



FACULTY OF ENGINEERING AND TECHNOLOGY BACHELOR OF TECHNOLOGY

Enterprise Programming using Java Laboratory (303105310)

5th SEMESTER

# Computer Science & Engineering Department

Laboratory Manual Session 2025-26

A.Y 2025-26

**Submitted By :** **Submitted to :**

Name : Ajay Duddi

Enroll. No. :

Division :

Roll No. :

**CERTIFICATE**

# This is to certify that

Mr./Ms......................................................................................................................................with

enrollment no. has successfully completed his/her laboratory

experiments in the **Enterprise Programming using Java Laboratory (303105310)** from the department of **COMPUTER SCIENCE & ENGINEERING** during the academic year 2025-2026.



Date of Submission - . Staff In Charge -

Head Of Department -

**TABLE OF CONTENT**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Sr. No | Experiment Title | Page No | | Date of Start | Date of  Completion | Sign | Marks  (out of 10) |
| From | To |
| 1. | Write a program to insert and retrieve the data from database using JDBC. |  |  |  |  |  |  |
| 2. | Write a program to demonstrate the use of Prepared Statement and Result Set interface. |  |  |  |  |  |  |
| 3. | Servlet Programming :  Servlet Execution on tomcat A servlet program to print hello world.  A servlet program to display request details A servlet program to handle user form.  A servlet program to create a cookie A servlet program to display cookie .  A servlet program to do session tracking .  Write a program to implement chat Server using Server Socket  Write a Servlet program to send username and password using HTML forms and authenticate the user |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 4. | JSP Programming  JSP program to display hello world.  JSP program to demonstrate arithmetic operations.  JSP program to demonstrate jsp: forward action tag JSP program to request implicit object Developing a web application to insert record into Oracle Database using JSP and JDBC |  |  |  |  |  |  |
| 5. | Create application to store the data in database to perform Hibernate CRUD operations. |  |  |  |  |  |  |
| 6. | Create a application store the data in database to perform Spring CRUD operations. |  |  |  |  |  |  |
| 7. | Create a web application to store the data in database with spring boot. |  |  |  |  |  |  |

EXPERIMENT NO.: 1

Aim: Write a program to insert and retrieve the data from database using JDBC.

JDBC (Java Database Connectivity) is an API that allows Java applications to interact with databases. It provides a set of interfaces and classes for sending SQL statements, retrieving results, and managing database connections. The main components of JDBC are:

List of JDBC components:

1. JDBC Drivers – Translate Java calls into DB-specific calls (Type 1–4, with Type 4 most common).
2. DriverManager – Manages drivers and creates DB connections.
3. Connection – Represents a session with the database, used to create statements and manage transactions.
4. Statement – Executes SQL queries.
   * Statement – simple queries
   * PreparedStatement – precompiled, parameterized queries
   * CallableStatement – stored procedures
5. ResultSet – Holds query results, navigates row by row.
6. SQLException – Handles DB errors and exceptions.

Steps:

1. Create a table in your database.

create database epj;

use epj;

create table Amit(

id int primary key auto\_increment,

name varchar(30),

marks int);

select \* from Amit;

2. Open Eclispe (Enterprise Web Developers) and create Java Project(File>New>JavaProject) and create Class.

3. After creating the Class, Configure the MySql Connector jar file(buildpath>configure build path>libraries>classpath>AddExternal Jarsand add the file).

Code:-

package jdbclearn;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

public class learn {

public static void main(String[] args){

try {

Class.*forName*("com.mysql.cj.jdbc.Driver");

Connection con=DriverManager.*getConnection*("jdbc:mysql://localhost:3306/epj","root","");

System.***out***.println("Connected to DB");

Statement stmt=con.createStatement();

String name="Amit";

int marks=80;

String insertquery="Insert into Amit(name,marks) values ('"+name+"', "+marks+")";

int rowsInserted=stmt.executeUpdate(insertquery);

System.***out***.println(rowsInserted+" rows inserted.");

String selectquery="Select \* from Amit";

ResultSet rs=stmt.executeQuery(selectquery);

System.***out***.println("Records from Amit table are:");

while(rs.next()) {

int id=rs.getInt("id");

String studentName=rs.getString("name");

int studentMarks=rs.getInt("marks");

System.***out***.println(id+" | "+studentName+" | "+studentMarks+" | ");

}

rs.close();

stmt.close();

con.close();

}

catch(Exception e) {

e.printStackTrace();

}

}

}

Console Ouput:

A screenshot of a computer program

AI-generated content may be incorrect.

Database output:

A screenshot of a computer

AI-generated content may be incorrect.

**Conclusion:**  
The program successfully demonstrates how to connect to a database using JDBC, insert records, and retrieve data, showing the basic working of JDBC for database operations.

EXPERIMENT NO.: 2

Aim:- Write a program to demonstrate the use of Prepared Statement and Result Set interface.

Prepared statements in MySQL (and other databases) are a way to **pre-compile SQL queries** and then execute them with different values.

Here are **2–3 key points**:

1. **Security (prevents SQL injection)**
   * User input is treated as data, not as part of the SQL query.
   * For example, instead of building a query by string concatenation, placeholders (?) are used.
2. **Performance**
   * The SQL query is parsed and compiled once, then executed multiple times with different parameters.
   * This reduces overhead when running the same query repeatedly.
3. **Cleaner Code**
   * Placeholders (?) make the query easier to read and maintain.
   * Parameters are bound separately, reducing string manipulation.

Code:-

package in;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.Statement;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

try {

Class.*forName*("com.mysql.cj.jdbc.Driver");

Connection con=DriverManager.*getConnection*("jdbc:mysql://localhost:3306/epj","root","");

String sql = "insert into Amit(name,marks) values (?,?)";

PreparedStatement prep = con.prepareStatement(sql);

Scanner in = new Scanner(System.***in***);

System.***out***.println("Enter name : ");

String name = in.nextLine();

System.***out***.println("Enter marks : ");

int marks = in.nextInt();

prep.setString(1,name);

prep.setInt(2,marks);

int x = prep.executeUpdate();

if(x>0) {

System.***out***.println("Record inserted successfully");

}

sql="Select \* from Amit";

Statement stmt = con.createStatement();

ResultSet res=stmt.executeQuery(sql);

while(res.next()) {

System.***out***.println(res.getInt("id")+" "+res.getString("name")+" "+res.getInt("marks"));

}

in.close();

}

catch(Exception e){

e.printStackTrace();

}

}

}

Console Output:

A screenshot of a computer program

AI-generated content may be incorrect.

Database Output:

A screenshot of a computer

AI-generated content may be incorrect.

**Conclusion:**  
The program successfully demonstrates how PreparedStatement provides secure and efficient query execution with dynamic parameters, while the ResultSet interface allows easy retrieval and navigation of query results.

EXPERIMENT NO.: 3

Aim:-Servlet Programming Servlet Execution on tomcat A servlet program to print hello world A servlet program to display request details A servlet program to handle user form A servlet program to create a cookie A servlet program to display cookie A servlet program to do session tracking Write a program to implement chat Server using Server Socket and Socket class. Write a Servlet program to send username and password using HTML forms and authenticate the user.

