**Practical NO 1.1:**

**Write a Java program to connect to a specific database (e.g., MySQL, workbench etc.) using JDBC. Create a table in the database using JDBC and insert some sample data and retrieve all data from a specific table and display it on the console.**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.Statement;

import java.sql.ResultSet;

import java.sql.SQLException;

public class JdbcExample {

static final String DB\_URL = "jdbc:mysql://localhost:3306/jdbc";

static final String DB\_USER = "root";

static final String DB\_PASSWORD = "Rohit@0801";

public static void main(String[] args) {

Connection conn = null;

Statement stmt = null;

try {

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

conn = DriverManager.getConnection(DB\_URL, DB\_USER, DB\_PASSWORD);

if (conn != null) {

System.out.println("Database connected");

}

stmt = conn.createStatement();

String createTableSQL = "CREATE TABLE IF NOT EXISTS emp (" +

"empno INTEGER PRIMARY KEY, " +

"name VARCHAR(20), " +

"job VARCHAR(20), " +

"salary INTEGER, " +

"deptno INTEGER)";

stmt.executeUpdate(createTableSQL);

String insertSQL1 = "INSERT INTO emp VALUES(1001, 'Amit Jain', 'clerk', 18000, 10)";

String insertSQL2 = "INSERT INTO emp VALUES(1002, 'Gopal Pandey', 'manager', 450000, 20)";

String insertSQL3 = "INSERT INTO emp VALUES(1003, 'Mona Mantri', 'clerk', 20000, 10)";

String insertSQL4 = "INSERT INTO emp VALUES(1004, 'Raja Patil', 'driver', 15000, 40)";

String insertSQL5 = "INSERT INTO emp VALUES(1005, 'Rohan Rathi', 'sr.clerk', 25000, 10)";

stmt.executeUpdate(insertSQL1);

stmt.executeUpdate(insertSQL2);

stmt.executeUpdate(insertSQL3);

stmt.executeUpdate(insertSQL4);

stmt.executeUpdate(insertSQL5);

String selectSQL = "SELECT \* FROM emp";

ResultSet rs = stmt.executeQuery(selectSQL);

System.out.println("Data from 'emp' table:");

while (rs.next()) {

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

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

String job = rs.getString("job");

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

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

System.out.printf("EmpNo: %d, Name: %s, Job: %s, Salary: %d, DeptNo: %d%n", empno, name, job, salary,

deptno);

}

rs.close();

} catch (SQLException se) {

se.printStackTrace();

} catch (Exception e) {

e.printStackTrace();

} finally {

try {

if (stmt != null)

stmt.close();

if (conn != null)

conn.close();

} catch (SQLException se) {

se.printStackTrace();

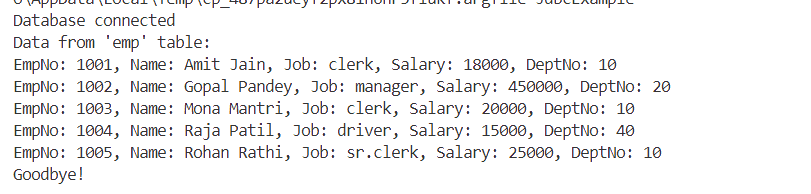
}

}

System.out.println("Goodbye!");

}

}



**Practical NO 1.2:**

**Implement a program to update a specific record in a table based on a given condition and delete a record from a table based on a specific criteria.**

**Crud.java**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.Scanner;

public class EmployeeCRUDApp {

static final String DB\_URL = "jdbc:mysql://localhost:3306/jdbc";

static final String DB\_USER = "root";

static final String DB\_PASSWORD = "Rohit@0801";

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

try (Connection conn = DriverManager.getConnection(DB\_URL, DB\_USER, DB\_PASSWORD)) {

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

boolean exit = false;

while (!exit) {

System.out.println("\nEmployee CRUD Operations:");

System.out.println("1. Create Employee");

System.out.println("2. Read All Employees");

System.out.println("3. Update Employee Salary");

System.out.println("4. Delete Employee");

System.out.println("5. Exit");

System.out.print("Select an option: ");

int choice = scanner.nextInt();

scanner.nextLine();

switch (choice) {

case 1:

createEmployee(conn, scanner);

break;

case 2:

readAllEmployees(conn);

break;

case 3:

updateEmployeeSalary(conn, scanner);

break;

case 4:

deleteEmployee(conn, scanner);

break;

case 5:

exit = true;

break;

default:

System.out.println("Invalid choice. Please try again.");

}

}

} catch (Exception e) {

System.out.println("Error: " + e.getMessage());

} finally {

scanner.close();

}

}

public static void createEmployee(Connection conn, Scanner scanner) {

try {

System.out.print("Enter employee name: ");

String name = scanner.nextLine();

System.out.print("Enter position: ");

String position = scanner.nextLine();

System.out.print("Enter salary: ");

double salary = scanner.nextDouble();

scanner.nextLine(); // Consume newline

System.out.print("Enter department: ");

String department = scanner.nextLine();

String insertSQL = "INSERT INTO Employee (name, position, salary, department) VALUES (?, ?, ?, ?)";

try (PreparedStatement pstmt = conn.prepareStatement(insertSQL)) {

pstmt.setString(1, name);

pstmt.setString(2, position);

pstmt.setDouble(3, salary);

pstmt.setString(4, department);

int rows = pstmt.executeUpdate();

System.out.println("Inserted " + rows + " row(s) successfully.");

}

} catch (SQLException e) {

System.out.println("Error inserting employee: " + e.getMessage());

}

}

public static void readAllEmployees(Connection conn) {

String selectSQL = "SELECT \* FROM Employee";

try (PreparedStatement pstmt = conn.prepareStatement(selectSQL);

ResultSet rs = pstmt.executeQuery()) {

System.out.println("\nEmployee Records:");

System.out.printf("%-5s %-20s %-20s %-10s %-15s%n", "ID", "Name", "Position", "Salary", "Department");

System.out.println("------------------------------------------------------------");

while (rs.next()) {

int id = rs.getInt("emp\_id");

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

String position = rs.getString("position");

double salary = rs.getDouble("salary");

String department = rs.getString("department");

System.out.printf("%-5d %-20s %-20s %-10.2f %-15s%n", id, name, position, salary, department);

}

} catch (SQLException e) {

System.out.println("Error retrieving employees: " + e.getMessage());

}

}

public static void updateEmployeeSalary(Connection conn, Scanner scanner) {

try {

System.out.print("Enter employee ID to update: ");

int empId = scanner.nextInt();

System.out.print("Enter new salary: ");

double newSalary = scanner.nextDouble();

String updateSQL = "UPDATE Employee SET salary = ? WHERE emp\_id = ?";

try (PreparedStatement pstmt = conn.prepareStatement(updateSQL)) {

pstmt.setDouble(1, newSalary);

pstmt.setInt(2, empId);

int rows = pstmt.executeUpdate();

if (rows > 0) {

System.out.println("Updated employee salary successfully.");

} else {

System.out.println("Employee not found.");

}

}

} catch (SQLException e) {

System.out.println("Error updating employee salary: " + e.getMessage());

}

}

public static void deleteEmployee(Connection conn, Scanner scanner) {

try {

System.out.print("Enter employee ID to delete: ");

int empId = scanner.nextInt();

String deleteSQL = "DELETE FROM Employee WHERE emp\_id = ?";

try (PreparedStatement pstmt = conn.prepareStatement(deleteSQL)) {

pstmt.setInt(1, empId);

int rows = pstmt.executeUpdate();

if (rows > 0) {

System.out.println("Deleted employee successfully.");

} else {

System.out.println("Employee not found.");

}

}

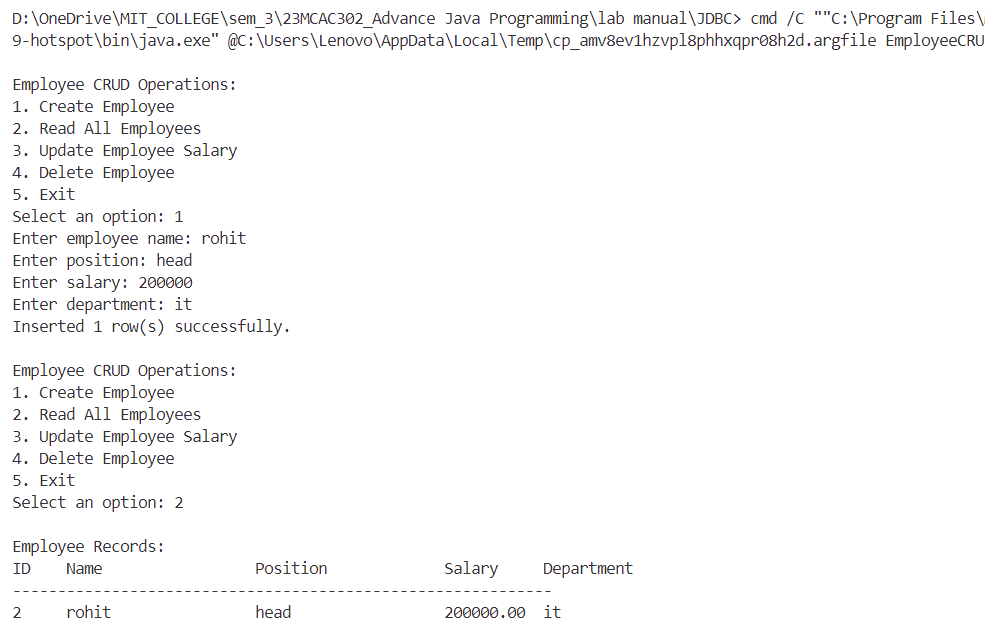
} catch (SQLException e) {

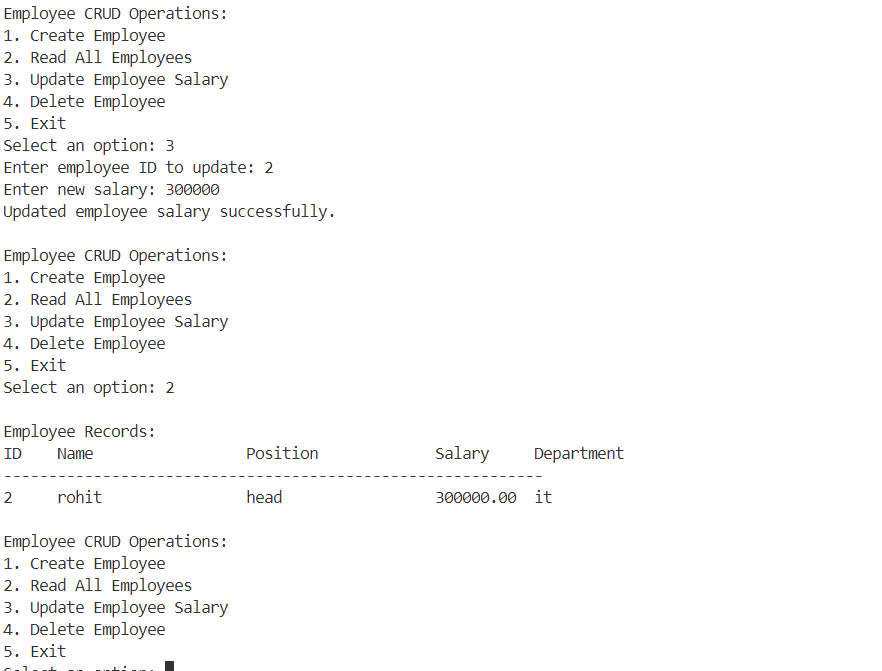
System.out.println("Error deleting employee: " + e.getMessage());

}

}

}

****

****

****

**Practical NO 1.3:**

**Write a program to utilize transactions in JDBC, demonstrating both commit and rollback functionalities.**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Statement;

public class JdbcTransactionExample {

static final String DB\_URL = "jdbc:mysql://localhost:3306/jdbc";

static final String DB\_USER = "root";

static final String DB\_PASSWORD = "Rohit@0801";

public static void main(String[] args) {

Connection conn = null;

Statement stmt = null;

try {

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

conn = DriverManager.getConnection(DB\_URL, DB\_USER, DB\_PASSWORD);

conn.setAutoCommit(false);

stmt = conn.createStatement();

String insertSQL1 = "INSERT INTO emp VALUES(1009, 'Deepak Rathod', 'analyst', 40000, 30)";

String insertSQL2 = "INSERT INTO emp VALUES(1010, 'Adarsh Nair', 'developer', 50000, 20)";

String insertSQL3 = "INSERT INTO emp VALUES(1011, 'Ritesh Bobhate', 'manager', 60000, 40)";

stmt.executeUpdate(insertSQL1);

stmt.executeUpdate(insertSQL2);

stmt.executeUpdate(insertSQL3);

conn.commit();

System.out.println("Transaction committed successfully.");

} catch (SQLException se) {

try {

if (conn != null)

conn.rollback();

System.out.println("Transaction rolled back due to an error.");

} catch (SQLException rollbackEx) {

rollbackEx.printStackTrace();

}

se.printStackTrace();

} catch (Exception e) {

e.printStackTrace();

} finally {

try {

if (stmt != null)

stmt.close();

if (conn != null)

conn.close();

} catch (SQLException se) {

se.printStackTrace();

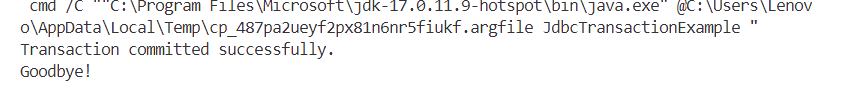
}

}

System.out.println("Goodbye!");

