10 import java.io.PrintWriter;
11 import java.sql.Timestamp;
12 import java.time.LocalDate;
13 import java.time.format.DateTimeFormatter;
14 import java.util.Date;
15
16 /** Q1: current date, time and timestamp. URL: /datetime */
17 @WebServlet("/datetime")
18 public class DateTimeServlet extends HttpServlet {
19
20     @Override
21     protected void doGet(HttpServletRequest request, HttpServletResponse response)
22         throws ServletException, IOException {
23         response.setContentType("text/html;charset=UTF-8");
24
25         long millis = System.currentTimeMillis();
26         LocalDate now = LocalDate.now();
27         DateTimeFormatter dateFmt = DateTimeFormatter.ofPattern("dd-MM-yyyy");
28         DateTimeFormatter timeFmt = DateTimeFormatter.ofPattern("HH:mm:ss");
29
30         try (PrintWriter out = response.getWriter()) {
31             out.println("<!DOCTYPE html><html><head><title>Date and Time</title>");
32             out.println("<h2>Current Date and Time</h2>");
33             out.println("<p>Date: " + now.format(dateFmt) + "</p>");
34             out.println("<p>Time: " + now.format(timeFmt) + "</p>");
35             out.println("<p>Full (java.util.Date): " + new Date(millis) + "</p>");
36             out.println("<p>Timestamp (milliseconds since 1 Jan 1970 UTC): " +
37                 new Timestamp(millis) + "</p>");
38             out.println("</body></html>");
39         }
40     }
41 }
```

Output

Expected browser page (values for the moment of the request; checked by reading the listing, not executed, since no Tomcat is available here):

```
1 Current Date and Time
2 Date: 26-09-2026
3 Time: 10:42:07
4 Full (java.util.Date): Sat Sep 26 10:42:07 IST 2026
5 Timestamp (milliseconds since 1 Jan 1970 UTC): 1790397127342
6 SQL Timestamp: 2026-09-26 10:42:07.342
```

Explanation

- `@WebServlet("/datetime")` registers the servlet with the container; no `servlet` and `servlet-mapping` entries are needed in `web.xml`. The manual's examples use `web.xml`; both work on Tomcat 10.
- `doGet` runs once per GET request. `response.setContentType` must come before `getWriter()` or the charset is ignored.
- `System.currentTimeMillis()` is the timestamp: milliseconds since the Unix epoch. `LocalDateTime` and `DateTimeFormatter` (java.time) format the same instant as a readable date and time; `java.sql.Timestamp` is the form a database column would store.
- The `try` with resources closes the `PrintWriter`, which flushes the buffer to the client.

Question 2

Problem Statement

■ Write in lab record

Create an HTML form with the input of student information using HTTP Protocol and method, then display the input information using Servlet.

Solution

■ Write in lab record

Steps

1. Add `student-form.html` under `src/main/webapp/` (NetBeans: right-click Web Pages, New, HTML File).
2. Add servlet `StudentInfoServlet` in package `ignou`.
3. Run and open `http://localhost:8080/ServletLab/student-form.html` (it is also the welcome file, so the bare context URL works).
4. Fill the form and press Submit. Then change `method="post"` to `method="get"` in the HTML, redeploy, submit again and compare the address bar.

Program

- Lab record: every tab is one file of the answer. Write all of them.

student-form.html

student-form.html

```
1 <!DOCTYPE html>
2 <!-- src/main/webapp/student-form.html : Q2 input form.
3     method="post" sends the fields in the request body over HTTP;
4     change to method="get" once to see them appear in the URL instead. -->
5 <html lang="en">
6 <head>
7   <meta charset="UTF-8">
8   <title>Student Information Form</title>
9 </head>
10 <body>
11   <h2>Student Information</h2>
12   <form action="studentInfo" method="post">
13     <p><label>Enrolment No: <input type="text" name="enrolmentNo" required></label></p>
14     <p><label>Name: <input type="text" name="name" required></label></p>
15     <p><label>Date of Birth: <input type="date" name="dob"></label></p>
16     <p><label>Email: <input type="email" name="email"></label></p>
17     <p><label>Mobile: <input type="tel" name="mobile" pattern="[0-9]{10}"></label></p>
18     <p><label>Programme:
19       <select name="programme">
20         <option>MCA</option>
21         <option>BCA</option>
22         <option>MSc</option>
23       </select></label></p>
24     <p>Courses:
25       <label><input type="checkbox" name="courses" value="MCS-218"> MCS-218</label>
26       <label><input type="checkbox" name="courses" value="MCS-219"> MCS-219</label>
27       <label><input type="checkbox" name="courses" value="MCS-220"> MCS-220</label>
28       <label><input type="checkbox" name="courses" value="MCS-221"> MCS-221</label>
29     </p>
30     <p><button type="submit">Submit</button> <button type="reset">Reset</button>
31   </form>
32 </body>
33 </html>
```

Output

Checked by reading, not executed. With POST the address bar shows `/ServletLab/studentInfo` and the page reads:

```
1 Submitted Student Information
2 HTTP method: POST, protocol: HTTP/1.1, content type: application/x-www-form-
3 Enrolment No    2451001234
4 Name            Asha Verma
5 Date of Birth   2001-03-14
6 Email           asha.verma@example.com
7 Mobile          9876543210
8 Programme       MCA
9 Courses         MCS-218, MCS-220
10 Back to form
```

With GET the first line becomes `HTTP method: GET, protocol: HTTP/1.1, content type: null` and the address bar carries every field: `studentInfo?enrolmentNo=2451001234&name=Asha+Verma&...`.

Explanation

- The form `action="studentInfo"` is relative, so the browser resolves it against `/ServletLab/`; the servlet is mapped to `/studentInfo`.
- `request.getParameter(name)` returns one value; `getParameterValues("courses")` returns all ticked checkboxes as an array, or `null` when none is ticked.
- `request.getMethod()` and `getProtocol()` show the HTTP method and version, which is what the question means by "HTTP Protocol and method". GET puts the fields in the query string (visible, bookmarkable, length-limited); POST puts them in the body, so passwords and long text belong in POST.
- `esc()` HTML-escapes the values before they are written back. Without it a name like `<script>` would run in the browser.
- `setCharacterEncoding("UTF-8")` before the first `getParameter` call makes Hindi or other non-ASCII names decode correctly.

Question 3

Problem Statement

■ Write in lab record

Write a servlet program to capture client IPs and display it.

Solution

■ Write in lab record

Steps

1. Add servlet `ClientIpServlet` in package `ignou` .
2. Run and open `http://localhost:8080/ServletLab/clientIp` from the same machine, then `http://127.0.0.1:8080/ServletLab/clientIp` , then from a phone on the same Wi-Fi using the PC's LAN address (find it with `ipconfig` or `ip addr`).

Program

ClientIpServlet.java

```
1  package ignou;
2
3  import jakarta.servlet.ServletException;
4  import jakarta.servlet.annotation.WebServlet;
5  import jakarta.servlet.http.HttpServlet;
6  import jakarta.servlet.http.HttpServletRequest;
7  import jakarta.servlet.http.HttpServletResponse;
8
9  import java.io.IOException;
10 import java.io.PrintWriter;
11
12 /** Q3: shows the client's IP address and related request details. URL: /cli
13 @WebServlet("/clientIp")
14 public class ClientIpServlet extends HttpServlet {
15
16     @Override
17     protected void doGet(HttpServletRequest request, HttpServletResponse res
18         throws ServletException, IOException {
19         response.setContentType("text/html;charset=UTF-8");
20
21         // Behind a proxy or load balancer the real client IP arrives in thi
22         // on a direct localhost connection it is null.
23         String forwarded = request.getHeader("X-Forwarded-For");
24         String clientIp = forwarded != null ? forwarded.split(",")[0].trim()
25
26         try (PrintWriter out = response.getWriter()) {
27             out.println("<!DOCTYPE html><html><head><title>Client IP</title>
28             out.println("<h2>Client Information</h2>");
29             out.println("<p>Client IP address: <b>" + clientIp + "</b></p>")
30             out.println("<p>request.getRemoteAddr(): " + request.getRemoteAd
31             out.println("<p>request.getRemoteHost(): " + request.getRemoteHo
32             out.println("<p>request.getRemotePort(): " + request.getRemotePo
33             out.println("<p>X-Forwarded-For header: " + forwarded + "</p>");
34             out.println("<p>Server name and port: " + request.getServerName(
35             out.println("<p>User-Agent: " + request.getHeader("User-Agent")
36             out.println("</body></html>");
37         }
38     }
39 }
```

Output

Checked by reading. From the same PC using `localhost` :