* Create Dynamic web Project(file>new>other>dwp) and give class name
* Install Apache Tomcat
* configure servlet api jar file in the project(buildpath>configure buildpath>downloads>apache tomacat>lib>servlet api)
* Configure apache tomcat server(run on server > choose version of tomcat in Apache> browse the path and select jre version and run) and start the server.

1. Servlet Execution on Tomcat

* Install Tomcat (e.g., Apache Tomcat 9.0).
* Addservlet-api.jar to your project’s classpath (comes with Tomcat in lib folder).
* Create your Java servlet program inside src folder of a Dynamic Web Project in Eclipse (or manually in Tomcat’s WEB-INF/classes).
* Configureweb.xml OR use @WebServlet annotation.
* Deploy WAR to Tomcat’s webapps folder and start Tomcat.

2. Display Hello World By Running on server.

Code :

package in;

import java.io.IOException;

import java.io.PrintWriter;

import org.apache.catalina.connector.Response;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

public class MyServlet extends HttpServlet{

@Override

protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

PrintWriter outPrintWriter=resp.getWriter();

outPrintWriter.print("Hello Amit");

}

}

Output on Default Browser:-

A screenshot of a computer

AI-generated content may be incorrect.

3. Servlet to Display Request Details

Code :

package in;

import java.io.IOException;

import java.io.PrintWriter;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

public class MyServlet extends HttpServlet{

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.println("<h1>Hello Amit</h1>");

out.println("<h2>Request Details</h2>");

out.println("Method: " + request.getMethod() + "<br>");

out.println("URI: " + request.getRequestURI() + "<br>");

out.println("Protocol: " + request.getProtocol() + "<br>");

out.println("Remote Address: " + request.getRemoteAddr() + "<br>");

out.println("User Agent: " + request.getHeader("User-Agent") + "<br>");

}

}

Output on Default Browser :

A screenshot of a computer

AI-generated content may be incorrect.

1. Servlet to Handle User Form :

index.html

Code :

<!**DOCTYPE** html>

<**html**>

<**head**>

<**meta** charset=*"UTF-8"*>

<**title**>Insert title here</**title**>

</**head**>

<**body**>

<**form** action=*"submitForm"* method=*"post"*>

<**div**>Name: </**div**>

<**input** type=*"text"* name=*"name1"*/><**br**/><**br**/>

<**div**>Email: </**div**>

<**input** type=*"text"* name=*"email1"*/><**br**/><**br**/>

<**input** type=*"submit"* value=*"Click Me"*/>

</**form**>

</**body**>

</**html**>

MyServlet.java :

package in;

import java.io.IOException;

import java.io.PrintWriter;

import jakarta.servlet.ServletException;

import jakarta.servlet.annotation.WebServlet;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/submitForm")

public class MyServlet extends HttpServlet{

@Override

protected void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

String myname=req.getParameter("name1");

String myemail=req.getParameter("email1");

PrintWriter out=resp.getWriter();

out.println("Name : "+myname);

out.println("Email : "+myemail);

}

}

Browser Output :

A screenshot of a computer

AI-generated content may be incorrect.

Output after submitting form :

A screenshot of a computer

AI-generated content may be incorrect.

1. Creation and Display of Cookie :

Code :

package in;

import java.io.IOException;

import jakarta.servlet.ServletException;

import jakarta.servlet.annotation.WebServlet;

import jakarta.servlet.http.Cookie;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/COOKIE")

public class COOKIE extends HttpServlet {

private static final long serialVersionUID = 1L;

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

Cookie cookie = new Cookie("username", "Amit");

Cookie cookie2 = new Cookie("user\_name", "XYZ");

cookie.setMaxAge(60 \* 60);

cookie2.setMaxAge(60\*60);

response.addCookie(cookie);

response.addCookie(cookie2);

response.setContentType("text/html");

response.getWriter().println("<h1>Cookie has been created successfully!</h1>");

response.getWriter().println("<p>Check your browser's developer tools to see it.</p>");

Cookie[] cookies = request.getCookies(); // Get cookies sent by browser

if (cookies != null) {

response.getWriter().println("<h2>Cookies available:</h2>");

for (Cookie c : cookies) {

response.getWriter().println("<p>" + c.getName() + " = " + c.getValue() + "</p>");

}

} else {

response.getWriter().println("<p>No cookies found.</p>");

}

}

}

Browser Output :

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Session tracking Servlet :

Code :

package in;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.Date;

import jakarta.servlet.ServletException;

import jakarta.servlet.annotation.WebServlet;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

import jakarta.servlet.http.HttpSession;

@WebServlet("/SessionTrackerServlet")

public class SessionTrackerServlet extends HttpServlet {

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

HttpSession session = request.getSession(true);

Integer visitCount = (Integer) session.getAttribute("visitCount");

if (visitCount == null) {

visitCount = 1;

} else {

visitCount++;

}

session.setAttribute("visitCount", visitCount);

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.println("<h2>Hello Amit</h2>");

out.println("<h2>Your Session Information:</h2>");

out.println("Session ID: " + session.getId() + "<br>");

out.println("<h2>Visit Count : "+visitCount+"</h2>");

}

}

Browser Output :

A screenshot of a computer

AI-generated content may be incorrect.

1. Chat Server using Server Socket :

ChatServer.java

package com;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

public class ChatServer {

public static void main(String[] args) throws IOException {

ServerSocket serverSocket = new ServerSocket(5000);

System.***out***.println("Chat server started on port 5000...");

Socket socket = serverSocket.accept();

System.***out***.println("Client connected.");

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

BufferedReader console = new BufferedReader(new InputStreamReader(System.***in***));

String msg;

while (true) {

msg = in.readLine();

if (msg == null || msg.equalsIgnoreCase("bye")) {

break;

}

System.***out***.println("Client: " + msg);

System.***out***.print("You: ");

out.println(console.readLine());

}

socket.close();

serverSocket.close();

}

}

ChatClient.java

package com;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.Socket;

public class ChatClient {

public static void main(String[] args) throws IOException {

Socket socket = new Socket("localhost", 5000);

System.***out***.println("Connected to server.");

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

BufferedReader console = new BufferedReader(new InputStreamReader(System.***in***));

String msg;

while (true) {

System.***out***.print("You: ");

out.println(console.readLine());

msg = in.readLine();

if (msg == null || msg.equalsIgnoreCase("bye")) {

break;

}

System.***out***.println("Server: " + msg);

}

// Close the connection

socket.close();

}

}

Server Console :

A screenshot of a computer program

AI-generated content may be incorrect.

Client Console :

A screenshot of a computer

AI-generated content may be incorrect.