}

}



**Practical NO 1.4:**

**Implement a program to handle different types of JDBC exceptions effectively. Write JDBC Program to calculate Employee salary and print the salary statement in tabular form by selecting the details from database table (Emp\_Sal) using Prepared Statement.**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.SQLException;**

**import java.sql.SQLIntegrityConstraintViolationException;**

**import java.sql.Statement;**

**public class JDBCExceptionHandlingExample {**

**static final String DB\_URL = "jdbc:mysql://localhost:3306/jdbc";**

**static final String DB\_USER = "root";**

**static final String DB\_PASSWORD = "Rohit@0801";**

**public static void main(String[] args) {**

**Connection conn = null;**

**Statement stmt = null;**

**try {**

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

**conn = DriverManager.getConnection(DB\_URL, DB\_USER, DB\_PASSWORD);**

**stmt = conn.createStatement();**

**String sql = "INSERT INTO emp (empno, name, job, salary, deptno) VALUES (1005, 'John Doe', 'Analyst', 40000, 10)";**

**stmt.executeUpdate(sql);**

**} catch (SQLIntegrityConstraintViolationException e) {**

**System.out.println("Data integrity error: Duplicate entry or constraint violation. " + e.getMessage());**

**} catch (SQLException e) {**

**switch (e.getSQLState()) {**

**case "08001":**

**System.out.println("Database connection error: " + e.getMessage());**

**break;**

**case "42000":**

**System.out.println("SQL syntax error: " + e.getMessage());**

**break;**

**default:**

**System.out.println("SQL error occurred: " + e.getMessage());**

**}**

**} catch (Exception e) {**

**System.out.println("Unexpected error: " + e.getMessage());**

**} finally {**

**try {**

**if (stmt != null)**

**stmt.close();**

**if (conn != null)**

**conn.close();**

**} catch (SQLException e) {**

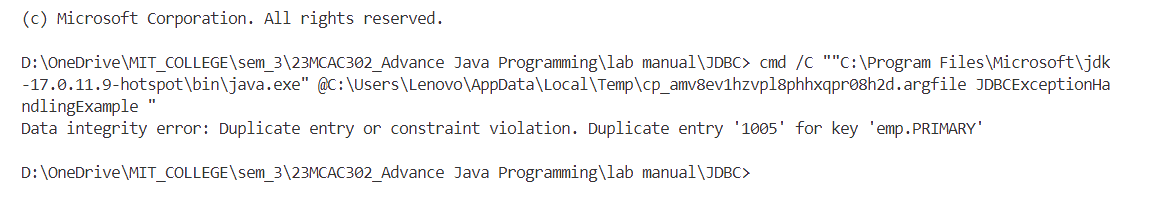
**System.out.println("Error closing resources: " + e.getMessage());**

**}**

**}**

**}**

**}**

****

**Practical NO 1.5:**

**Write a program to perform aggregation functions (e.g., COUNT, SUM,AVERAGE) on data retrieved from a database.**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.PreparedStatement;**

**import java.sql.ResultSet;**

**import java.sql.SQLException;**

**public class EmployeeAggregationExample {**

**static final String DB\_URL = "jdbc:mysql://localhost:3306/jdbc";**

**static final String DB\_USER = "root";**

**static final String DB\_PASSWORD = "Rohit@0801";**

**public static void main(String[] args) {**

**Connection conn = null;**

**PreparedStatement pstmt = null;**

**ResultSet rs = null;**

**try {**

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

**conn = DriverManager.getConnection(DB\_URL, DB\_USER, DB\_PASSWORD);**

**// Perform COUNT, SUM, and AVG on base\_salary column**

**String query = "SELECT COUNT(\*) AS emp\_count, SUM(base\_salary) AS total\_salary, AVG(base\_salary) AS avg\_salary FROM Emp\_Sal";**

**pstmt = conn.prepareStatement(query);**

**rs = pstmt.executeQuery();**

**if (rs.next()) {**

**int employeeCount = rs.getInt("emp\_count");**

**double totalSalary = rs.getDouble("total\_salary");**

**double averageSalary = rs.getDouble("avg\_salary");**

**System.out.println("Employee Aggregation Results:");**

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

**System.out.printf("Total Employees: %d%n", employeeCount);**

**System.out.printf("Sum of Salaries: %.2f%n", totalSalary);**

**System.out.printf("Average Salary: %.2f%n", averageSalary);**

**}**

**} catch (SQLException e) {**

**System.out.println("Database error: " + e.getMessage());**

**} catch (ClassNotFoundException e) {**

**System.out.println("JDBC Driver not found: " + e.getMessage());**

**} finally {**

**try {**

**if (rs != null)**

**rs.close();**

**if (pstmt != null)**

**pstmt.close();**

**if (conn != null)**

**conn.close();**

**} catch (SQLException e) {**

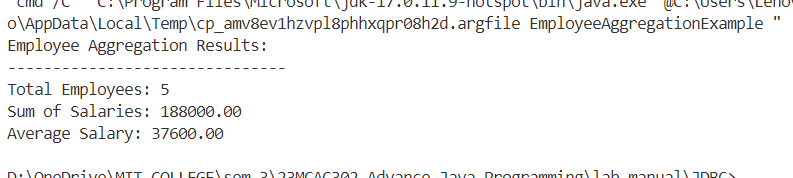
**System.out.println("Error closing resources: " + e.getMessage());**

**}**

**}**

**}**

**}**

****

**Practical NO 1.6:**

**Write a program to create a simple Java application that interacts with a database to perform CRUD operations (Create, Read, Update, Delete) on a specific table.**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.PreparedStatement;**

**import java.sql.ResultSet;**

**import java.sql.SQLException;**

**import java.util.Scanner;**

**public class EmployeeCRUDApp {**

**static final String DB\_URL = "jdbc:mysql://localhost:3306/jdbc";**

**static final String DB\_USER = "root";**

**static final String DB\_PASSWORD = "Rohit@0801";**

**public static void main(String[] args) {**

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

**try (Connection conn = DriverManager.getConnection(DB\_URL, DB\_USER, DB\_PASSWORD)) {**

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

**boolean exit = false;**

**while (!exit) {**

**System.out.println("\nEmployee CRUD Operations:");**

**System.out.println("1. Create Employee");**

**System.out.println("2. Read All Employees");**

**System.out.println("3. Update Employee Salary");**

**System.out.println("4. Delete Employee");**

**System.out.println("5. Exit");**

**System.out.print("Select an option: ");**

**int choice = scanner.nextInt();**

**scanner.nextLine();**

**switch (choice) {**

**case 1:**

**createEmployee(conn, scanner);**

**break;**

**case 2:**

**readAllEmployees(conn);**

**break;**

**case 3:**

**updateEmployeeSalary(conn, scanner);**

**break;**

**case 4:**

**deleteEmployee(conn, scanner);**

**break;**

**case 5:**

**exit = true;**

**break;**

**default:**

**System.out.println("Invalid choice. Please try again.");**

**}**

**}**

**} catch (Exception e) {**

**System.out.println("Error: " + e.getMessage());**

**} finally {**

**scanner.close();**

**}**

**}**

**public static void createEmployee(Connection conn, Scanner scanner) {**

**try {**

**System.out.print("Enter employee name: ");**

**String name = scanner.nextLine();**

**System.out.print("Enter position: ");**

**String position = scanner.nextLine();**

**System.out.print("Enter salary: ");**

**double salary = scanner.nextDouble();**

**scanner.nextLine(); // Consume newline**

**System.out.print("Enter department: ");**

**String department = scanner.nextLine();**

**String insertSQL = "INSERT INTO Employee (name, position, salary, department) VALUES (?, ?, ?, ?)";**

**try (PreparedStatement pstmt = conn.prepareStatement(insertSQL)) {**

**pstmt.setString(1, name);**

**pstmt.setString(2, position);**

**pstmt.setDouble(3, salary);**

**pstmt.setString(4, department);**

**int rows = pstmt.executeUpdate();**

**System.out.println("Inserted " + rows + " row(s) successfully.");**

**}**

**} catch (SQLException e) {**

**System.out.println("Error inserting employee: " + e.getMessage());**

**}**

**}**

**public static void readAllEmployees(Connection conn) {**

**String selectSQL = "SELECT \* FROM Employee";**

**try (PreparedStatement pstmt = conn.prepareStatement(selectSQL);**

**ResultSet rs = pstmt.executeQuery()) {**

**System.out.println("\nEmployee Records:");**

**System.out.printf("%-5s %-20s %-20s %-10s %-15s%n", "ID", "Name", "Position", "Salary", "Department");**

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

**while (rs.next()) {**

**int id = rs.getInt("emp\_id");**

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

**String position = rs.getString("position");**

**double salary = rs.getDouble("salary");**

**String department = rs.getString("department");**

**System.out.printf("%-5d %-20s %-20s %-10.2f %-15s%n", id, name, position, salary, department);**

**}**

**} catch (SQLException e) {**

**System.out.println("Error retrieving employees: " + e.getMessage());**

**}**

**}**

**public static void updateEmployeeSalary(Connection conn, Scanner scanner) {**

**try {**

**System.out.print("Enter employee ID to update: ");**

**int empId = scanner.nextInt();**

**System.out.print("Enter new salary: ");**

**double newSalary = scanner.nextDouble();**

**String updateSQL = "UPDATE Employee SET salary = ? WHERE emp\_id = ?";**

**try (PreparedStatement pstmt = conn.prepareStatement(updateSQL)) {**

**pstmt.setDouble(1, newSalary);**

**pstmt.setInt(2, empId);**

**int rows = pstmt.executeUpdate();**

**if (rows > 0) {**

**System.out.println("Updated employee salary successfully.");**

**} else {**

**System.out.println("Employee not found.");**

**}**

**}**

**} catch (SQLException e) {**

**System.out.println("Error updating employee salary: " + e.getMessage());**

**}**

**}**

**public static void deleteEmployee(Connection conn, Scanner scanner) {**

**try {**

**System.out.print("Enter employee ID to delete: ");**

**int empId = scanner.nextInt();**

**String deleteSQL = "DELETE FROM Employee WHERE emp\_id = ?";**

**try (PreparedStatement pstmt = conn.prepareStatement(deleteSQL)) {**

**pstmt.setInt(1, empId);**

**int rows = pstmt.executeUpdate();**

**if (rows > 0) {**

**System.out.println("Deleted employee successfully.");**

**} else {**

**System.out.println("Employee not found.");**

**}**

**}**

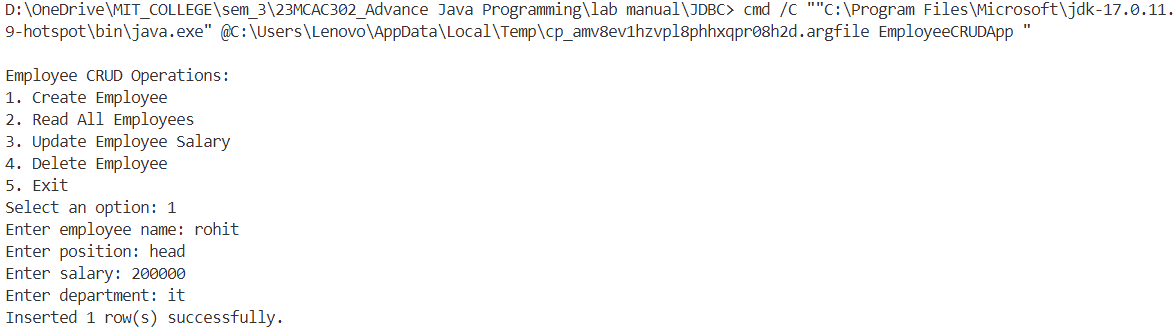
**} catch (SQLException e) {**

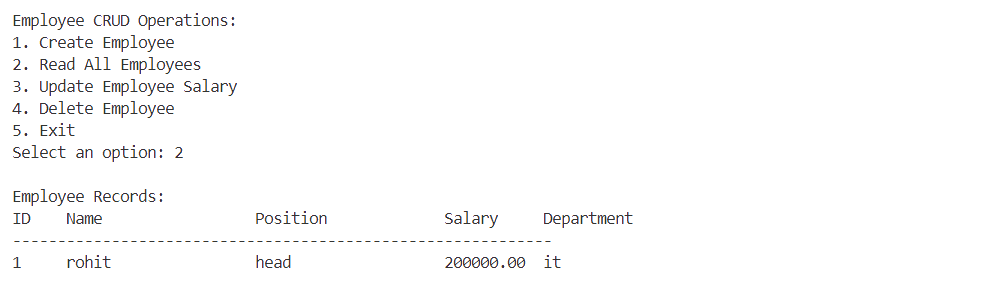
**System.out.println("Error deleting employee: " + e.getMessage());**