```
1 Client Information
2 Client IP address: 0:0:0:0:0:0:0:1
3 request.getRemoteAddr(): 0:0:0:0:0:0:0:1
4 request.getRemoteHost(): 0:0:0:0:0:0:0:1
5 request.getRemotePort(): 53412
6 X-Forwarded-For header: null
7 Server name and port: localhost:8080
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) ... Chrome/129.0
```

Using `127.0.0.1` in the URL the first two lines show `127.0.0.1` ; from a phone they show something like `192.168.1.7` .

Explanation

- `getRemoteAddr()` is the address of the TCP peer that connected to Tomcat. On localhost that is the IPv6 loopback `0:0:0:0:0:0:0:1` on most Windows builds, because the browser prefers IPv6 for `localhost` .
- When a reverse proxy or load balancer sits in front of Tomcat, the peer is the proxy, and the original client is in the `X-Forwarded-For` header (a comma list, first entry is the client). The servlet prefers that header when present. Trust it only when you control the proxy; anyone can send that header.
- `getRemoteHost()` returns a host name only if Tomcat's `enableLookups` is on; by default it just repeats the IP.
- `getRemotePort()` is the client's ephemeral port, different for every connection.

Question 4

Problem Statement

■ Write in lab record

Write a servlet program for session management using HTTP Session along with tracking and also use a cookie for session tracking.

Solution

■ Write in lab record

Steps

1. Add servlet `SessionTrackingServlet` in package `ignou`.
2. Run and open `http://localhost:8080/ServletLab/session`. Reload three times and watch "Visits in this session" count up.
3. Type a name, click "Remember me". The name appears in the heading and in the `visitorName` row.
4. Open the browser dev tools, Application, Cookies: you see `JSESSIONID` (session cookie, no expiry) and `visitorName` (expires in 7 days).
5. Click Logout: the visit count restarts at 1, the name is gone, and the Session ID is new.
6. Block cookies for `localhost` in the browser and reload twice: "Session id came from cookie?" turns false, and the "Reload" link now contains `;jsessionid=...` because `encodeURL` switched to URL rewriting.

Program

SessionTrackingServlet.java

```
1 package ignou;
2
3 import jakarta.servlet.ServletException;
4 import jakarta.servlet.annotation.WebServlet;
5 import jakarta.servlet.http.Cookie;
6 import jakarta.servlet.http.HttpServlet;
7 import jakarta.servlet.http.HttpServletRequest;
8 import jakarta.servlet.http.HttpServletResponse;
9 import jakarta.servlet.http.HttpSession;
10
11 import java.io.IOException;
12 import java.io.PrintWriter;
13 import java.util.Date;
14
15 /**
16  * Q4: session management with HttpSession plus a cookie that survives the s
17  * URL: /session          shows counters and session details
18  * URL: /session?action=logout  invalidates the session and deletes the cook
19  */
20 @WebServlet("/session")
21 public class SessionTrackingServlet extends HttpServlet {
22
23     private static final String NAME_COOKIE = "visitorName";
24
25     @Override
26     protected void doGet(HttpServletRequest request, HttpServletResponse res
27         throws ServletException, IOException {
28         if ("logout".equals(request.getParameter("action"))) {
29             HttpSession old = request.getSession(false);
30             if (old != null) old.invalidate();
31             Cookie gone = new Cookie(NAME_COOKIE, "");
32             gone.setMaxAge(0); // max age 0 tells the
33             gone.setPath(request.getContextPath());
34             response.addCookie(gone);
35             response.sendRedirect(request.getContextPath() + "/session");
36             return;
37         }
38
39         HttpSession session = request.getSession(); // creates one on the f
40         Integer visits = (Integer) session.getAttribute("visits");
41         visits = visits == null ? 1 : visits + 1;
42         session.setAttribute("visits", visits);
43
```