1. Login Validation :

login.html

<!**DOCTYPE** html>

<**html**>

<**head**>

<**meta** charset=*"UTF-8"*>

<**title**>Login Page</**title**>

</**head**>

<**body**>

<**h2**>Login Form</**h2**>

<**form** action=*"login"* method=*"post"*>

Username: <**input** type=*"text"* name=*"username"* required><**br**><**br**>

Password: <**input** type=*"password"* name=*"password"* required><**br**><**br**>

<**input** type=*"submit"* value=*"Login"*>

</**form**>

</**body**>

</**html**>

LoginServlet.java

package in;

import java.io.IOException;

import java.io.PrintWriter;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

public class LoginServlet extends HttpServlet {

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

response.setContentType("text/html");

PrintWriter out = response.getWriter();

String user = request.getParameter("username");

String pass = request.getParameter("password");

if ("Amit".equals(user) && "Hepsto\*11".equals(pass)) {

out.println("<h3>Login Successful</h3>");

} else {

out.println("<h3>Invalid Username or Password</h3>");

}

}

}

Web.xml :

<?**xml** version=*"1.0"* encoding=*"UTF-8"*?>

<**web-app** xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"* xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"* version=*"3.1"*>

<**display-name**>LoginApp</**display-name**>

<**servlet**>

<**servlet-name**>LoginServlet</**servlet-name**>

<**servlet-class**>in.LoginServlet</**servlet-class**>

</**servlet**>

<**servlet-mapping**>

<**servlet-name**>LoginServlet</**servlet-name**>

<**url-pattern**>/login</**url-pattern**>

</**servlet-mapping**>

</**web-app**>

Default Browser Output :

If Username=Amit and Password=Hepsto\*11

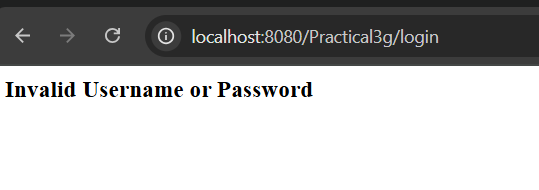
A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

If Username=Amit and Password=aabb



Conclusion:  
Servlet programming on Tomcat shows how Java can handle web requests, manage state with cookies and sessions, and perform user authentication. Along with socket-based chat, it demonstrates building dynamic, interactive, and scalable web applications.

EXPERIMENT NO.: 4

Aim : - JSP Programming JSP program to display hello world. JSP program to demonstrate arithmetic operations JSP program to demonstrate jsp: forward action tag JSP program to request implicit object Developing a web application to insert record into MySQL Database using JSP and JDBC.

**JSP (JavaServer Pages) :**

1. JSP is a **server-side technology** to create dynamic web pages using Java + HTML.
2. Every JSP page is **compiled into a servlet** and executed on a web server like Tomcat.
3. Provides **implicit objects**: request, response, session, application, out, etc.
4. Supports **scripting elements** (<% %>), **expressions** (<%= %>), and **declarations** (<%! %>).
5. Offers **action tags** like <jsp:include> and <jsp:forward> for page navigation and reuse.
6. Can connect to databases via **JDBC**, though DB logic is better handled in servlets/beans.
7. **Advantages**: simple, quick for building dynamic pages, easy integration with Java.
8. **Limitations**: mixing Java with HTML leads to messy code, replaced by modern frameworks (Spring MVC, Thymeleaf, React).

Hello World using JSP :

Hello.jsp :

<%@ **page** language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

<!**DOCTYPE** html>

<**html**>

**head**>

<**meta** charset=*"UTF-8"*>

<**title**>Hello World - JSP</**title**>

</**head**>

<**body**>

<**h1**>Hello Amit!!!</**h1**>

</**body**>

</**html**>

Default Browser Output :

A screenshot of a computer

AI-generated content may be incorrect.

Arithmetic Operations using JSP :

calc.jsp :

<%@ **page** language=*"java"* contentType=*"text/html; charset=UTF-8"*

pageEncoding=*"UTF-8"*%>

<!**DOCTYPE** html>

<**html**>

<**head**>

<**meta** charset=*"UTF-8"*>

<**title**>Calculations</**title**>

</**head**>

<**body**>

<**h1**>Arithmetic Operations in JSP</**h1**>

<%

int a = 20;

int b = 5;

int sum = a + b;

int difference = a - b;

int product = a \* b;

int quotient = a / b;

int mod=a%b;

%>

<**p**>Numbers are <%= a %> and <%= b %></**p**>

<**br**>

<**br**>

<**h2**>Results:</**h2**>

<**div**>Addition (a + b): <%= sum %></**div**>

<**div**>Subtraction (a - b): <%= difference %></**div**>

<**div**>Multiplication (a \* b): <%= product %></**div**>

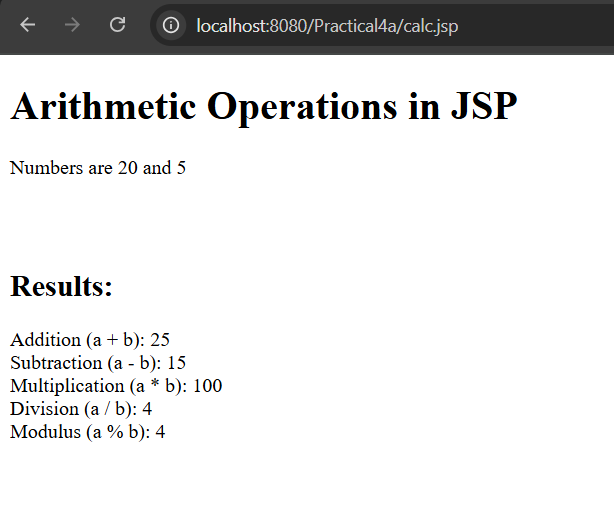
<**div**>Division (a / b): <%= quotient %></**div**>

<**div**>Modulus (a % b): <%= quotient %></**div**>

</**body**>

</**html**>

Default Browser Output :



Web Application to insert data into MySQL Database :

Code :

StudentDao.java

package com.dao;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

import com.pojo.Student;

public class StudentsDao {

private String url = "jdbc:mysql://localhost:3306/epj";

private String username = "root";

private String password = "";

private Connection getConnection() throws SQLException {

try {

Class.forName("com.mysql.cj.jdbc.Driver");

return DriverManager.getConnection(url, username, password);

} catch (ClassNotFoundException e) {

throw new SQLException("MySQL JDBC Driver not found", e);

}

}

public void create(String name, int age) throws SQLException {

String sql = "insert into students(name, age) values(?, ?)";

try (Connection conn = getConnection();

PreparedStatement ptmt = conn.prepareStatement(sql)) {

ptmt.setString(1, name);

ptmt.setInt(2, age);

ptmt.executeUpdate();

}

}

public List<Student> get() throws SQLException {

String sql = "select \* from students";

List<Student> data = new ArrayList<>();

try (Connection conn = getConnection();

PreparedStatement ptmt = conn.prepareStatement(sql);

ResultSet result = ptmt.executeQuery()) {

while(result.next()) {

data.add(new Student(result.getInt("id"), result.getString("name"), result.getInt("age")));

}

}

return data;

}

}

**Student.java**

package com.pojo;

public class Student {

int id;