**}**

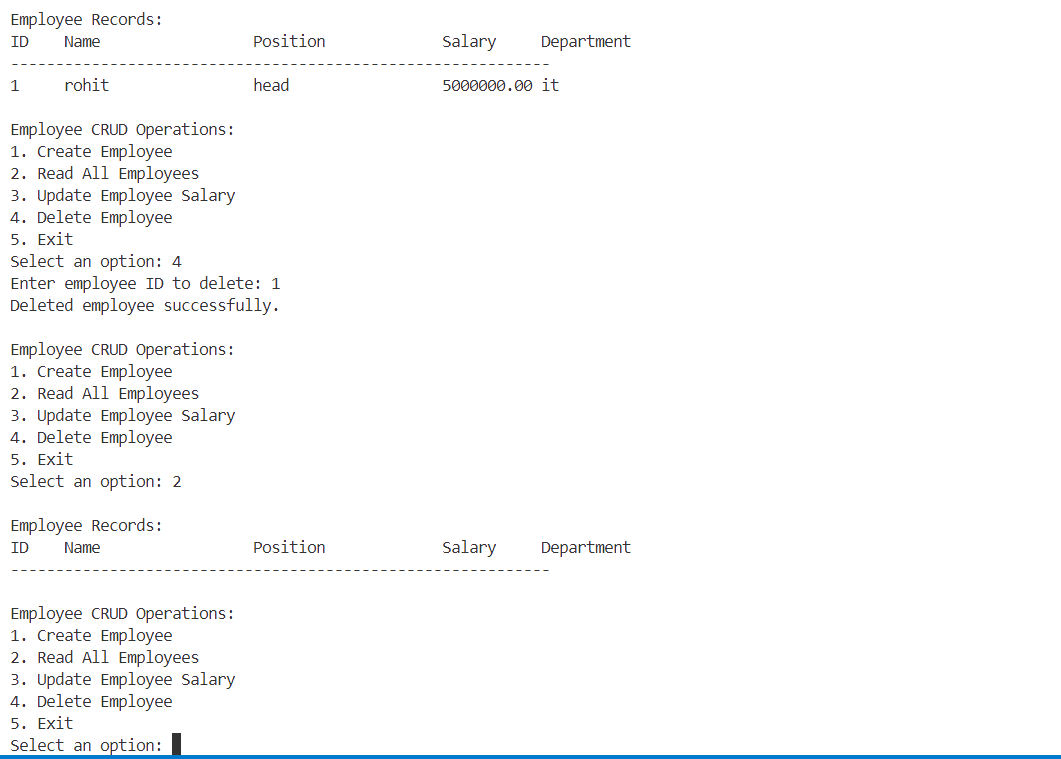
**}**

**}**

****

****

****

****

**Practical NO 1.7**

**Develop a program that utilizes JDBC to implement a basic login functionality with user authentication against a database.**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.PreparedStatement;**

**import java.sql.ResultSet;**

**import java.sql.SQLException;**

**import java.util.Scanner;**

**public class UserLoginApp {**

**static final String DB\_URL = "jdbc:mysql://localhost:3306/jdbc";**

**static final String DB\_USER = "root";**

**static final String DB\_PASSWORD = "Rohit@0801";**

**public static void main(String[] args) {**

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

**try (Connection conn = DriverManager.getConnection(DB\_URL, DB\_USER, DB\_PASSWORD)) {**

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

**System.out.println("Welcome to the Login System");**

**System.out.print("Enter Username: ");**

**String username = scanner.nextLine();**

**System.out.print("Enter Password: ");**

**String password = scanner.nextLine();**

**boolean loginSuccess = authenticateUser(conn, username, password);**

**if (loginSuccess) {**

**System.out.println("Login successful! Welcome, " + username + ".");**

**} else {**

**System.out.println("Invalid username or password. Please try again.");**

**}**

**} catch (Exception e) {**

**System.out.println("Error: " + e.getMessage());**

**} finally {**

**scanner.close();**

**}**

**}**

**public static boolean authenticateUser(Connection conn, String username, String password) {**

**String query = "SELECT \* FROM Users WHERE username = ? AND password = ?";**

**try (PreparedStatement pstmt = conn.prepareStatement(query)) {**

**pstmt.setString(1, username);**

**pstmt.setString(2, password);**

**try (ResultSet rs = pstmt.executeQuery()) {**

**return rs.next(); // If a record is found, return true for successful login**

**}**

**} catch (SQLException e) {**

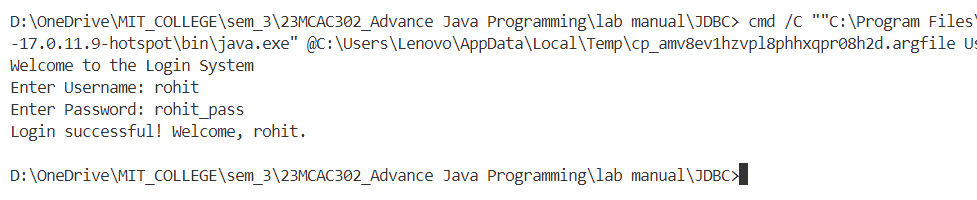
**System.out.println("Error during authentication: " + e.getMessage());**

**}**

**return false; // If no record is found, return false for failed login**

**}**

**}**

****

**Practical NO 2.1:**

**Design a simple servlet that displays a welcome message with the user's name retrieved from request parameters.**

package com.example;

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;

import java.io.IOException;

/\*\*

\* Servlet implementation class WelcomeServlet

\*/

@WebServlet(name = "welcome", urlPatterns = { "/welcome" })

public class WelcomeServlet extends HttpServlet {

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

// Retrieve the 'name' parameter from the request

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

// Set response content type

response.setContentType("text/html");

// Prepare the output HTML

PrintWriter out = response.getWriter();

out.println("<html><body>");

if (name != null && !name.isEmpty()) {

out.println("<h1>Welcome, " + name + "!</h1>");

} else {

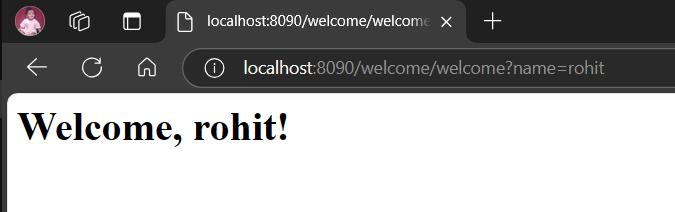
out.println("<h1>Welcome, Guest!</h1>");

}

out.println("</body></html>");

}

}

****

**Practical NO 2.2:**

**Implement a servlet that handles a login form and validates user credentials against a database.**

**Login.jsp**

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

pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

<title>Login</title>

</head>

<body>

<h2>Login</h2>

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

<label for="username">Username:</label>

<input type="text" id="username" name="username" required>

<br>

<label for="password">Password:</label>

<input type="password" id="password" name="password" required>

<br>

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

</form>

</body>

</html>

**Login.java**

package com.example;

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 java.io.IOException;

import java.io.PrintWriter;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

/\*\*

\* Servlet implementation class login

\*/

@WebServlet("/login")

public class login extends HttpServlet {

private static final long serialVersionUID = 1L;

// Database credentials

private static final String JDBC\_URL = "jdbc:mysql://localhost:3306/userdb?useSSL=false";

private static final String USER = "root";

private static final String PASSWORD = "Rohit@0801";

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

// Get username and password from request

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

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

response.setContentType("text/html");

PrintWriter out = response.getWriter();

// Database connection and validation

try {

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

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

Connection conn = DriverManager.getConnection(JDBC\_URL, USER, PASSWORD);

// Prepare SQL statement

String sql = "SELECT \* FROM users WHERE username = ? AND password = ?";

PreparedStatement statement = conn.prepareStatement(sql);

statement.setString(1, username);

statement.setString(2, password);

// Execute query

ResultSet rs = statement.executeQuery();

if (rs.next()) {

// Login successful

out.println("<h2>Welcome, " + username + "!</h2>");

} else {

// Login failed

out.println("<h2>Invalid username or password.</h2>");

out.println("<a href=\"login.jsp\">Try Again</a>");

}

} catch (SQLException e) {

e.printStackTrace();

out.println(e);

out.println("<h2>Database connection failed.</h2>");

} catch (ClassNotFoundException e) {

e.printStackTrace();

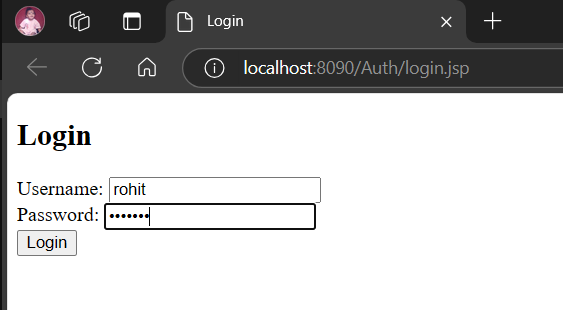
out.println(e);

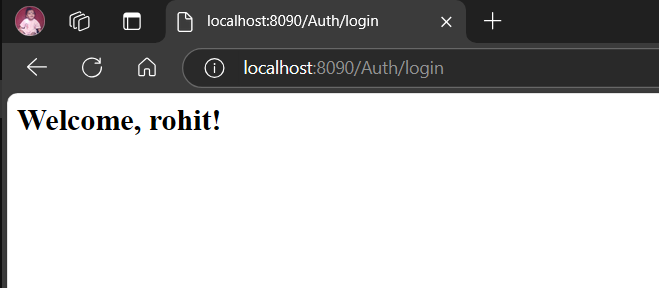
out.println("<h2>MySQL Driver not found.</h2>");

}

}

}

****

****

**Practical NO 2.3:**

**Create a servlet that utilizes session management to maintain a shopping cart for an online store.**

**addItem.html**

<!DOCTYPE html>

<html>

<head>

<title>Add Item</title>

</head>

<body>

<h1>Add Item to Cart</h1>

<form action="cart" method="post">

<label for="item">Item Name:</label>

<input type="text" id="item" name="item" required>

<br>

<label for="quantity">Quantity:</label>

<input type="number" id="quantity" name="quantity" required>

<br>

<input type="submit" value="Add to Cart">

</form>

<a href="cart">View Cart</a>

</body>

</html>

* **Cart.java**

package com.example;

import java.io.PrintWriter;

import java.util.HashMap;

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 java.io.IOException;

import jakarta.servlet.http.HttpSession;

/\*\*

\* Servlet implementation class ShoppingCartServlet

\*/

@WebServlet("/cart")

public class ShoppingCartServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

@Override

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

response.setContentType("text/html");

PrintWriter out = response.getWriter();

HttpSession session = request.getSession();

// Retrieve the shopping cart from the session or create a new one

@SuppressWarnings("unchecked")

HashMap<String, Integer> cart = (HashMap<String, Integer>) session.getAttribute("cart");

if (cart == null) {

cart = new HashMap<>();

session.setAttribute("cart", cart);

}

// Display the current cart

out.println("<html><body>");

out.println("<h1>Shopping Cart</h1>");

if (cart.isEmpty()) {

out.println("<p>Your cart is empty.</p>");

} else {

out.println("<ul>");

for (HashMap.Entry<String, Integer> entry : cart.entrySet()) {

out.println("<li>" + entry.getKey() + ": " + entry.getValue() + "</li>");

}

out.println("</ul>");

}

out.println("<a href='addItem.html'>Add More Items</a>");

out.println("<a href='?action=clear'>Clear Cart</a>");

// Handle cart clearing

String action = request.getParameter("action");

if ("clear".equals(action)) {

cart.clear();

out.println("<p>Cart cleared!</p>");

}

out.println("</body></html>");

}

@Override

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

response.setContentType("text/html");

PrintWriter out = response.getWriter();

HttpSession session = request.getSession();

// Retrieve the shopping cart from the session or create a new one

@SuppressWarnings("unchecked")

HashMap<String, Integer> cart = (HashMap<String, Integer>) session.getAttribute("cart");

if (cart == null) {

cart = new HashMap<>();

session.setAttribute("cart", cart);

}

// Add item to the cart

String item = request.getParameter("item");

int quantity = Integer.parseInt(request.getParameter("quantity"));

cart.put(item, cart.getOrDefault(item, 0) + quantity);

out.println("<html><body>");

out.println("<h1>Item Added</h1>");

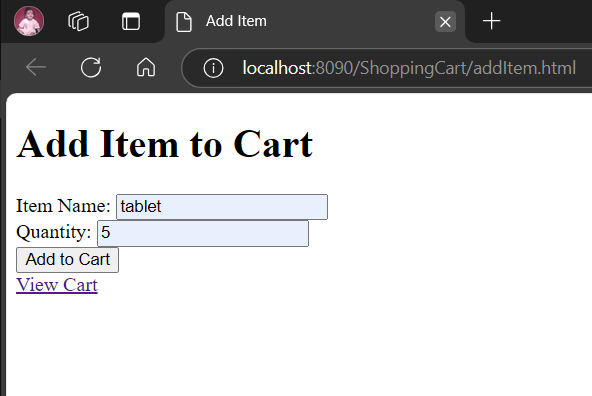
out.println("<p>Added " + quantity + " of " + item + " to your cart.</p>");

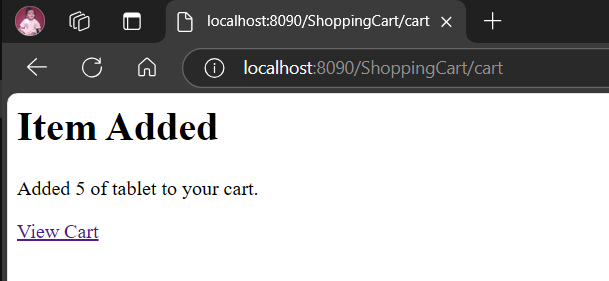
out.println("<a href='cart'>View Cart</a>");

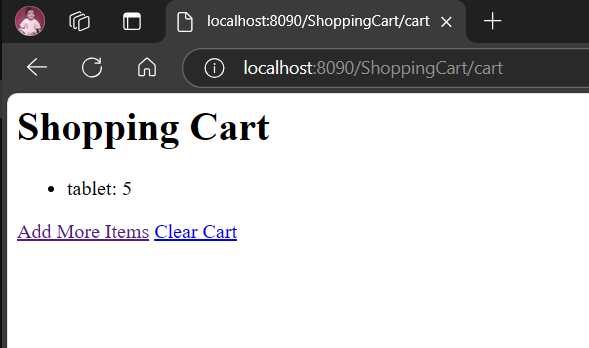
out.println("</body></html>");