```
44     String name = readCookie(request, NAME_COOKIE);
45
46     response.setContentType("text/html;charset=UTF-8");
47     try (PrintWriter out = response.getWriter()) {
48         out.println("<!DOCTYPE html><html><head><title>Session Tracking<
49         out.println("<h2>Welcome " + (name == null ? "guest" : esc(name)
50         out.println("<table border='1' cellpadding='4'>");
51         out.println("<tr><th align='left'>Session ID</th><td>" + session
52         out.println("<tr><th align='left'>New session?</th><td>" + sessi
53         out.println("<tr><th align='left'>Created</th><td>" + new Date(s
54         out.println("<tr><th align='left'>Last accessed</th><td>" + new
55         out.println("<tr><th align='left'>Timeout (s)</th><td>" + sessio
56         out.println("<tr><th align='left'>Visits in this session</th><td>
57         out.println("<tr><th align='left'>Session id came from cookie?</
58             + request.isRequestedSessionIdFromCookie() + "</td></tr>
59         out.println("<tr><th align='left'>Session id came from URL?</th>
60             + request.isRequestedSessionIdFromURL() + "</td></tr>");
61         out.println("<tr><th align='left'>visitorName cookie</th><td>" +
62         out.println("</table>");
63
64         out.println("<form method='post'><p>Your name: <input name='name
65             + "<button type='submit'>Remember me (cookie, 7 days)</b
66         // encodeURL appends ;jsessionid=... only when the browser refus
67         out.println("<p><a href='" + response.encodeURL("session") + "'>
68             + "<a href='session?action=logout'>Logout</a></p>");
69         out.println("</body></html>");
70     }
71 }
72
73 @Override
74 protected void doPost(HttpServletRequest request, HttpServletResponse re
75     throws ServletException, IOException {
76     request.setCharacterEncoding("UTF-8");
77     String name = request.getParameter("name");
78     Cookie c = new Cookie(NAME_COOKIE, java.net.URLEncoder.encode(name,
79     c.setMaxAge(7 * 24 * 60 * 60);
80     c.setPath(request.getContextPath());
81     c.setHttpOnly(true);
82     response.addCookie(c);
83     request.getSession().setAttribute("name", name);
84     response.sendRedirect(request.getContextPath() + "/session");
85 }
86
87 private static String readCookie(HttpServletRequest request, String key)
```

```
88     Cookie[] cookies = request.getCookies();
89     if (cookies == null) return null;
90     for (Cookie c : cookies) {
91         if (key.equals(c.getName())) return java.net.URLDecoder.decode(c
92     }
93     return null;
94 }
95
96 private static String esc(String s) {
97     if (s == null) return "";
98     return s.replace("&", "&amp;").replace("<", "&lt;").replace(">", "&g
99 }
100
101 }
```

Output

Checked by reading. After two reloads and "Remember me" with the name Asha:

```
1 Welcome Asha
2 Session ID                    5F3A0C1E9B7D4E2A8C6F1B0D9E8A7C5B
3 New session?                  false
4 Created                       Sat Sep 26 10:50:02 IST 2026
5 Last accessed                 Sat Sep 26 10:50:41 IST 2026
6 Timeout (s)                   1800
7 Visits in this session        3
8 Session id came from cookie?  true
9 Session id came from URL?     false
10 visitorName cookie            Asha
11 Your name: [                  ] [Remember me (cookie, 7 days)]
12 Reload (URL rewriting aware) | Logout
```

Explanation

- HTTP is stateless. `request.getSession()` makes Tomcat create an `HttpSession` object on the server and send its id to the browser in the `JSESSIONID` cookie; every later request carries the cookie, so Tomcat finds the same object. The `visits` attribute lives in that object, not in the browser.
- The session cookie has no expiry, so it dies when the browser closes; the session itself dies after 30 minutes of inactivity (`session-timeout` in `web.xml` , shown as 1800 seconds).

- The `visitorName` cookie is the second tracking mechanism: it is stored by the browser with `setMaxAge(7 days)`, so it survives browser restarts and even `session.invalidate()`. Logout deletes it by sending the same cookie with max age 0.
- `response.encodeURL()` is the fallback when cookies are refused: it appends `;jsessionid=` to links so the id travels in the URL. `isRequestedSessionIdFromCookie()` and `isRequestedSessionIdFromURL()` show which route was used.
- Cookie values may not contain spaces or semicolons, so the name is URL-encoded before storing and decoded when read. `setHttpOnly(true)` keeps JavaScript from reading it.

Question 5

Problem Statement

■ Write in lab record

Write a CRUD (Create/Save, Read, Edit/Update, Delete) application using servlet. Create a Database named IGNOU, create a table named Student which must capture the student information (basics, contact, enrollment details along with courses). Make necessary assumptions required.

Solution

■ Write in lab record

Assumptions

- Enrolment number is the primary key (IGNOU issues a unique 9 to 12 digit number).
- Courses are stored in one column as a comma-separated list of course codes. A student registers for a handful of courses, so a separate Course table and join table would add two more screens without teaching anything new for this session. Session 6 (Hibernate) is the place to normalise it.
- One application login `ignou / ignou123` with only SELECT, INSERT, UPDATE and DELETE rights on the IGNOU database.
- MySQL 8 runs on `localhost:3306`.

Steps

1. Start MySQL and run `schema.sql` in MySQL Workbench (File, Open SQL Script, then the lightning-bolt Execute button) or with `mysql -u root -p` then `source schema.sql`. Check with `SELECT * FROM Student;` that three rows exist.
2. Add `Student.java`, `StudentDao.java` and `StudentServlet.java` to package `ignou`. The `mysql-connector-j` dependency is already in `pom.xml`.
3. Run and open `http://localhost:8080/ServletLab/students`.
4. Read: the list shows the three seeded rows. Create: click "Add new student", fill the form, Save. Update: click Edit on a row, change the mobile, Save. Delete: click Delete, confirm.
5. Verify each step in Workbench with `SELECT enrolment_no, name, mobile FROM Student;`.

Program

- Lab record: every tab is one file of the answer. Write all of them.

schema.sql

schema.sql