String name;

int age;

public Student() {

}

public Student(int id, String name, int age) {

super();

this.id = id;

this.name = name;

this.age = age;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

}

**StudentServlet.java**

package com.servlet5a1;

import java.io.IOException;

import java.sql.SQLException;

import java.util.List;

import com.dao.StudentsDao;

import com.pojo.Student;

import jakarta.servlet.RequestDispatcher;

import jakarta.servlet.ServletException;

import jakarta.servlet.annotation.WebServlet;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/students")

class StudentServlet extends HttpServlet{

@Override

protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

String name = req.getParameter("name");

int age = Integer.parseInt(req.getParameter("age"));

StudentsDao dao = new StudentsDao();

try {

dao.create(name, age);

List<Student> data = dao.get();

req.setAttribute("data", data);

RequestDispatcher dispatch = req.getRequestDispatcher("result.jsp");

dispatch.forward(req, resp);

} catch (SQLException e) {

e.printStackTrace();

}

}

}

**student.jsp**

<%@ **page** language=*"java"* contentType=*"text/html"* pageEncoding=*"UTF-8"* %>

<!**DOCTYPE** HTML>

<**html**>

<**head**>

<**title**> Students </**title**>

</**head**>

<**body**>

<**form** action=*"students"* method=*"post"*>

Name : <**input** type=*'text'* name=*"name"* /><**br**/>

Age : <**input** type=*'number'* name=*"age"* /><**br**/>

<**input** type=*"submit"* /><**br**/>

</**form**>

</**body**>

**result.jsp**

<%@ **page** language=*"java"* contentType=*"text/html"* pageEncoding=*"UTF-8"* import=*"java.util.List, com.pojo.Student"* %>

<!**DOCTYPE** HTML>

<**html**>

<**head**>

<**title**> Students </**title**>

</**head**>

<**body**>

<%

List<Student> data = (List<Student>) request.getAttribute("data");

for (int i = 0; i < data.size(); i++) {

%>

<**p**>

id: <%= data.get(i).getId() %><**br**>

name: <%= data.get(i).getName() %><**br**>

age: <%= data.get(i).getAge() %><**br**>

</**p**>

<% } %>

</**body**>

</**html**>

**web.xml**

<**web-app** xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"*

version=*"3.1"*>

<**servlet**>

<**servlet-name**>firstServlet</**servlet-name**>

<**servlet-class**>com.servlet5a1.Myservlet</**servlet-class**>

</**servlet**>

<**servlet-mapping**>

<**servlet-name**>firstServlet</**servlet-name**>

<**url-pattern**>/hello</**url-pattern**>

</**servlet-mapping**>

</**web-app**>

Default Browser Output :

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Conclusion:

JSP facilitates dynamic web development by supporting content generation, arithmetic processing, and seamless database integration with MySQL via JDBC.

EXPERIMENT NO.: 5

Aim : - Create application to store the data in database to perform Hibernate CRUD operations.

**The Hibernate Framework :**

Hibernate is a high-performance, open-source **Object-Relational Mapping (ORM)** framework for the Java language. Its primary purpose is to solve the **"object-relational impedance mismatch"**—the fundamental conflict between the object-oriented model of Java (classes, objects) and the relational model of databases (tables, rows, keys).

Hibernate acts as a "bridge," automatically translating Java objects into database records and vice versa. This automates the low-level JDBC (Java Database Connectivity) code, allowing developers to focus on business logic.

**Core Concepts:**

* **Entity Mapping:** Java classes (POJOs) are mapped to database tables using annotations (e.g., @Entity, @Table, @Id). These annotations define how the object's fields correspond to the table's columns.
* **SessionFactory:** A heavyweight, thread-safe object that is created once per application. It acts as a factory for Session objects and manages the database connection pool and second-level cache.
* **Session:** A lightweight, non-thread-safe object that represents a single "unit of work" with the database. It is the main interface used by the developer to perform CRUD (Create, Read, Update, Delete) operations.
* **HQL (Hibernate Query Language):** An object-oriented query language that is database-independent. It allows developers to write queries based on Java class and property names (e.g., FROM Student WHERE age > 20) instead of SQL table and column names.
* **Caching:** Hibernate uses a First-Level Cache (tied to the Session object, on by default) and an optional Second-Level Cache (tied to the SessionFactory) to store frequently accessed data in memory,

significantly improving application performance by reducing database queries.

**Application to perform Hibernate CRUD Operations :**

EmployeeDao.java

package com.dao;

import java.util.List;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import com.hibernate.Factory;

import com.pojo.Employee;

public class EmployeeDao {

SessionFactory factory = null;

public EmployeeDao() {

4 super();

factory = Factory.getFactory();

}

**// Create**

public void insert(String name,String department) {

Session session= factory.openSession();

try {

Transaction tx=session.beginTransaction();

Employee record=new Employee(name,department);

session.persist(record);

tx.commit();

System.out.println("Data Inserted Successfully");

session.close();

}

catch(Exception e) {

e.printStackTrace();

}

}

**// READ**

public List<Employee> find() {

Session session= factory.openSession();

List<Employee> data=null;

try {

Transaction tx=session.beginTransaction();

data = session.createQuery("from Employee", Employee.class).list();

tx.commit();

System.out.println("Table Data");

session.close();

}

catch(Exception e) {

e.printStackTrace();

}

return data;

}

/**/UPDATE**

public void update(int id, String name, String department) {

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

Employee employee = session.get(Employee.class, id);

if (employee != null) {

employee.setName(name);

employee.setDepartment(department);

session.merge(employee);

System.out.println("Employee with ID " + id + " updated successfully.");

} else {

System.out.println("Employee with ID " + id + " not found.");

}

tx.commit();

session.close();

} catch (Exception e) {

e.printStackTrace();

}

}

**//DELETE**

public void delete(int id) {

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

Employee employee = session.get(Employee.class, id);

if (employee != null) {

session.remove(employee);

System.out.println("Employee with ID " + id + " deleted successfully.");

} else {

System.out.println("Employee with ID " + id + " not found.");

}

tx.commit();

session.close();

} catch (Exception e) {

if (tx != null) {

tx.rollback();

}

e.printStackTrace();

}

}

}

**Factory.java**

package com.hibernate;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

public class Factory {

static SessionFactory factory=null;

public static SessionFactory getFactory() {

if(factory == null) {

Configuration conf=new Configuration();

conf.configure();

factory=conf.buildSessionFactory();

}

return factory;

}

}

**Employee.java**

package com.pojo;

import java.util.UUID;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = "employee")

public class Employee {

@Id

//@GeneratedValue(strategy = GenerationType.UUID)

@GeneratedValue(strategy = GenerationType.IDENTITY)

//@GeneratedValue(strategy = GenerationType.SEQUENCE)

int id;

//UUID id;

String name;

String department;

public Employee( String name, String department) {

super();

this.name = name;

this.department = department;

}

public Employee() {

super();

// TODO Auto-generated constructor stub

}

public int getId() {

return id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDepartment() {

return department;

}

public void setDepartment(String department) {

this.department = department;