}

}

****

****

****

**Practical NO 2.4:**

**Write a Servlet Program to calculate Employee Salary and print the salary statement.**

* **Employe.java**

package com.exmaple;

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;

import java.io.IOException;

/\*\*

\* Servlet implementation class EmployeeSalaryServlet

\*/

@WebServlet("/calculateSalary")

public class EmployeeSalaryServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

@Override

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

response.setContentType("text/html");

PrintWriter out = response.getWriter();

// Retrieve input parameters from the request

String empName = request.getParameter("empName");

double basicSalary = Double.parseDouble(request.getParameter("basicSalary"));

double hra = Double.parseDouble(request.getParameter("hra"));

double da = Double.parseDouble(request.getParameter("da"));

double tax = Double.parseDouble(request.getParameter("tax"));

// Calculate net salary

double grossSalary = basicSalary + hra + da;

double netSalary = grossSalary - tax;

// Print salary statement

out.println("<html><body>");

out.println("<h1>Employee Salary Statement</h1>");

out.println("<p><strong>Employee Name:</strong> " + empName + "</p>");

out.println("<p><strong>Basic Salary:</strong> " + basicSalary + "</p>");

out.println("<p><strong>HRA:</strong> " + hra + "</p>");

out.println("<p><strong>DA:</strong> " + da + "</p>");

out.println("<p><strong>Tax Deducted:</strong> " + tax + "</p>");

out.println("<p><strong>Gross Salary:</strong> " + grossSalary + "</p>");

out.println("<p><strong>Net Salary:</strong> " + netSalary + "</p>");

out.println("<a href='employeeSalary.html'>Calculate Again</a>");

out.println("</body></html>");

}

}

**Employes.html**

<!DOCTYPE html>

<html>

<head>

<title>Employee Salary Calculator</title>

</head>

<body>

<h1>Employee Salary Calculator</h1>

<form action="calculateSalary" method="post">

<label for="empName">Employee Name:</label>

<input type="text" id="empName" name="empName" required><br><br>

<label for="basicSalary">Basic Salary:</label>

<input type="number" id="basicSalary" name="basicSalary" required><br><br>

<label for="hra">HRA:</label>

<input type="number" id="hra" name="hra" required><br><br>

<label for="da">DA:</label>

<input type="number" id="da" name="da" required><br><br>

<label for="tax">Tax:</label>

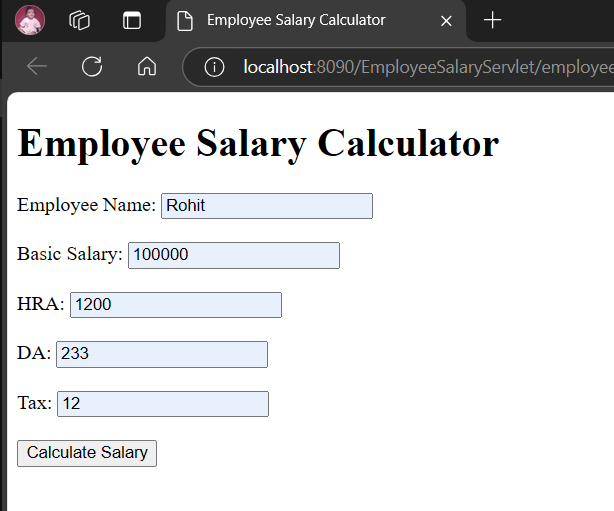
<input type="number" id="tax" name="tax" required><br><br>

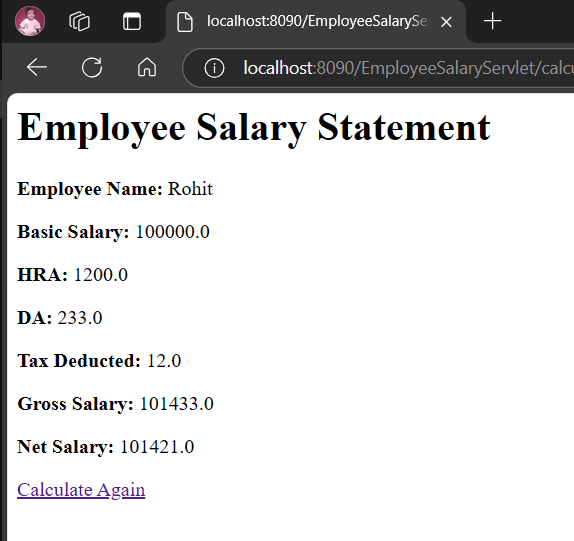
<input type="submit" value="Calculate Salary">

</form>

</body>

</html>

****

****

**Practical NO 2.5:**

**Write a servlet Program to calculate the addition of two numbers and print the result.(Eg:Addition of two numbers=50).**

**Add.html**

<!DOCTYPE html>

<html>

<head>

<title>Addition of Two Numbers</title>

</head>

<body>

<h1>Addition Calculator</h1>

<form action="addNumbers" method="post">

<label for="number1">Enter First Number:</label>

<input type="number" id="number1" name="number1" required><br><br>

<label for="number2">Enter Second Number:</label>

<input type="number" id="number2" name="number2" required><br><br>

<input type="submit" value="Calculate Addition">

</form>

</body>

</html>

**Add.java**

package com.example;

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;

import java.io.IOException;

/\*\*

\* Servlet implementation class AdditionServlet

\*/

@WebServlet("/addNumbers")

public class AdditionServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

@Override

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

response.setContentType("text/html");

PrintWriter out = response.getWriter();

// Retrieve input parameters

int number1 = Integer.parseInt(request.getParameter("number1"));

int number2 = Integer.parseInt(request.getParameter("number2"));

// Calculate addition

int sum = number1 + number2;

// Display result

out.println("<html><body>");

out.println("<h1>Result of Addition</h1>");

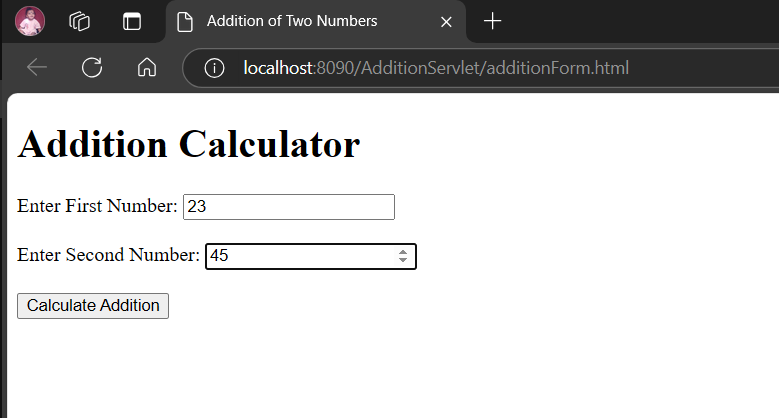
out.println("<p>Addition of two numbers = " + sum + "</p>");

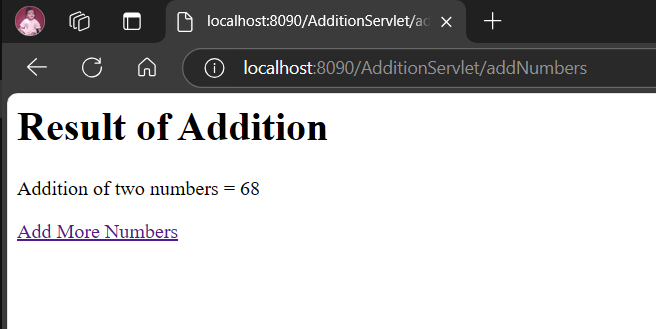
out.println("<a href='additionForm.html'>Add More Numbers</a>");

out.println("</body></html>");

}

}





**Practical NO 2.6:**

**Write a Servlet Program to create a registration form using in html and CSS and print the message Registration is successful.**

**Registration.html**

package com.example;

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;

import java.io.IOException;

/\*\*

\* Servlet implementation class AdditionServlet

\*/

@WebServlet("/addNumbers")

public class AdditionServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

@Override

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

response.setContentType("text/html");

PrintWriter out = response.getWriter();

// Retrieve input parameters

int number1 = Integer.parseInt(request.getParameter("number1"));

int number2 = Integer.parseInt(request.getParameter("number2"));

// Calculate addition

int sum = number1 + number2;

// Display result

out.println("<html><body>");

out.println("<h1>Result of Addition</h1>");

out.println("<p>Addition of two numbers = " + sum + "</p>");

out.println("<a href='additionForm.html'>Add More Numbers</a>");

out.println("</body></html>");

}

}

**Registration.java**

package com.example;

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;

import java.io.IOException;

/\*\*

\* Servlet implementation class RegistrationServlet

\*/

@WebServlet("/register")

public class RegistrationServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

@Override

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

response.setContentType("text/html");

PrintWriter out = response.getWriter();

// Retrieve form data

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

String email = request.getParameter("email");

String phone = request.getParameter("phone");

// Display registration success message

out.println("<html><head>");

out.println("<style>");

out.println("body { font-family: Arial, sans-serif; margin: 50px; }");

out.println(".message { color: green; font-size: 20px; }");

out.println("</style>");

out.println("</head><body>");

out.println("<h1>Registration Successful</h1>");

out.println("<p class='message'>Thank you, " + name + ", for registering!</p>");

out.println("<p>Your email: " + email + "</p>");

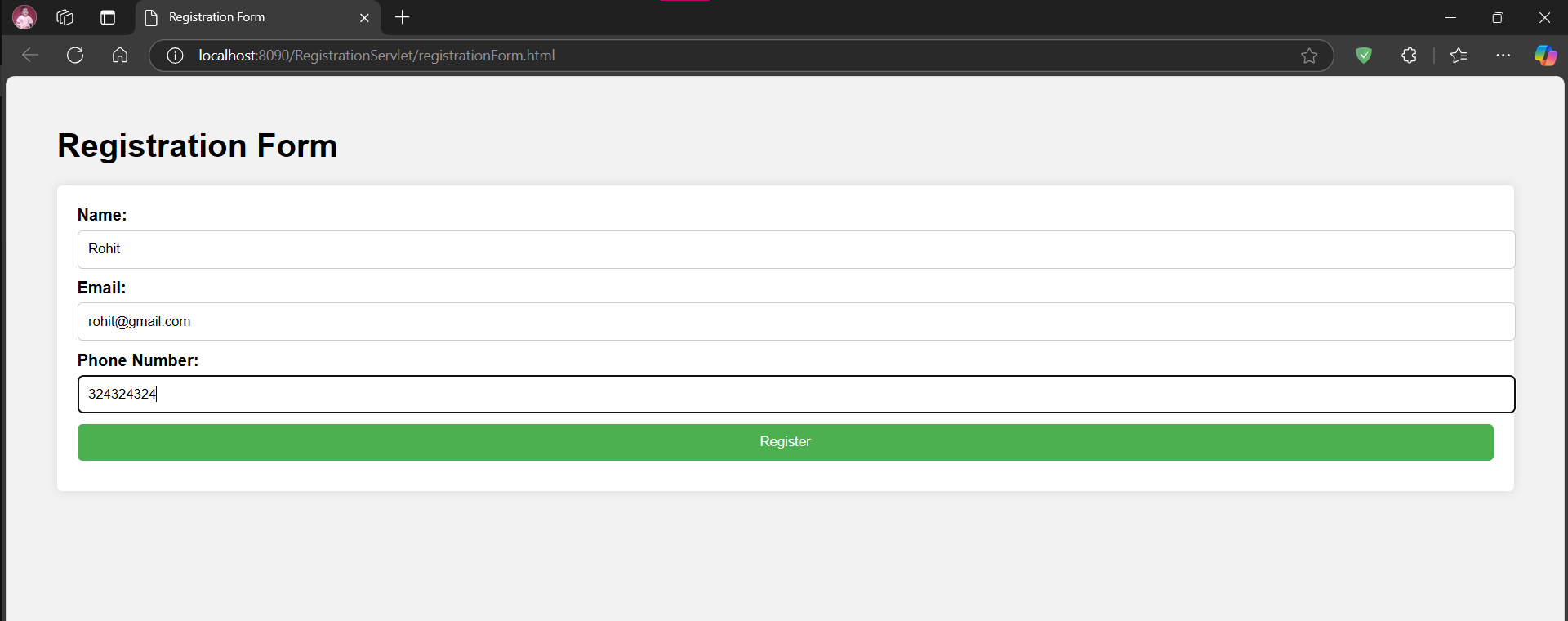
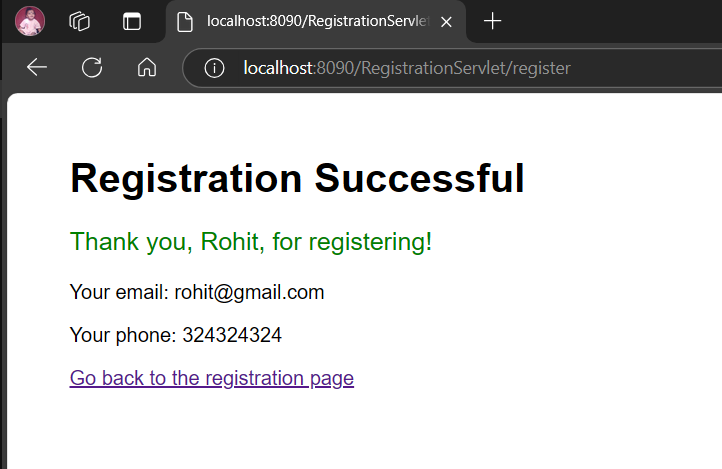
out.println("<p>Your phone: " + phone + "</p>");

out.println("<a href='registrationForm.html'>Go back to the registration page</a>");

out.println("</body></html>");