```

1  -- Q5: IGNOU database and Student table.
2  -- Run in MySQL Workbench or:  mysql -u root -p < schema.sql
3  -- One table holds basics, contact and enrolment details; the courses column
4  -- comma-separated list of course codes (assumption: a student registers for
5  -- a handful of courses, so a separate Course table is not needed for this 1
6
7  CREATE DATABASE IF NOT EXISTS IGNOU CHARACTER SET utf8mb4;
8  USE IGNOU;
9
10 DROP TABLE IF EXISTS Student;
11
12 CREATE TABLE Student (
13     -- basics
14     enrolment_no    VARCHAR(12) NOT NULL,
15     name            VARCHAR(80) NOT NULL,
16     dob            DATE          NOT NULL,
17     gender          ENUM('M', 'F', 'O') NOT NULL DEFAULT 'M',
18     -- contact
19     email           VARCHAR(120) NOT NULL,
20     mobile          CHAR(10)     NOT NULL,
21     address         VARCHAR(200),
22     city            VARCHAR(60),
23     state           VARCHAR(60),
24     pincode         CHAR(6),
25     -- enrolment details
26     programme       VARCHAR(10) NOT NULL,           -- MCA, BCA, MSc ...
27     semester        TINYINT     NOT NULL DEFAULT 1,
28     admission_year  YEAR         NOT NULL,
29     study_centre    VARCHAR(10),                    -- e.g. 0710
30     courses         VARCHAR(200),                    -- 'MCS-218,MCS-219,MCS-
31     PRIMARY KEY (enrolment_no),
32     UNIQUE KEY uq_student_email (email),
33     CONSTRAINT chk_mobile CHECK (mobile REGEXP '^[0-9]{10}$'),
34     CONSTRAINT chk_semester CHECK (semester BETWEEN 1 AND 6)
35 );
36
37 -- Application login used by the servlet: create once, grant only what the 1
38 CREATE USER IF NOT EXISTS 'ignou'@'localhost' IDENTIFIED BY 'ignou123';
39 GRANT SELECT, INSERT, UPDATE, DELETE ON IGNOU.* TO 'ignou'@'localhost';
40
41 INSERT INTO Student (enrolment_no, name, dob, gender, email, mobile, address
42                     programme, semester, admission_year, study_centre, cour
43 ('2451001234', 'Asha Verma', '2001-03-14', 'F', 'asha.verma@example.com',

```

```
44 ('2451001235', 'Rahul Singh', '2000-11-02', 'M', 'rahul.singh@example.com',  
45 ('2451001236', 'Meera Nair', '2002-07-25', 'F', 'meera.nair@example.com',  
46  
47 SELECT enrolment_no, name, programme, semester, courses FROM Student;
```

Output

Checked by reading; the SQL was checked against MySQL 8 syntax. After adding a fourth student the list page reads:

```
1 IGNOU Students (4)  
2 Student added  
3 Add new student  
4 Enrolment      Name      DOB      Email      Mobile  
5 2451001234     Asha Verma 2001-03-14 asha.verma@example.com 9876543210  
6 2451001235     Rahul Singh 2000-11-02 rahul.singh@example.com 9123456780  
7 2451001236     Meera Nair 2002-07-25 meera.nair@example.com 9988776655  
8 2451001237     Vikram Rao 2001-08-30 vikram.rao@example.com 9012345678
```

Submitting the add form a second time with the same enrolment number shows, in green text at the top of the list, `Database error: Duplicate entry '2451001237' for key 'student.PRIMARY'`. After Delete the count drops to 3 and the message reads `Student deleted`.

Explanation

- Three layers: `Student` (data), `StudentDao` (SQL), `StudentServlet` (HTTP and HTML). The servlet never builds SQL; the DAO never touches the request. That is the split every later session (Spring, Hibernate) formalises.
- `PreparedStatement` sends the SQL text with `?` placeholders first and the values separately, so a name containing a quote cannot change the statement (SQL injection). `setString`, `setInt` also handle quoting and type conversion.
- Every DAO method opens and closes its own connection through `try` with resources. That is fine for a lab; a real application uses a connection pool (Tomcat's JNDI `DataSource`).
- The servlet is a front controller: the `action` parameter selects list, new, edit, insert, update or delete. Reads are GET; writes are POST, and every POST ends in `sendRedirect` back to the list (Post-Redirect-Get), so pressing F5 on the list page never re-inserts a row.
- Delete is a tiny POST form rather than a link, because browsers and crawlers prefetch links; a GET that deletes data is a classic bug.

- The `CHECK` constraints in `schema.sql` catch bad mobile numbers and semesters even if someone bypasses the HTML `pattern` attributes; the database is the last line of validation.

Viva Questions

✗ Do not copy. Read for understanding and the viva

- **Q:** What is the servlet life cycle? **A:** The container loads the class, calls `init()` once, calls `service()` (which dispatches to `doGet`, `doPost`) for every request on a pooled thread, and calls `destroy()` once at undeploy.
- **Q:** Why `jakarta.servlet` and not `javax.servlet`? **A:** Tomcat 10 implements Jakarta EE 9+, where every package was renamed from `javax.*` to `jakarta.*`. Code with `javax.servlet` imports compiles but Tomcat 10 never calls it.
- **Q:** Difference between `@WebServlet` and `web.xml` mapping? **A:** Same effect; the annotation keeps the mapping next to the code, `web.xml` lets you change it without recompiling and is needed for container-wide settings like session timeout and error pages.
- **Q:** GET versus POST? **A:** GET carries parameters in the URL, is idempotent and cacheable; POST carries them in the body and is used for anything that changes state.
- **Q:** How does Tomcat know which session belongs to which browser? **A:** By the `JSESSIONID` cookie, or by `;jsessionid=` in the URL when cookies are off.
- **Q:** Where is session data stored, browser or server? **A:** On the server, in the `HttpSession` object; the browser only holds the id.
- **Q:** Why `PreparedStatement` instead of `Statement`? **A:** Parameters are bound, not concatenated, so user input cannot alter the SQL; the driver can also cache the compiled statement.
- **Q:** What does Post-Redirect-Get solve? **A:** A browser refresh after a POST re-sends the form; redirecting to a GET page after the write prevents duplicate inserts.

Common Mistakes

✗ Do not copy. Read for understanding and the viva

- Importing `javax.servlet.*`: the project compiles against an old jar but Tomcat 10 returns 404 for the servlet.
- Calling `getWriter()` before `setContentType()`: the charset header is ignored and non-ASCII text appears garbled.

- Forgetting `mysql-connector-j` in `WEB-INF/lib` (or `pom.xml`): No suitable driver found for `jdbc:mysql:// ...` at run time.
- Leaving the MySQL server stopped or the `IGNOU` schema uncreated: `Communications link failure` or `Unknown database 'ignou'`.
- Deleting through a GET link, or forgetting to redirect after POST, so a refresh repeats the write.
- Writing user input straight into HTML without escaping, which lets a `<script>` in the name field run in every viewer's browser.

Session Summary

■ Write in lab record

- Project `ServletLab` on Tomcat 10.1 with `pom.xml` (`jakarta.servlet-api`, `JSTL`, `mysql-connector-j`) and `web.xml`
- Question 1: `DateTimeServlet` at `/datetime` printing date, time and millisecond timestamp
- Question 2: `student-form.html` posting to `StudentInfoServlet` at `/studentInfo`, showing method and protocol
- Question 3: `ClientIpServlet` at `/clientIp` with `getRemoteAddr` and `X-Forwarded-For`
- Question 4: `SessionTrackingServlet` at `/session` with `HttpSession` visit counter, `visitorName` cookie, logout and URL rewriting
- Question 5: `schema.sql` (`IGNOU` database, `Student` table), `Student`, `StudentDao` with `PreparedStatement`, `StudentServlet` CRUD at `/students`

 Edit this page

 [Previous](#)
Section 2 · Web Technologies

[Next](#) 
Session 2