}

}

**Main.java**

package pom.hibernate;

import java.util.List;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import com.dao.EmployeeDao;

import com.pojo.Employee;

public class Main {

public static void main(String[] args) {

EmployeeDao dao=new EmployeeDao();

for(int i=0;i<10;i++) {

dao.insert("Vname "+ i ,"CSE" );

}

List<Employee> data=dao.find();

for(int i=0;i<data.size();i++) {

System.out.println(data.get(i).getId()+" | ");

System.out.println(data.get(i).getName()+" | ");

System.out.println(data.get(i).getDepartment()+"\n");

}

dao.update(5, "Amit", "ECE");

data=dao.find();

for(int i=0;i<data.size();i++) {

System.out.println(data.get(i).getId()+" | ");

System.out.println(data.get(i).getName()+" | ");

System.out.println(data.get(i).getDepartment()+"\n");

}

dao.delete(8);

data=dao.find();

for(int i=0;i<data.size();i++) {

System.out.println(data.get(i).getId()+" | ");

System.out.println(data.get(i).getName()+" | ");

System.out.println(data.get(i).getDepartment()+"\n");

}

}

}

**pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven- 4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>Hibernate</groupId>

<artifactId>com.5A1hibernate</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!-- https://mvnrepository.com/artifact/org.hibernate.orm/hibernate-core -->

<dependency>

<groupId>org.hibernate.orm</groupId>

<artifactId>hibernate-core</artifactId>

<version>7.1.0.Final</version>

</dependency>

<!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j -->

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<version>9.4.0</version>

</dependency>

<!--

https://mvnrepository.com/artifact/jakarta.persistence/jakarta.persistence-api -->

<dependency>

<groupId>jakarta.persistence</groupId>

<artifactId>jakarta.persistence-api</artifactId>

<version>3.2.0</version>

</dependency>

</dependencies>

</project>

**hibernate.cfg.xml**

<?**xml** version=*"1.0"* encoding=*"UTF-8"*?>

<**hibernate-configuration**>

<**session-factory**>

<!-- Database connection settings -->

<**property** name=*"hibernate.connection.driver\_class"*>com.mysql.cj.jdbc.Driver</**property**>

<**property** name=*"hibernate.connection.url"*>jdbc:mysql://localhost:3306/mydb</**property**>

<**property** name=*"hibernate.connection.username"*>root</**property**>

<**property** name=*"hibernate.connection.password"*></**property**>

<!-- SQL dialect -->

<**property** name=*"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</**property**>

<!--Create Table if not exists-->

<**property** name=*"hibernate.hbm2ddl.auto"*>update</**property**>

<!-- Enable showing executed SQL in console -->

<**property** name=*"hibernate.show\_sql"*>true</**property**>

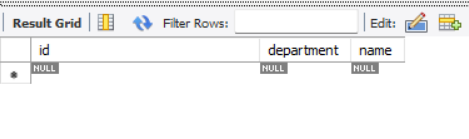
<!-- Mapping annotated classes -->

<**mapping** class=*"com.pojo.Employee"*/>

</**session-factory**>

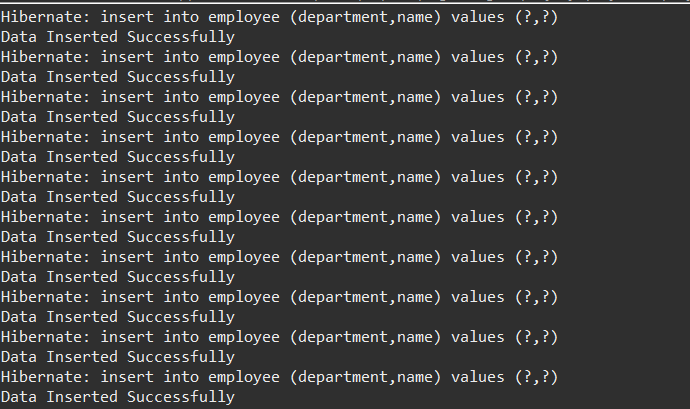
</**hibernate-configuration**>

**employee table initially(Before CRUD operations) :**



After CRUD operations on employee table :

**Console Output :**



A screenshot of a computer

AI-generated content may be incorrect.

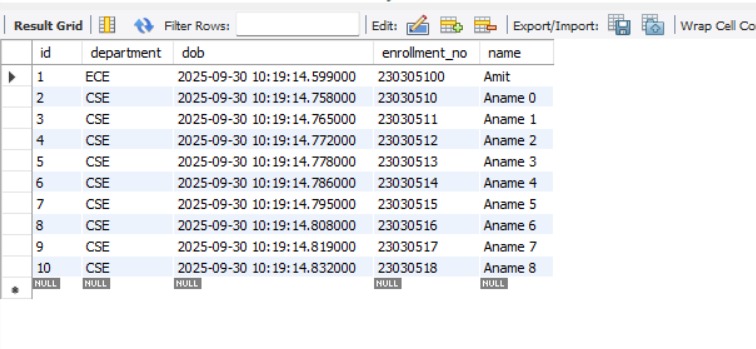
A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Database Output(After CRUD operations) :**



**Conclusion: -**

Hibernate simplifies database management by providing an object-relational mapping framework to efficiently perform CRUD operations with minimal SQL.

EXPERIMENT NO.: 6

**Aim: -** Create a application store the data in database to perform Spring CRUD operations.

**The Spring Framework :**

The Spring Framework is a comprehensive and modular application framework for Java. Its primary goal is to simplify the development of complex, enterprise-level applications by promoting **loose coupling** and **testability**.

At the heart of the Spring Framework is the **Inversion of Control (IoC) Container**, which manages the lifecycle and configuration of application objects (called "Beans").

**Core Concepts:**

* **Inversion of Control (IoC):** This is the fundamental principle of the Spring. Instead of the developer writing code to create and manage the lifecycle of objects, the Spring Container takes over this "control."
* **Dependency Injection (DI):** This is the primary design pattern Spring uses to achieve IoC. Instead of an object creating its own dependencies (e.g., new DatabaseService()), the framework "injects" those dependencies into the object (e.g., via a constructor or setter method). This makes components modular and easy to test in isolation.
* **Aspect-Oriented Programming (AOP):** Spring uses AOP to address **cross-cutting concerns**. These are services that apply to many parts of an application, such as logging, security, and transaction management. AOP allows you to declare this logic separately and apply it to your business objects without modifying their core code.
* **Modular Design:** Spring is not a single, monolithic framework but a collection of modules (e.g., **Spring Core**, **Spring MVC** for web apps, **Spring Boot** for simplified setup, **Spring Data** for database access). Developers can pick and choose the modules they need.