}

}

**Practical NO 2.7:**

**Write a servlet Program for student information and display the information in tabular form by selecting the details from student database table.**

**Studt.java**

package com.example;

import java.io.PrintWriter;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

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 java.io.IOException;

import java.sql.SQLException;

/\*\*

\* Servlet implementation class StudentInfoServlet

\*/

@WebServlet("/studentInfo")

public class StudentInfoServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

@Override

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

response.setContentType("text/html");

PrintWriter out = response.getWriter();

// JDBC setup

String jdbcURL = "jdbc:mysql://localhost:3306/student\_db";

String jdbcUsername = "root";

String jdbcPassword = "Rohit@0801"; // Replace with your MySQL password

try {

// Establish connection

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

Connection connection = DriverManager.getConnection(jdbcURL, jdbcUsername, jdbcPassword);

// Query student details

String sql = "SELECT \* FROM students";

Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery(sql);

// Display student details in a table

out.println("<html><head><title>Student Information</title></head><body>");

out.println("<h1>Student Information</h1>");

out.println("<table border='1' cellpadding='10'>");

out.println("<tr><th>ID</th><th>Name</th><th>Age</th><th>Grade</th><th>Email</th></tr>");

while (resultSet.next()) {

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

String name = resultSet.getString("name");

int age = resultSet.getInt("age");

String grade = resultSet.getString("grade");

String email = resultSet.getString("email");

out.println("<tr>");

out.println("<td>" + id + "</td>");

out.println("<td>" + name + "</td>");

out.println("<td>" + age + "</td>");

out.println("<td>" + grade + "</td>");

out.println("<td>" + email + "</td>");

out.println("</tr>");

}

out.println("</table>");

out.println("</body></html>");

resultSet.close();

statement.close();

connection.close();

} catch (Exception e) {

e.printStackTrace();

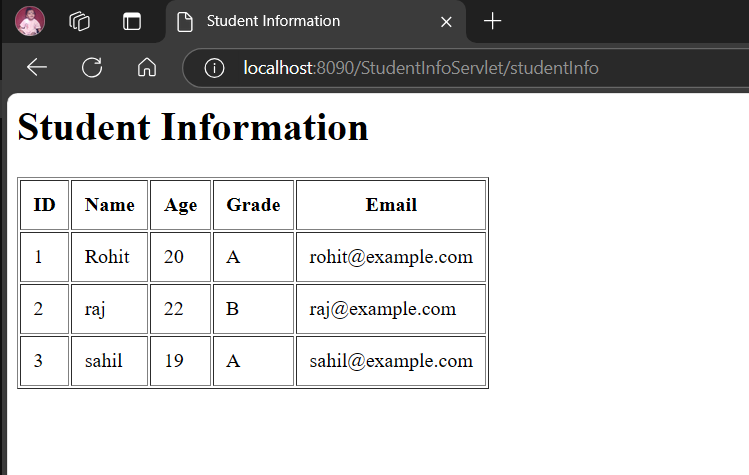
out.println("<p>Error: Unable to fetch data from the database.</p>");

out.println("<p>Details: " + e + "</p>");

}

}

}

****

**Practical NO 2.8:**

**Write a Java Servlet program to read employee details including employee number (empno), name, designation, basic pay, deductions, and allowances, and then calculate and display the net salary. display the information in tabular form by selecting the details from Emp\_sal database table.**

**Emp.java**

package com.example;

import java.io.IOException;

import java.io.PrintWriter;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

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 java.io.IOException;

/\*\*

\* Servlet implementation class EmployeeServlet

\*/

@WebServlet("/employeeDetails")

public class EmployeeServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

@Override

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

response.setContentType("text/html");

PrintWriter out = response.getWriter();

// Database credentials

String jdbcURL = "jdbc:mysql://localhost:3306/employee\_db";

String jdbcUsername = "root";

String jdbcPassword = "Rohit@0801"; // Replace with your MySQL password

try {

// Load JDBC driver

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

// Establish connection

Connection connection = DriverManager.getConnection(jdbcURL, jdbcUsername, jdbcPassword);

// Query the employee salary details

String sql = "SELECT \* FROM Emp\_sal";

Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery(sql);

// Display employee details in a table

out.println("<html><head><title>Employee Details</title></head><body>");

out.println("<h1>Employee Salary Details</h1>");

out.println("<table border='1' cellpadding='10'>");

out.println("<tr><th>Emp No</th><th>Name</th><th>Designation</th><th>Basic Pay</th><th>Deductions</th><th>Allowances</th><th>Net Salary</th></tr>");

while (resultSet.next()) {

int empno = resultSet.getInt("empno");

String name = resultSet.getString("name");

String designation = resultSet.getString("designation");

double basicPay = resultSet.getDouble("basic\_pay");

double deductions = resultSet.getDouble("deductions");

double allowances = resultSet.getDouble("allowances");

// Calculate net salary

double netSalary = basicPay + allowances - deductions;

// Display employee data

out.println("<tr>");

out.println("<td>" + empno + "</td>");

out.println("<td>" + name + "</td>");

out.println("<td>" + designation + "</td>");

out.println("<td>" + basicPay + "</td>");

out.println("<td>" + deductions + "</td>");

out.println("<td>" + allowances + "</td>");

out.println("<td>" + netSalary + "</td>");

out.println("</tr>");

}

out.println("</table>");

out.println("</body></html>");

resultSet.close();

statement.close();

connection.close();

} catch (Exception e) {

e.printStackTrace();

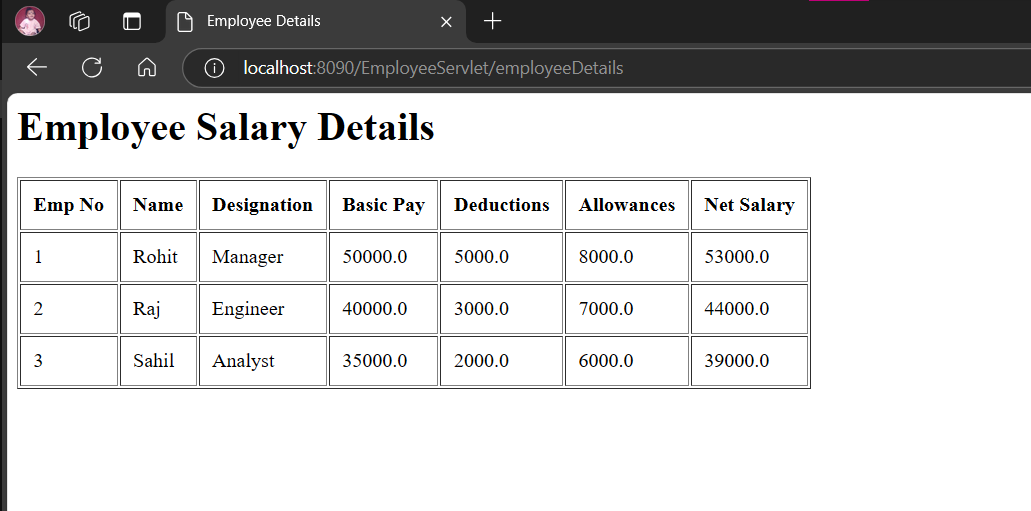
out.println("<p>Error: Unable to fetch employee details.</p>");

out.println("<p>Details: " + e.getMessage() + "</p>");

}

}

}

****

**Practical NO 3.1:**

**Write a JSP program calculates factorial of an integer number, while the input is taken from an HTML form.**

**Index.html**

<html>

<body>

<form action="Factorial.jsp">

Enter a value for n: <input type="text" name="val">

<input type="submit" value="Submit">

</form>

</body>

</html>

**Factorial.jsp**

<html>

<body>

<%!

long n, result;

String str;

long fact(long n) {

if(n==0)

return 1;

else

return n\*fact(n-1);

}

%>

<%

str = request.getParameter("val");

n = Long.parseLong(str);

result = fact(n);

%>

<b>Factorial value: </b> <%= result %>

</body>

</html>

**Demo.java**

package com.example;

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 java.io.IOException;

/\*\*

\* Servlet implementation class demo

\*/

public class demo extends HttpServlet {

private static final long serialVersionUID = 1L;

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

public demo() {

super();

// TODO Auto-generated constructor stub

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

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

// TODO Auto-generated method stub

response.getWriter().append("Served at: ").append(request.getContextPath());

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

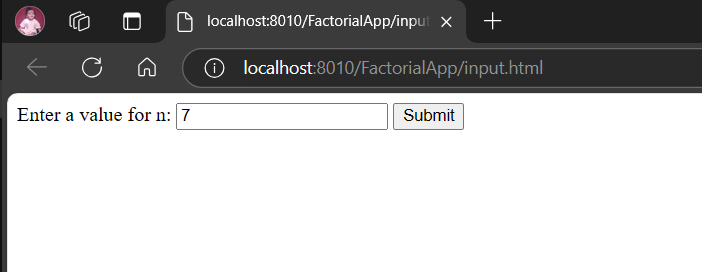
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

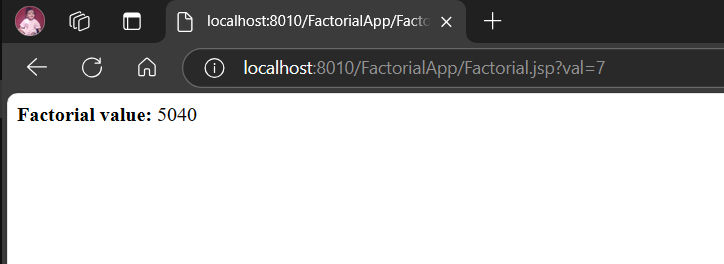
// TODO Auto-generated method stub

doGet(request, response);

}

}

****

****

**Practical NO 3.2:**

**Write a JSP program to generate the Fibonacci series up to a particular term, while the input is taken from an HTML form.**

**Index.html**

<html>

<body>

<form action="Fibonacci.jsp">

Enter a value for n: <input type="text" name="val">

<input type="submit" value="Submit">

</form>

</body>

</html>

**Fibonacci.jsp**

<html>

<body>

<%!

int n;

String str;

int fibo(int n) {

if(n<2)

return n;

else

return fibo(n-1) + fibo(n-2);

}

%>

<b>Fibonacci series: </b><br>

<%

str = request.getParameter("val");

n = Integer.parseInt(str);

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

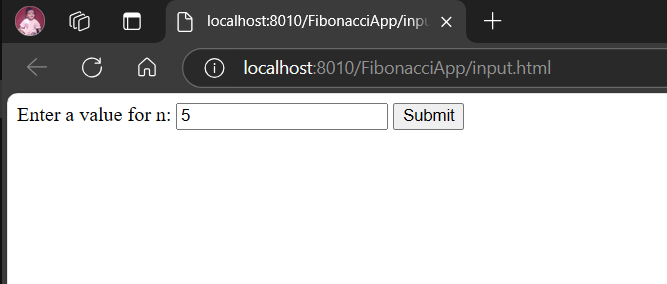
out.print(fibo(i) + " ");

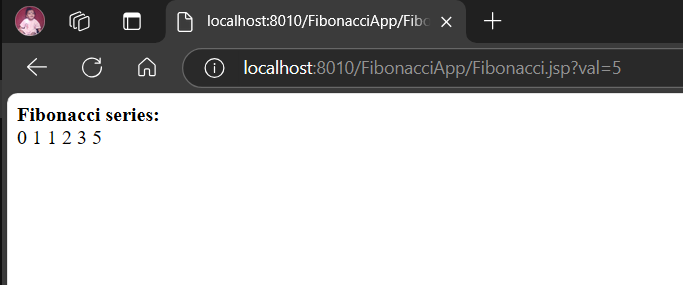
}

%>

</body>

</html>

****

****

**Practical NO 3.3:**

**Write a JSP program to display the System date and time.**

**Date.jsp**

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

pageEncoding="UTF-8"%>

<html>

<body>

<%-- JSP comments --%>

<%@page import="java.util.Date"%>

<%!

Date date;

%>

<%

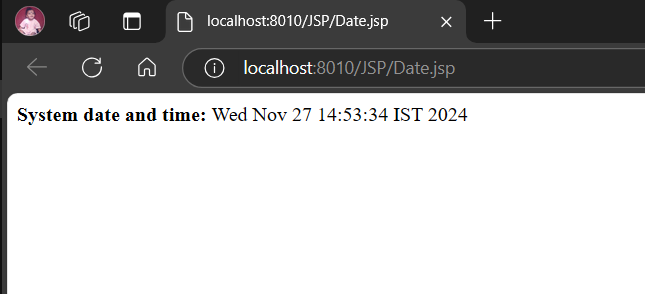
date = new Date();

%>

<b>System date and time: </b> <%= date %>

</body>

</html>

****

**Practical NO 3.4:**

**Write a JSP program to display a Sample shopping Order calculation Form and display output in tabular form.**

**Order.jsp**

<HTML>

<HEAD>

<TITLE>A Catalog Order Form</TITLE>

</HEAD>

<BODY>

<H1 ALIGN="center">A Sample Order Form</H1>

<%!