**Spring Application to perform CRUD Operations in a Database :-**

**EmployeeDao.java**

package com.dao;

import java.util.Date;

import java.util.List;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Repository;

import org.springframework.transaction.annotation.Transactional;

import com.pojo.Student;

@Repository

public class StudentDao {

private SessionFactory factory;

public StudentDao() {

super();

// TODO Auto-generated constructor stub

}

@Autowired

public void setFactory(SessionFactory factory) {

this.factory = factory;

}

**// Retrieve entire data**

@Transactional

public List<Student> find() {

Session session = factory.getCurrentSession();

List<Student> data = session.createQuery("From Student",Student.class).list();

System.out.println("data fetched successfully");

return data;

}

**// Retrieve data by id**

@Transactional

public Student find(Integer id) {

Session session = factory.getCurrentSession();

Student s = session.get(Student.class, id);

System.out.println("data fetched successfully");

return s;

}

**//Insert Data**

@Transactional

public void insert(String name, long enroll\_no, Date date, String dept) {

Session session = factory.getCurrentSession();

Student record = new Student(name,enroll\_no, date, dept);

session.persist(record);

System.out.println("data inserted successfully");

}

**//Update Data**

@Transactional

public void update(Student s) {

Session session = factory.getCurrentSession();

session.merge(s);

System.out.println("data updated successfully");

}

**//Delete Data**

@Transactional

public void remove(Student s) {

Session session = factory.getCurrentSession();

session.remove(s);

System.out.println("data removed successfully");

}

}

**Student.java**

package com.pojo;

import java.util.Date;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

@Entity

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

Integer id;

String name;

long enrollment\_no;

Date dob;

String department;

public Student() {

super();

// TODO Auto-generated constructor stub

}

public Student(String name, long enrollment\_no, Date dob, String department) {

super();

this.name = name;

this.enrollment\_no = enrollment\_no;

this.dob = dob;

this.department = department;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public long getEnrollment\_no() {

return enrollment\_no;

}

public void setEnrollment\_no(long enrollment\_no) {

this.enrollment\_no = enrollment\_no;

}

public Date getDob() {

return dob;

}

public void setDob(Date dob) {

this.dob = dob;

}

public String getDepartment() {

return department;

}

public void setDepartment(String department) {

this.department = department;

}

}

**AppConfig.java**

package com.config;

import java.util.Properties;

import javax.sql.DataSource;

import org.hibernate.SessionFactory;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.orm.hibernate5.HibernateTransactionManager;

import org.springframework.orm.hibernate5.LocalSessionFactoryBean;

import org.springframework.transaction.annotation.EnableTransactionManagement;

import com.mchange.v2.c3p0.ComboPooledDataSource;

@Configuration

@ComponentScan(basePackages = "com") // replaces <ctx:component-scan base-package="com"/>

@EnableTransactionManagement // replaces <tx:annotation-driven/>

public class AppConfig {

// Equivalent of <bean id="myDataSource" ...>

@Bean(destroyMethod = "close")

public DataSource myDataSource() throws Exception {

ComboPooledDataSource dataSource = new ComboPooledDataSource();

dataSource.setDriverClass("com.mysql.cj.jdbc.Driver");

dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/epj");

dataSource.setUser("root");

dataSource.setPassword("");

return dataSource;

}

// Equivalent of <bean id="sessionFactory" ...>

@Bean

public LocalSessionFactoryBean sessionFactory(DataSource myDataSource) {

LocalSessionFactoryBean sessionFactory = new LocalSessionFactoryBean();

sessionFactory.setDataSource(myDataSource);

sessionFactory.setPackagesToScan("com");

Properties hibernateProps = new Properties();

hibernateProps.put("hibernate.dialect", "org.hibernate.dialect.MySQLDialect");

hibernateProps.put("hibernate.show\_sql", "false");

hibernateProps.put("hibernate.hbm2ddl.auto", "update");

sessionFactory.setHibernateProperties(hibernateProps);

return sessionFactory;

}

// Equivalent of <bean id="myTransactionManager" ...>

@Bean

public HibernateTransactionManager transactionManager(SessionFactory sessionFactory) {

HibernateTransactionManager txManager = new HibernateTransactionManager();

txManager.setSessionFactory(sessionFactory);

return txManager;

}

}

/\*\*

\* @Configuration → replaces <beans> root.

\*

\* @ComponentScan("com") → replaces <ctx:component-scan base-package="com"/>.

\*

\* @EnableTransactionManagement → replaces <tx:annotation-driven/>.

\*

\* @Bean methods → replace <bean> tags.

\*

\* Properties for Hibernate are added via Properties object instead of <props> in XML.

\*

\* The datasource lifecycle (destroy-method="close") is preserved with @Bean(destroyMethod="close").

\*/

**SpringApplication.java**

package com.spring;

import java.util.Date;

import java.util.List;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.dao.StudentDao;

import com.pojo.Student;

public class SpringApplication {

public static void main(String[] args) {

ApplicationContext container = new ClassPathXmlApplicationContext("beans.xml");

StudentDao dao = container.getBean(StudentDao.class);

// insert single record

dao.insert("Amit", 230305100, new Date(), "CSE");

// insert 10 records

for(int i = 0; i < 10; i++) {

dao.insert("name " + i, (long)(23030510 + i), new Date(), "CSE");

}

// Retrieve and print record

List<Student> data = dao.find();

System.***out***.println();

for(int i = 0; i < data.size(); i++) {

System.***out***.print(data.get(i).getId() + " | ");

System.***out***.print(data.get(i).getName() + " | ");

System.***out***.print(data.get(i).getEnrollment\_no() + " | ");

System.***out***.print(data.get(i).getDob() + " | ");

System.***out***.print(data.get(i).getDepartment()+"\n");

}

System.***out***.println();

// update 1st record

Student s = data.getFirst();

s.setDepartment("ECE");

dao.update(s);

// print the updated row

s = dao.find(s.getId());

System.***out***.print("\nupdated details : \n" + s.getId() + " | ");

System.***out***.print(s.getName() + " | ");

System.***out***.print(s.getEnrollment\_no() + " | ");

System.***out***.print(s.getDob() + " | ");

System.***out***.print(s.getDepartment()+"\n");

// delete last row

s = data.getLast();

System.***out***.println("\nDeleted Record details : ");

System.***out***.print(s.getId() + " | ");

System.***out***.print(s.getName() + " | ");

System.***out***.print(s.getEnrollment\_no() + " | ");

System.***out***.print(s.getDob() + " | ");

System.***out***.print(s.getDepartment()+"\n");

dao.remove(s);

}

}

**SpringApplicationAnnotationBased.java**

package com.spring;

I mport java.util.Date;

import java.util.List;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.config.AppConfig;

import com.dao.StudentDao;

import com.pojo.Student;

import com.sun.tools.javac.launcher.Main;

public class SpringApplicationAnnotationBased {

public static void main(String[] args) {

// Ioc Container Creation

ApplicationContext container = new AnnotationConfigApplicationContext(AppConfig.class);

// Craete a object for student dao to preform curd operation on the student table

StudentDao dao = container.getBean(StudentDao.class);

// insert single record

dao.insert("Amit", 230305100, new Date(), "CSE");

// insert 10 records

for(int i = 0; i < 10; i++) {

dao.insert("name " + i, (long)(23030510 + i), new Date(), "CSE");

}

// Retrieve and print records

List<Student> data = dao.find();

System.out.println();

for(int i = 0; i < data.size(); i++) {

System.out.print(data.get(i).getId() + " | ");

System.out.print(data.get(i).getName() + " | ");

System.out.print(data.get(i).getEnrollment\_no() + " | ");

System.out.print(data.get(i).getDob() + " | ");

System.out.print(data.get(i).getDepartment()+"\n");

}

System.out.println();

// update 1st record/row

Student s = data.getFirst();

s.setDepartment("ECE");

dao.update(s);

// print the updated row details

s = dao.find(s.getId());

System.out.print("\nupdated details : \n" + s.getId() + " | ");

System.out.print(s.getName() + " | ");

System.out.print(s.getEnrollment\_no() + " | ");

System.out.print(s.getDob() + " | ");

System.out.print(s.getDepartment()+"\n");

// delete last row

s = data.getLast();

System.out.println("\nDeleted Record details : ");

System.out.print(s.getId() + " | ");

System.out.print(s.getName() + " | ");

System.out.print(s.getEnrollment\_no() + " | ");

System.out.print(s.getDob() + " | ");

System.out.print(s.getDepartment()+"\n");

dao.remove(s); // called to delete the record

}

}

**pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven- 4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>spring</groupId>

<artifactId>com.spring</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<!--

https://mvnrepository.com/artifact/org.springframework/spring-context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>6.2.11</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate.orm/hibernate-core -->

<dependency>

<groupId>org.hibernate.orm</groupId>

<artifactId>hibernate-core</artifactId>

<version>7.1.1.Final</version>

</dependency>

<!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j -->

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<version>9.4.0</version>

</dependency>

<!--

https://mvnrepository.com/artifact/jakarta.persistence/jakarta.persistence-api -->

<dependency>

<groupId>jakarta.persistence</groupId>

<artifactId>jakarta.persistence-api</artifactId>

<version>3.2.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-orm -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>6.2.11</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.springframework/spring-tx -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</artifactId>

<version>6.2.11</version>

</dependency>

<!-- https://mvnrepository.com/artifact/com.mchange/c3p0 -->

<dependency>

<groupId>com.mchange</groupId>

<artifactId>c3p0</artifactId>

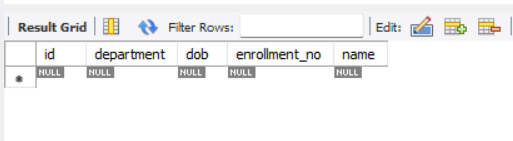
<version>0.11.2</version>

</dependency>

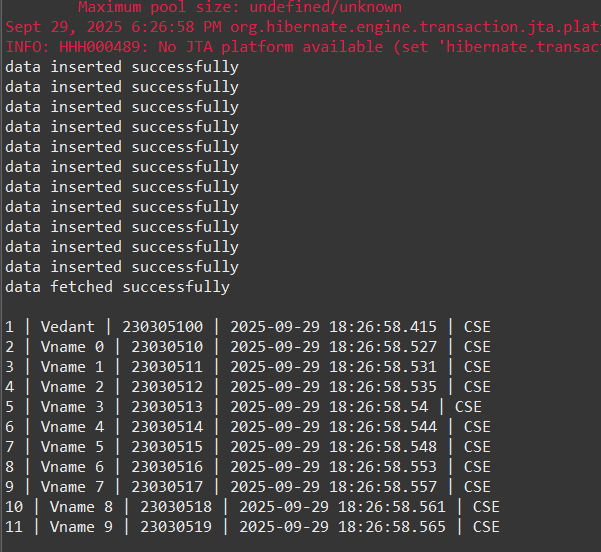
</dependencies>

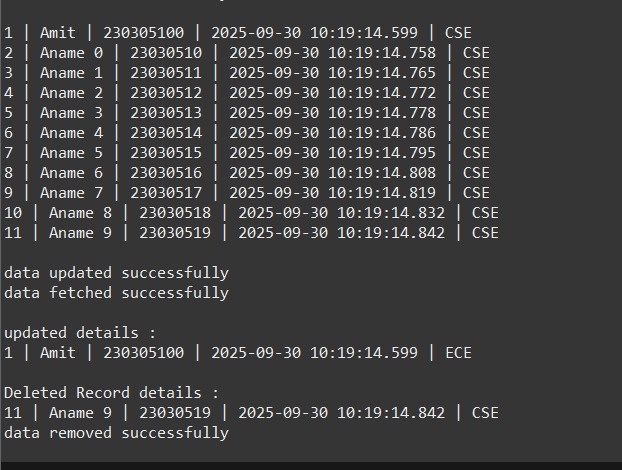
</project>

**student table before CRUD operations :**



**After CRUD operations :**





**Database :**

A screenshot of a computer

AI-generated content may be incorrect.

**Conclusion:**

In this experiment, we successfully developed a Spring application to perform all four CRUD (Create, Read, Update, Delete) operations. This practical demonstrated how the Spring framework, particularly Spring Data, simplifies data persistence. We learned how to efficiently interact with a database to create, retrieve, update, and delete records, confirming Spring's effectiveness in building robust, data-driven applications.

EXPERIMENT NO.: 7

**Aim:-** Create a web application to store the data in database with spring boot.

**The Spring Boot Framework:**

Spring Boot is an extension of the Spring Framework that aims to radically simplify the setup and development of new Spring applications. It is not a replacement for Spring but rather a tool that makes it faster and easier to use. It follows a "convention over configuration" philosophy.

Its primary goal is to allow developers to create production-ready, stand-alone Spring applications that can "just run" with minimal configuration.

**Core Concepts:**

* **Autoconfiguration:** This is the most powerful feature. Spring Boot inspects the libraries (JARs) on your application's classpath and automatically configures the necessary Spring beans based on what it finds. For example, if it sees a web dependency (spring-boot-starter-web), it automatically configures a Tomcat server and Spring MVC.
* **"Starter" Dependencies:** Spring Boot provides a set of "starter" dependencies (e.g., spring-boot-starter-web, spring-boot-starter-data-jpa). These are pre-packaged bundles of common libraries that are tested and known to work well together. This simplifies build configuration, as you only need to add one starter instead of many individual dependencies.
* **Embedded Server:** Spring Boot applications include an embedded web server (like Tomcat, Jetty, or Undertow) by default. This allows the application to be packaged as a single, executable .jar file, which can be run with a simple java -jar myapp.jar command, eliminating the need to deploy a .war file to an external server.
* **Production-Ready Features:** Spring Boot includes non-functional features out-of-the-box, most notably the **Actuator** module. The Actuator provides built-in endpoints for monitoring and managing the application (e.g., checking health, viewing metrics, and understanding configurations).