String item[] = {"DVD", "CD", "Diskette"};

double price[] = {190, 200, 100};

int quantity[] = {2, 9, 24};

%>

<TABLE ALIGN="center" BGCOLOR="lightgray" BORDER="1" WIDTH="75%">

<TR><TD>Item</TD>

<TD>Price</TD>

<TD>Quantity</TD>

<TD>Total Price</TD>

</TR>

<% for (int i=0; i<3; i++) { %>

<TR><TD><%= item[i] %></TD>

<TD><%= price[i] %></TD>

<TD><%= quantity[i] %></TD>

<TD><%= price[i] \* quantity[i] %></TD>

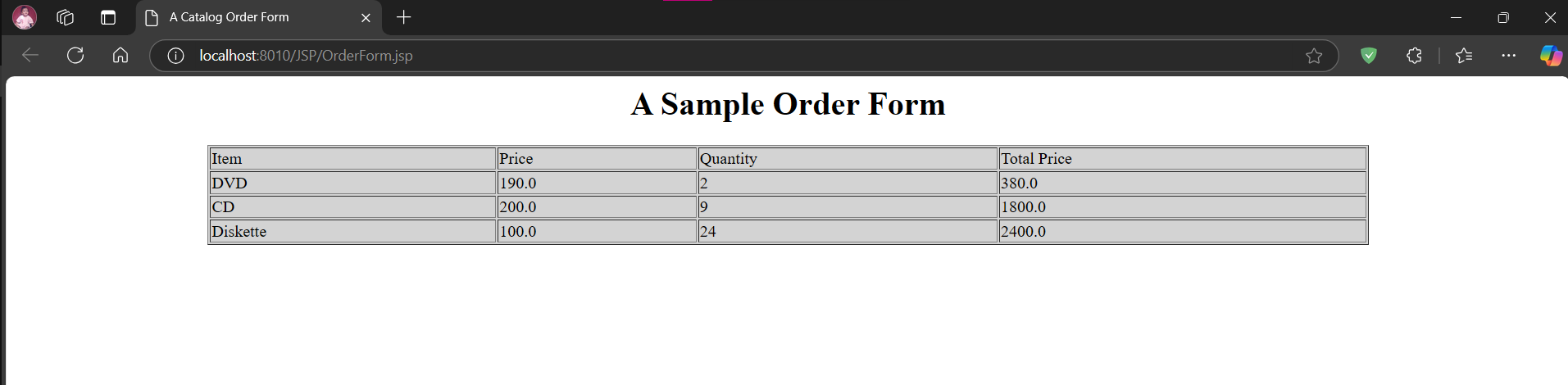
</TR>

<% } //end for loop %>

</TABLE>

</BODY>

</HTML>

****

**Practical NO 3.5:**

**Write a JSP program to perform Arithmetic operations such as Addition, Subtraction, Multiplication and Division. Design a HTML to accept two numbers in text box and radio buttons to display operations.**

**Index.html**

<!DOCTYPE html>

<html>

<head>

<title>Arithmetic Operations</title>

</head>

<body>

<h1>Arithmetic Operations</h1>

<form action="calculate.jsp" method="post">

<label for="num1">Enter First Number:</label>

<input type="text" id="num1" name="num1" required><br><br>

<label for="num2">Enter Second Number:</label>

<input type="text" id="num2" name="num2" required><br><br>

<label>Select Operation:</label><br>

<input type="radio" id="add" name="operation" value="add" required>

<label for="add">Addition</label><br>

<input type="radio" id="subtract" name="operation" value="subtract">

<label for="subtract">Subtraction</label><br>

<input type="radio" id="multiply" name="operation" value="multiply">

<label for="multiply">Multiplication</label><br>

<input type="radio" id="divide" name="operation" value="divide">

<label for="divide">Division</label><br><br>

<input type="submit" value="Calculate">

</form>

</body>

</html>

**Calculater.jsp**

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

<!DOCTYPE html>

<html>

<head>

<title>Calculation Result</title>

</head>

<body>

<h1>Calculation Result</h1>

<%

// Retrieve parameters from the form

String num1Str = request.getParameter("num1");

String num2Str = request.getParameter("num2");

String operation = request.getParameter("operation");

try {

double num1 = Double.parseDouble(num1Str);

double num2 = Double.parseDouble(num2Str);

double result = 0.0;

String operationName = "";

// Perform the selected operation

switch (operation) {

case "add":

result = num1 + num2;

operationName = "Addition";

break;

case "subtract":

result = num1 - num2;

operationName = "Subtraction";

break;

case "multiply":

result = num1 \* num2;

operationName = "Multiplication";

break;

case "divide":

if (num2 != 0) {

result = num1 / num2;

operationName = "Division";

} else {

operationName = "Division";

out.println("<p>Error: Division by zero is not allowed.</p>");

return;

}

break;

default:

out.println("<p>Error: Invalid operation selected.</p>");

return;

}

// Display the result

out.println("<p>Operation: " + operationName + "</p>");

out.println("<p>Result: " + result + "</p>");

} catch (NumberFormatException e) {

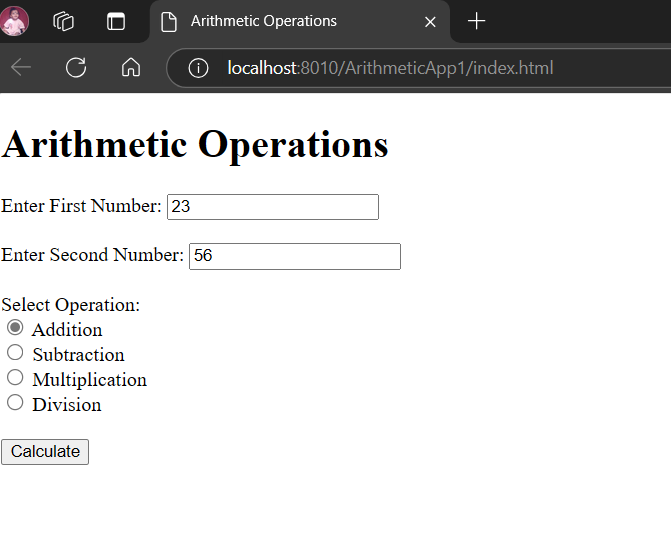
out.println("<p>Error: Please enter valid numbers.</p>");

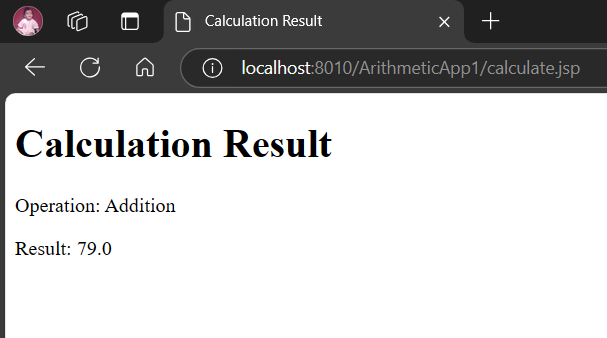
}

%>

</body>

</html>s

****

****

**Practical NO 4.1:**

**Define and illustrate the concept of entity mapping in JPA.**

* **Explain how JPA maps Java classes (entities) to database tables.**
* **Provide an example of an entity class with annotations and its corresponding database table schema.**

**Answer:-**

**Entity Mapping in JPA**

Entity mapping in Java Persistence API (JPA) refers to the process of linking Java objects (entities) to relational database tables. JPA uses annotations to define how the attributes of an entity class correspond to columns in a database table. The primary goal is to enable object-relational mapping (ORM), which simplifies database interactions in Java applications.

**Mapping Java Classes to Database Tables in JPA**

1. **Entity Class**:
   * Represented by a Java class annotated with @Entity.
   * Each instance of the class corresponds to a row in the database table.
2. **Table Mapping**:
   * The @Table annotation specifies the database table name. If omitted, the table name defaults to the entity class name.
3. **Attributes Mapping**:
   * Attributes are mapped to table columns using annotations like @Column.
   * Primary keys are defined using @Id. For auto-generation, @GeneratedValue is used.

**Example: Entity Class and Table Schema**

**Entity Class**

Java

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Column;

import jakarta.persistence.Table;

@Entity

@Table(name = "users")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(name = "username", nullable = false, unique = true)

private String username;

@Column(name = "email", nullable = false, unique = true)

private String email;

@Column(name = "password", nullable = false)

private String password;

// Getters and Setters

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

}

**Database Table Schema**

sql

Copy code

CREATE TABLE users (

id BIGINT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(255) NOT NULL UNIQUE,

email VARCHAR(255) NOT NULL UNIQUE,

password VARCHAR(255) NOT NULL

);

**Key Points:**

1. The @Entity annotation marks the class as an entity.
2. The @Table annotation specifies the table name (users).
3. Each attribute is annotated with @Column to map it to a database column, allowing custom settings like nullable or unique.
4. The primary key id is auto-generated using @GeneratedValue.

**Practical NO 4.2:**

**Describe the different types of relationships between entities (one-to-one, one-to-many, many-to-one, many-to-many).**

* **Explain how JPA represents these relationships using annotations.**
* **Provide code examples for each type of relationship.**

**Answer:-**

In JPA, entities can have different types of relationships with each other. These relationships correspond to associations in the underlying relational database schema. JPA provides annotations to represent these relationships, including @OneToOne, @OneToMany, @ManyToOne, and @ManyToMany.

Types of Relationships in JPA

One-to-One (@OneToOne)

One-to-Many (@OneToMany)

Many-to-One (@ManyToOne)

Many-to-Many (@ManyToMany)

1. One-to-One Relationship

In a one-to-one relationship, one entity is associated with exactly one instance of another entity.

JPA Annotation:

@OneToOne: Used on the field in one entity to map the relationship.

Example:

Entity 1: Person

java

Copy code

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.OneToOne;

@Entity

public class Person {

@Id

private Long id;

private String name;

@OneToOne

private Address address; // One person has one address

// Getters and setters

}

Entity 2: Address

java

Copy code

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@Entity

public class Address {

@Id

private Long id;

private String street;

// Getters and setters

}

Database Schema:

sql

Copy code

CREATE TABLE person (

id BIGINT PRIMARY KEY,

name VARCHAR(255),

address\_id BIGINT, -- Foreign key to address table

FOREIGN KEY (address\_id) REFERENCES address(id)

);

CREATE TABLE address (

id BIGINT PRIMARY KEY,

street VARCHAR(255)

);

2. One-to-Many Relationship

In a one-to-many relationship, one entity is associated with multiple instances of another entity.

JPA Annotation:

@OneToMany: Used on the field in the "one" side to represent a collection of entities.

@ManyToOne: Used on the field in the "many" side to represent the single reference back to the "one" side.

Example:

Entity 1: Customer

java

Copy code

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.OneToMany;

import java.util.List;

@Entity

public class Customer {

@Id

private Long id;

private String name;

@OneToMany(mappedBy = "customer")

private List<Order> orders; // One customer has many orders

// Getters and setters

}

Entity 2: Order

java

Copy code

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.ManyToOne;

@Entity

public class Order {

@Id

private Long id;

private String orderNumber;

@ManyToOne

private Customer customer; // Many orders belong to one customer

// Getters and setters

}

Database Schema:

sql

Copy code

CREATE TABLE customer (

id BIGINT PRIMARY KEY,

name VARCHAR(255)

);

CREATE TABLE order (

id BIGINT PRIMARY KEY,

order\_number VARCHAR(255),

customer\_id BIGINT, -- Foreign key to customer

FOREIGN KEY (customer\_id) REFERENCES customer(id)

);

3. Many-to-One Relationship

In a many-to-one relationship, multiple instances of one entity are associated with one instance of another entity.

JPA Annotation:

@ManyToOne: Used on the field in the "many" side to represent the single reference to the "one" side.

@OneToMany: Used on the "one" side to represent a collection of entities.

This is the reverse of the one-to-many relationship shown above.

Example:

Entity 1: Employee

java

Copy code

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.ManyToOne;

@Entity

public class Employee {

@Id

private Long id;

private String name;

@ManyToOne

private Department department; // Many employees belong to one department

// Getters and setters

}

Entity 2: Department

java

Copy code

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.OneToMany;

import java.util.List;

@Entity

public class Department {

@Id

private Long id;

private String name;

@OneToMany(mappedBy = "department")

private List<Employee> employees; // One department has many employees

// Getters and setters

}

Database Schema:

sql

Copy code

CREATE TABLE department (

id BIGINT PRIMARY KEY,

name VARCHAR(255)

);

CREATE TABLE employee (

id BIGINT PRIMARY KEY,

name VARCHAR(255),

department\_id BIGINT, -- Foreign key to department

FOREIGN KEY (department\_id) REFERENCES department(id)

);

4. Many-to-Many Relationship

In a many-to-many relationship, multiple instances of one entity are associated with multiple instances of another entity.

JPA Annotation:

@ManyToMany: Used on both entities to map the relationship. JPA automatically creates a join table to manage this association.

Example:

Entity 1: Student

java

Copy code

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.ManyToMany;

import java.util.List;

@Entity

public class Student {

@Id

private Long id;

private String name;

@ManyToMany

private List<Course> courses; // Many students can enroll in many courses

// Getters and setters

}

Entity 2: Course

java

Copy code

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.ManyToMany;

import java.util.List;

@Entity

public class Course {

@Id

private Long id;

private String courseName;

@ManyToMany(mappedBy = "courses")

private List<Student> students; // Many courses can have many students

// Getters and setters

}

Database Schema:

sql

Copy code

CREATE TABLE student (

id BIGINT PRIMARY KEY,

name VARCHAR(255)

);

CREATE TABLE course (

id BIGINT PRIMARY KEY,

course\_name VARCHAR(255)

);

CREATE TABLE student\_course (

student\_id BIGINT,

course\_id BIGINT,

PRIMARY KEY (student\_id, course\_id),

FOREIGN KEY (student\_id) REFERENCES student(id),

FOREIGN KEY (course\_id) REFERENCES course(id)

);

Summary of Annotations:

@OneToOne: Used for one-to-one relationships.