**Spring Boot Application to perform CRUD operations on Database:**

**Student.java**

package com.springboot.pojo;

import java.util.Date;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

@Entity

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

Integer id;

String name;

long enrollment\_no;

Date dob;

String department;

public Student() {

super();

// TODO Auto-generated constructor stub

}

public Student(String name, long enrollment\_no, Date dob, String department) {

super();

this.name = name;

this.enrollment\_no = enrollment\_no;

this.dob = dob;

this.department = department;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public long getEnrollment\_no() {

return enrollment\_no;

}

public void setEnrollment\_no(long enrollment\_no) {

this.enrollment\_no = enrollment\_no;

}

public Date getDob() {

return dob;

}

public void setDob(Date dob) {

this.dob = dob;

}

public String getDepartment() {

return department;

}

public void setDepartment(String department) {

this.department = department;

}

}

**StudentDao.java**

package com.springboot.dao;

import java.util.Date;

import java.util.List;

import org.springframework.stereotype.Repository;

import org.springframework.transaction.annotation.Transactional;

import com.springboot.pojo.Student;

import jakarta.persistence.EntityManager;

import jakarta.persistence.PersistenceContext;

@Repository

public class StudentDao {

@PersistenceContext

private EntityManager entityManager; // injected by Spring

**// Retrieve entire data**

@Transactional(readOnly = true)

public List<Student> find() {

List<Student> data = entityManager.createQuery("FROM Student", Student.class).getResultList();

System.out.println("data fetched successfully");

return data;

}

**// Retrieve data by id**

@Transactional(readOnly = true)

public Student find(Integer id) {

Student s = entityManager.find(Student.class, id);

System.out.println("data fetched successfully");

return s;

}

**// Insert new student**

@Transactional

public void insert(String name, long enroll\_no, Date date, String dept) {

Student record = new Student(name, enroll\_no, date, dept);

entityManager.persist(record);

System.out.println("data inserted successfully");

}

**// Update existing student**

@Transactional

public void update(Student s) {

entityManager.merge(s);

System.out.println("data updated successfully");

}

**// Remove student**

@Transactional

public void remove(Student s) {

entityManager.remove(entityManager.contains(s) ? s : entityManager.merge(s));

System.out.println("data removed successfully");

}

}

**Application.java**

package com.springboot;

import java.util.Date;

import java.util.List;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import com.springboot.dao.StudentDao;

import com.springboot.pojo.Student;

@SpringBootApplication

public class Application {

public static void main(String[] args) {

ApplicationContext container = SpringApplication.run(Application.class, args);

StudentDao dao = container.getBean(StudentDao.class);

**// insert single record**

dao.insert("Amit", 230510500, new Date(), "CSE");

// insert 10 records

for (int i = 0; i < 10; i++) {

dao.insert("Vname " + i, (long) (230510501 + i), new Date(), "CSE");

}

**// Retrieve and print record**

List<Student> data = dao.find();

System.out.println();

for (int i = 0; i < data.size(); i++) {

System.out.print(data.get(i).getId() + " | ");

System.out.print(data.get(i).getName() + " | ");

System.out.print(data.get(i).getEnrollment\_no() + " | ");

System.out.print(data.get(i).getDob() + " | ");

System.out.print(data.get(i).getDepartment() + "\n");

}

System.out.println();

**// update 1st record**

if (!data.isEmpty()) {

Student s = data.getFirst(); // Java 21 method

s.setDepartment("AIML");

dao.update(s);

// print the updated row}

s = dao.find(s.getId());

System.out.print("\nupdated details : \n" + s.getId() + " | ");

System.out.print(s.getName() + " | ");

System.out.print(s.getEnrollment\_no() + " | ");

System.out.print(s.getDob() + " | ");

System.out.print(s.getDepartment() + "\n");

} else {

System.out.println("No students found in DB.");

}

**// delete last row**

if (!data.isEmpty()) {

Student s = data.getLast();

System.out.println("\nDeleted Record details : ");

System.out.print(s.getId() + " | ");

System.out.print(s.getName() + " | ");

System.out.print(s.getEnrollment\_no() + " | ");

System.out.print(s.getDob() + " | ");

System.out.print(s.getDepartment() + "\n");

dao.remove(s);

}

}

}

**application.properties**

spring.application.name=SpringBoot

# ========================

# Data source

# ========================

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/epj

spring.datasource.username=root

spring.datasource.password=

# ===============================

# HIBERNATE / JPA

# ===============================

spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=false

# ===============================

# CONNECTION POOL (C3P0)

# ===============================

spring.datasource.type=com.mchange.v2.c3p0.ComboPooledDataSource

**pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven- 4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>4.0.0-SNAPSHOT</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<groupId>springboot</groupId>

<artifactId>com.springboot</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>SpringBoot</name>

<description>Spring Boot curd operation</description>

<url />

<licenses>

<license />

</licenses>

<developers>

<developer />

</developers>

<scm>

<connection />

<developerConnection />

<tag />

<url />

</scm>

<properties>

<java.version>21</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

<optional>true</optional>

</dependency>

<!-- https://mvnrepository.com/artifact/com.mchange/c3p0 -->

<dependency>

<groupId>com.mchange</groupId>

<artifactId>c3p0</artifactId>

<version>0.11.2</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<configuration>

<annotationProcessorPaths>

<path>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</path>

</annotationProcessorPaths>

</configuration>

</plugin>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<excludes>

<exclude>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</exclude>

</excludes>

</configuration>

</plugin>

</plugins>

</build>

<repositories>

<repository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<releases>

<enabled>false</enabled>

</releases>

</repository>

</repositories>

<pluginRepositories>

<pluginRepository>

<id>spring-snapshots</id>

<name>Spring Snapshots</name>

<url>https://repo.spring.io/snapshot</url>

<releases>

<enabled>false</enabled>

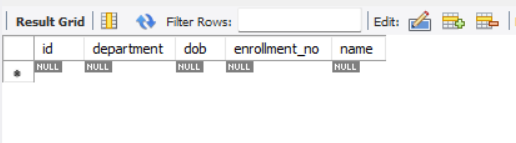
</releases>

</pluginRepository>

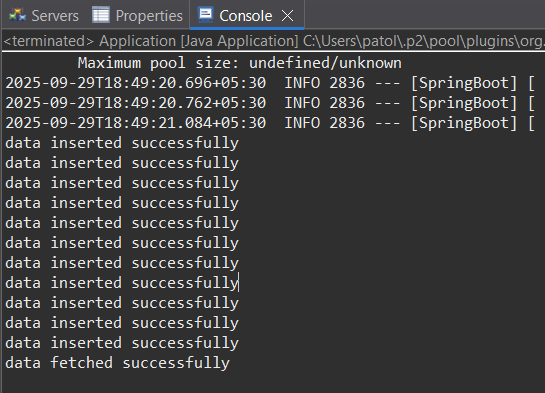
</pluginRepositories>

</project>

**student Database before CRUD operations :**



**After CRUD operations:**

****

A screenshot of a computer

AI-generated content may be incorrect.

**Database output:**

A screenshot of a computer

AI-generated content may be incorrect.

**Conclusion:**

This experiment confirmed that Spring Boot rapidly accelerates the development of a data-driven web application. We learned that its "starter" dependencies and autoconfiguration seamlessly integrate the web and database layers, allowing us to store and manage data with minimal setup.

**DOCUMENT LINK:-**

https://github.com/Err0R2811/EPJ\_MANUAL/blob/master/EPJ%20MANUAL.docx