@OneToMany: Used for one-to-many relationships. It’s used on the "one" side of the relationship.

@ManyToOne: Used for many-to-one relationships. It’s used on the "many" side of the relationship.

@ManyToMany: Used for many-to-many relationships.

These annotations help JPA manage the relationships between Java objects and their corresponding database tables, ensuring data consistency and simplifying database operations.

**Practical NO 4.3:**

**Explain the use of embedded objects and collections in entity mapping.**

* **Show how to map complex data structures using @Embeddable and collection annotations.**
* **Provide code examples for embedding objects and using different collection types (e.g., List, Set).**

**Answer:-**

In JPA, embedded objects and collections allow you to model complex data structures within an entity, enabling reuse and better organization of data. These features are particularly useful for grouping related attributes into logical units and handling relationships between multiple entities more flexibly.

Embedded Objects in JPA

@Embeddable is used to define a class whose instances can be embedded in other entities. An embedded object does not have its own identity (i.e., it's not an entity with a primary key) but is part of an owning entity.

@Embedded is used on the field of an entity to indicate that it contains an embedded object.

Use Case: Embedding Objects

1. Embedding a Simple Object using @Embeddable and @Embedded

Address is an embeddable class that can be reused across multiple entities.

@Embeddable marks the Address class as embeddable, and @Embedded indicates that the Address field in the Person entity will contain the Address data.

Example:

Entity 1: Person

java

Copy code

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Embedded;

@Entity

public class Person {

@Id

private Long id;

private String name;

@Embedded

private Address address; // Embedded object

// Getters and setters

}

Embeddable Class: Address

java

Copy code

import jakarta.persistence.Embeddable;

@Embeddable

public class Address {

private String street;

private String city;

private String zipCode;

// Getters and setters

}

Database Schema:

sql

Copy code

CREATE TABLE person (

id BIGINT PRIMARY KEY,

name VARCHAR(255),

street VARCHAR(255),

city VARCHAR(255),

zip\_code VARCHAR(255)

);

In this example, the Address class is embedded into the Person class, and the fields of Address are directly mapped to the person table. The embedded object's fields (e.g., street, city, zipCode) appear as columns in the person table.

Collections in JPA

JPA also allows you to map collections of related entities or values, which is useful for modeling one-to-many and many-to-many relationships, as well as for storing lists or sets of primitive values or embeddable objects.

@ElementCollection is used to specify a collection of basic types or embeddable objects.

@CollectionTable is used to define the table that stores the collection.

@OneToMany, @ManyToMany can also be used to represent relationships between entities.

Use Case: Storing Collections of Simple Types (e.g., List, Set)

Example 1: List of String (e.g., storing phone numbers)

Entity: Customer

java

Copy code

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.ElementCollection;

import jakarta.persistence.CollectionTable;

import jakarta.persistence.Column;

import java.util.List;

@Entity

public class Customer {

@Id

private Long id;

private String name;

@ElementCollection

@CollectionTable(name = "customer\_phone\_numbers", joinColumns = @JoinColumn(name = "customer\_id"))

@Column(name = "phone\_number")

private List<String> phoneNumbers; // List of phone numbers

// Getters and setters

}

Database Schema:

sql

Copy code

CREATE TABLE customer (

id BIGINT PRIMARY KEY,

name VARCHAR(255)

);

CREATE TABLE customer\_phone\_numbers (

customer\_id BIGINT,

phone\_number VARCHAR(255),

FOREIGN KEY (customer\_id) REFERENCES customer(id)

);

In this case, the Customer entity has a collection of phone numbers stored in the customer\_phone\_numbers table. The @ElementCollection annotation tells JPA to manage the collection, and @CollectionTable specifies the name of the join table.

Use Case: Storing Collections of Embedded Objects (e.g., Set of Address)

Example 2: Set of Embedded Addresses

Entity: Customer

java

Copy code

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.ElementCollection;

import jakarta.persistence.CollectionTable;

import jakarta.persistence.Embedded;

import jakarta.persistence.JoinColumn;

import java.util.Set;

@Entity

public class Customer {

@Id

private Long id;

private String name;

@ElementCollection

@CollectionTable(name = "customer\_addresses", joinColumns = @JoinColumn(name = "customer\_id"))

private Set<Address> addresses; // Set of embedded addresses

// Getters and setters

}

Embeddable Class: Address

java

Copy code

import jakarta.persistence.Embeddable;

@Embeddable

public class Address {

private String street;

private String city;

private String zipCode;

// Getters and setters

}

Database Schema:

sql

Copy code

CREATE TABLE customer (

id BIGINT PRIMARY KEY,

name VARCHAR(255)

);

CREATE TABLE customer\_addresses (

customer\_id BIGINT,

street VARCHAR(255),

city VARCHAR(255),

zip\_code VARCHAR(255),

FOREIGN KEY (customer\_id) REFERENCES customer(id)

);

Here, the Customer entity has a Set of Address objects, each stored in the customer\_addresses table. The @ElementCollection annotation is used to indicate that the addresses collection is a simple collection of embeddable objects.

Summary of Key Annotations

@Embeddable: Marks a class as embeddable. The fields of this class will be embedded in another entity.

@Embedded: Used to indicate that the fields of an embeddable object are part of the owning entity.

@ElementCollection: Used to map collections of basic types or embeddable objects.

@CollectionTable: Defines the table used to store the collection, specifying details like the join column and table name.

@JoinColumn: Specifies the foreign key column when associating collections with entities.

These annotations allow you to map complex data structures like collections and embedded objects in JPA, enabling the modeling of rich, hierarchical data.

**Practical NO 4.4:**

**CURD Operations using JPA**

**Create a JPA application to perform CRUD operations on a simple entity (e.g., Product).**

* **Include methods for creating, retrieving, updating, and deleting Product entities.**
* **Demonstrate the use of EntityManager for persistence operations.**

**Answer:-**

**Product.java**

package com.example.demo.entity;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

@Entity

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private Double price;

// Getters and setters

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Double getPrice() {

return price;

}

public void setPrice(Double price) {

this.price = price;

}

}

**ProductService.java**

package com.example.demo.service;

import com.example.demo.entity.Product;

import jakarta.persistence.EntityManager;

import jakarta.persistence.PersistenceContext;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service

@Transactional

public class ProductService {

@PersistenceContext

private EntityManager entityManager;

public Product createProduct(Product product) {

entityManager.persist(product);

return product;

}

public Product getProductById(Long id) {

return entityManager.find(Product.class, id);

}

public List<Product> getAllProducts() {

return entityManager.createQuery("SELECT p FROM Product p", Product.class).getResultList();

}

public Product updateProduct(Long id, Product updatedProduct) {

Product product = entityManager.find(Product.class, id);

if (product != null) {

product.setName(updatedProduct.getName());

product.setPrice(updatedProduct.getPrice());

entityManager.merge(product);

}

return product;

}

public void deleteProduct(Long id) {

Product product = entityManager.find(Product.class, id);

if (product != null) {

entityManager.remove(product);

}

}

}

**ProductController.java**

package com.example.demo.controller;

import com.example.demo.entity.Product;

import com.example.demo.service.ProductService;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/products")

public class ProductController {

private final ProductService productService;

public ProductController(ProductService productService) {

this.productService = productService;

}

@PostMapping

public Product createProduct(@RequestBody Product product) {

return productService.createProduct(product);

}

@GetMapping("/{id}")

public Product getProductById(@PathVariable Long id) {

return productService.getProductById(id);

}

@GetMapping

public List<Product> getAllProducts() {

return productService.getAllProducts();

}

@PutMapping("/{id}")

public Product updateProduct(@PathVariable Long id, @RequestBody Product product) {

return productService.updateProduct(id, product);

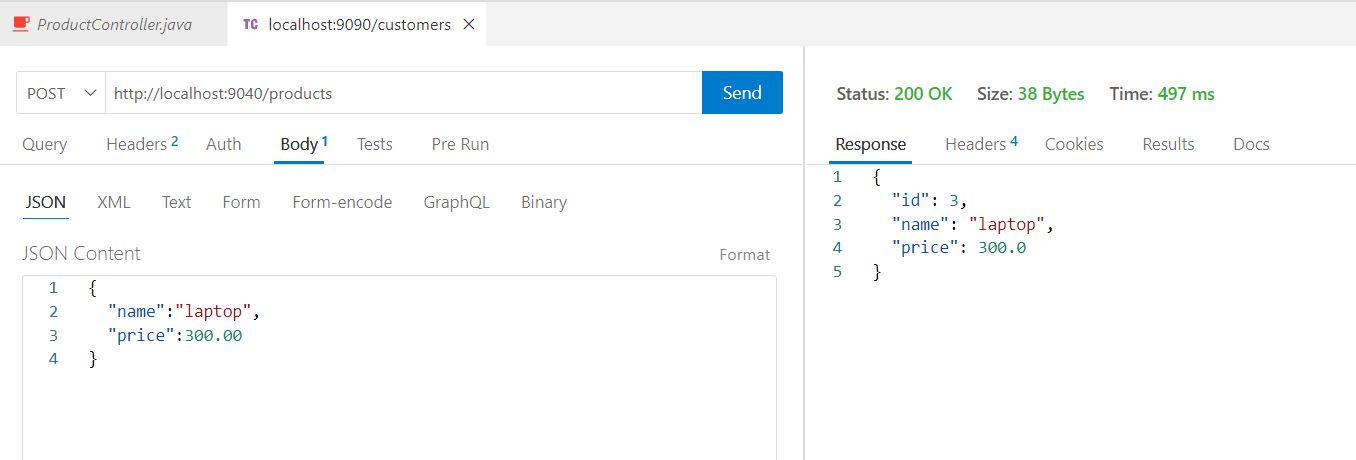
}

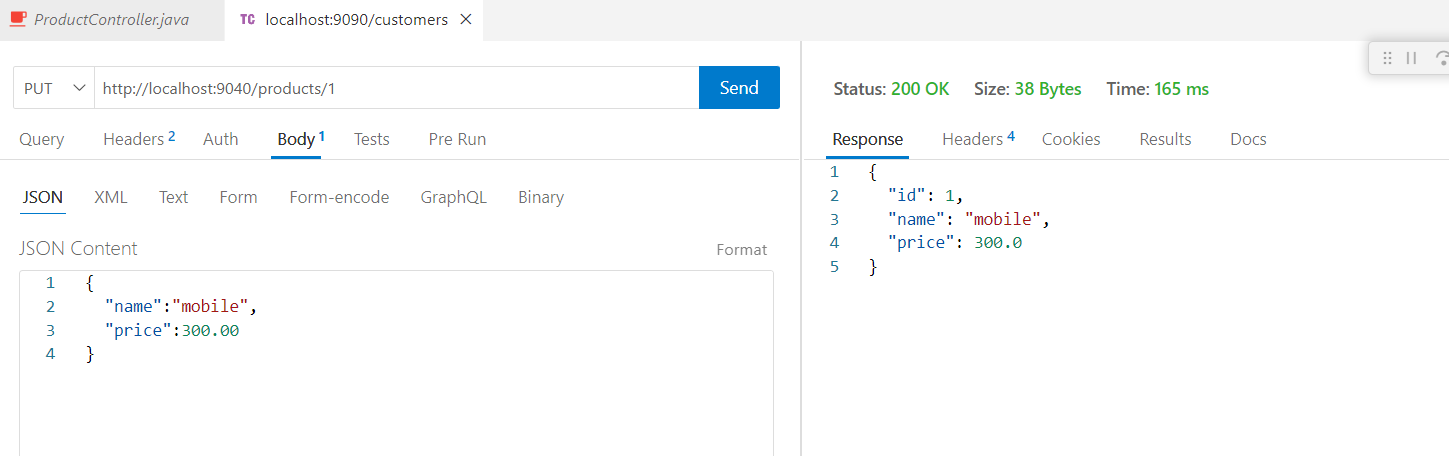
@DeleteMapping("/{id}")

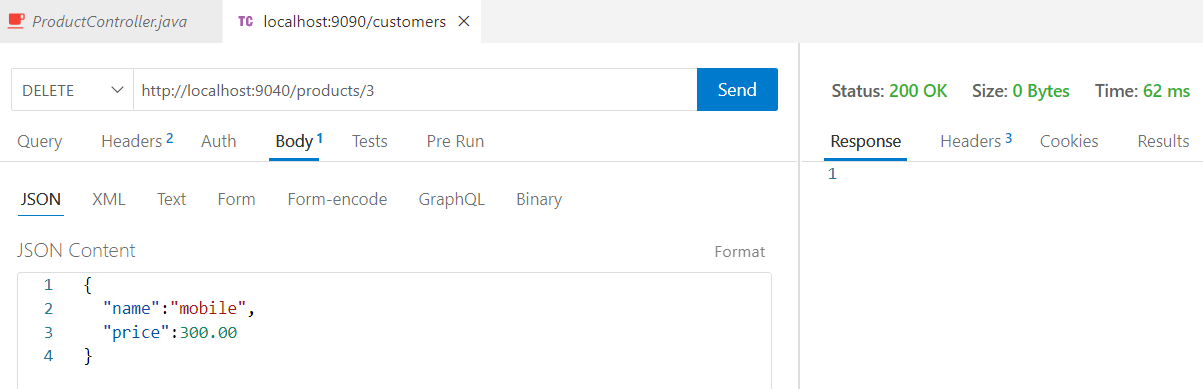
public void deleteProduct(@PathVariable Long id) {

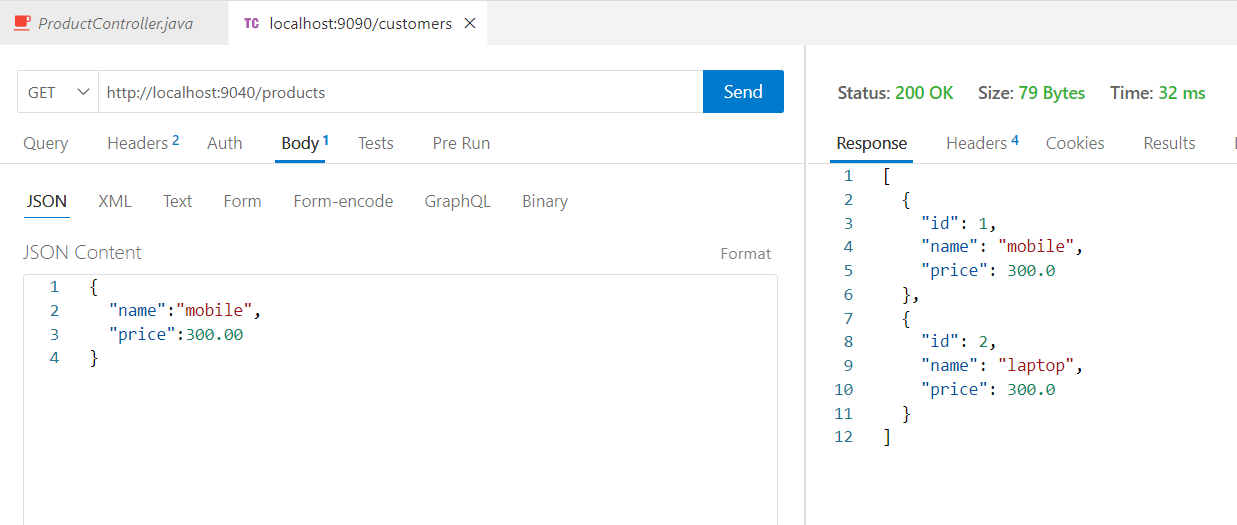
productService.deleteProduct(id);

}





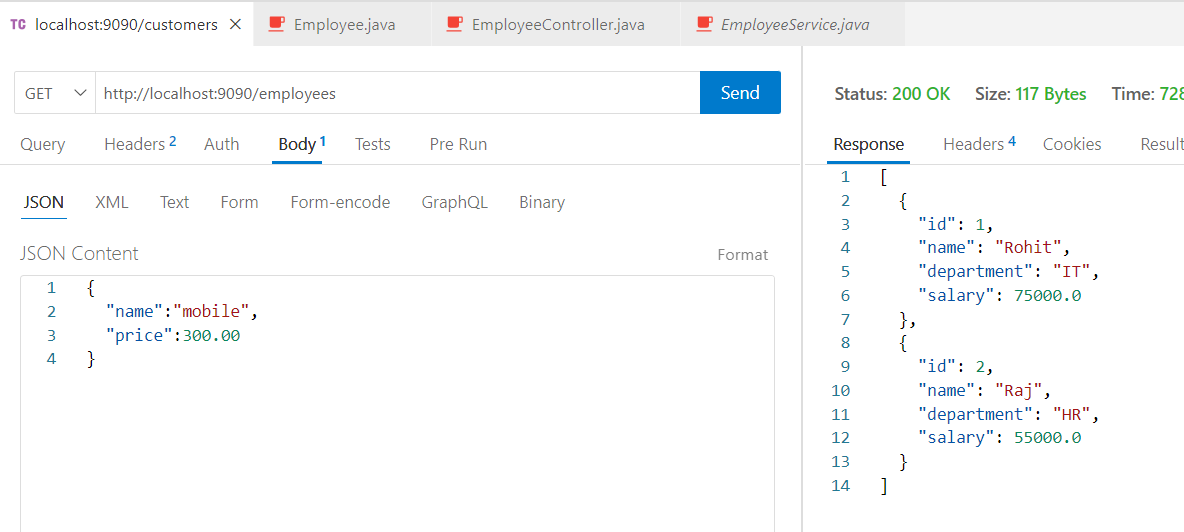




**Practical NO 5.1:**

**Configure a Spring Boot application to connect to a specific MySQL database without explicitly defining beans for connection pool, DataSource, etc.**

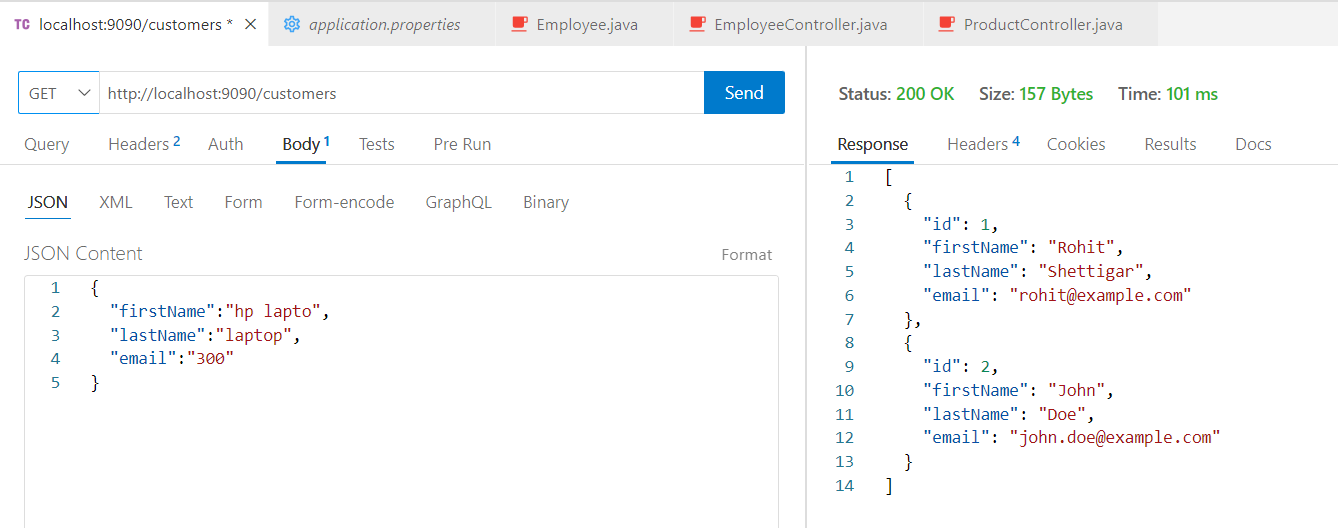
* **Use only the necessary dependencies and demonstrate how auto-configuration sets up the connection.**
* **Explore using application.properties to customize connection details (URL, username, password).**

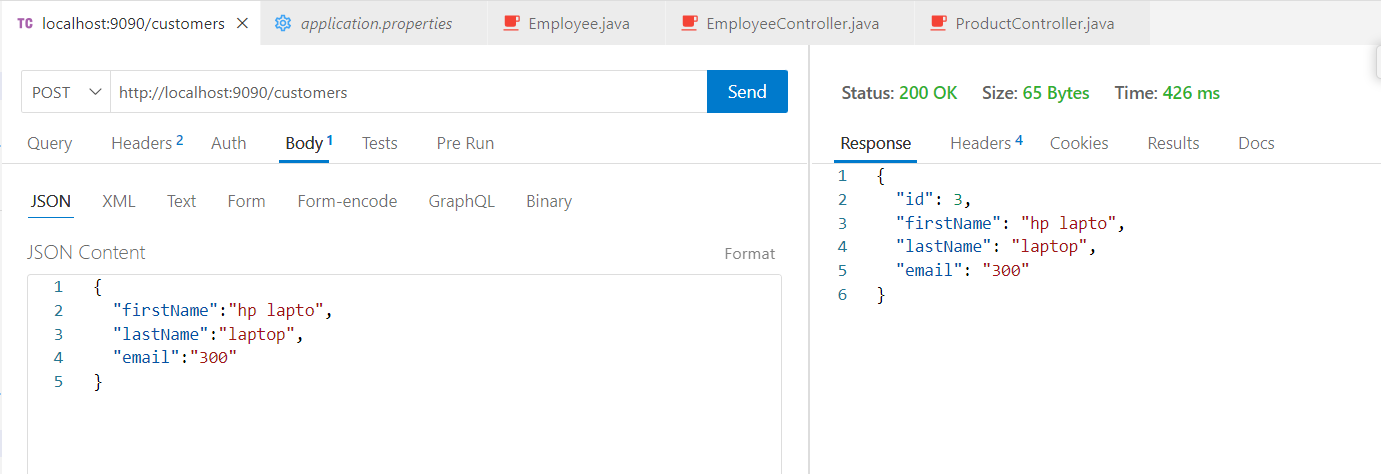
****

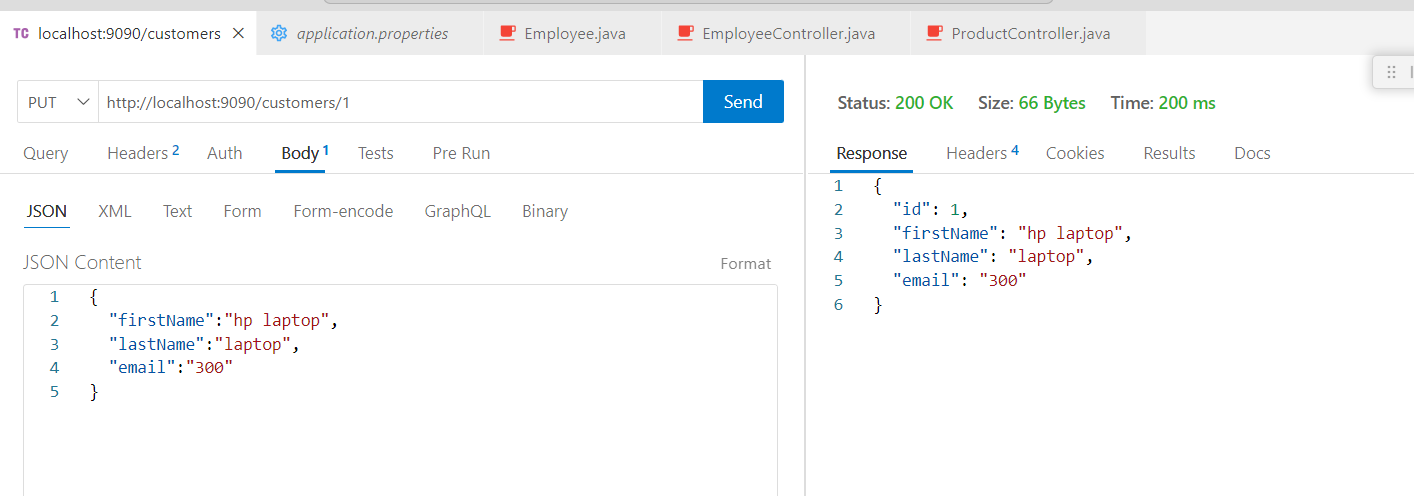
**Practical NO 5.2:**

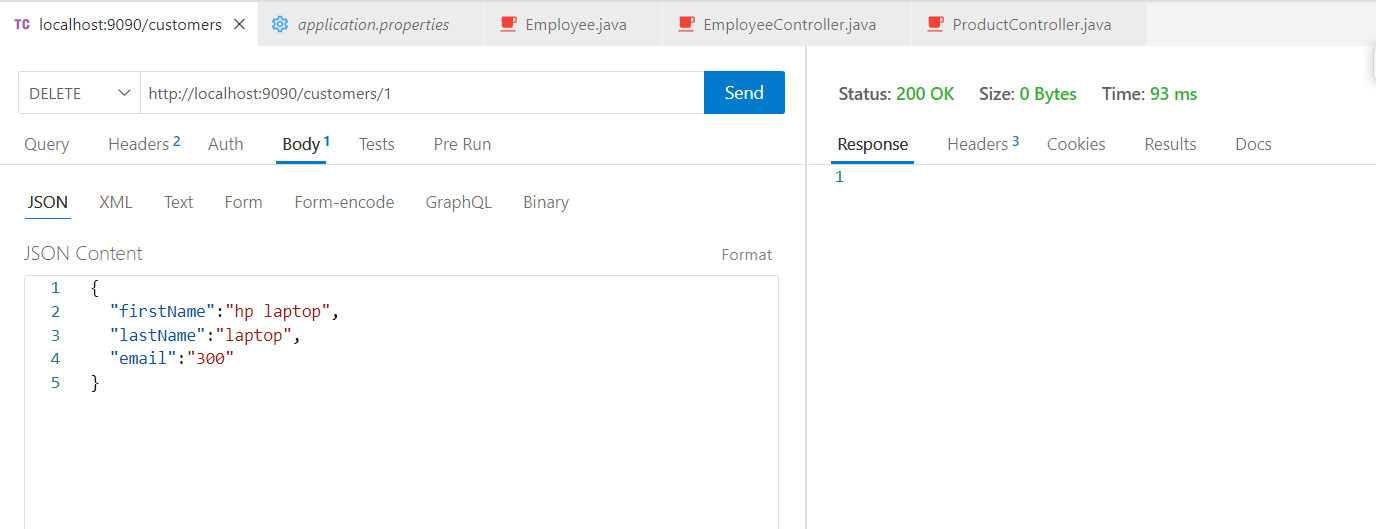
**Create a Spring Boot application that utilizes JPA repositories. Persist and retrieve data from an in-memory database (e.g., H2) without manual configuration.**

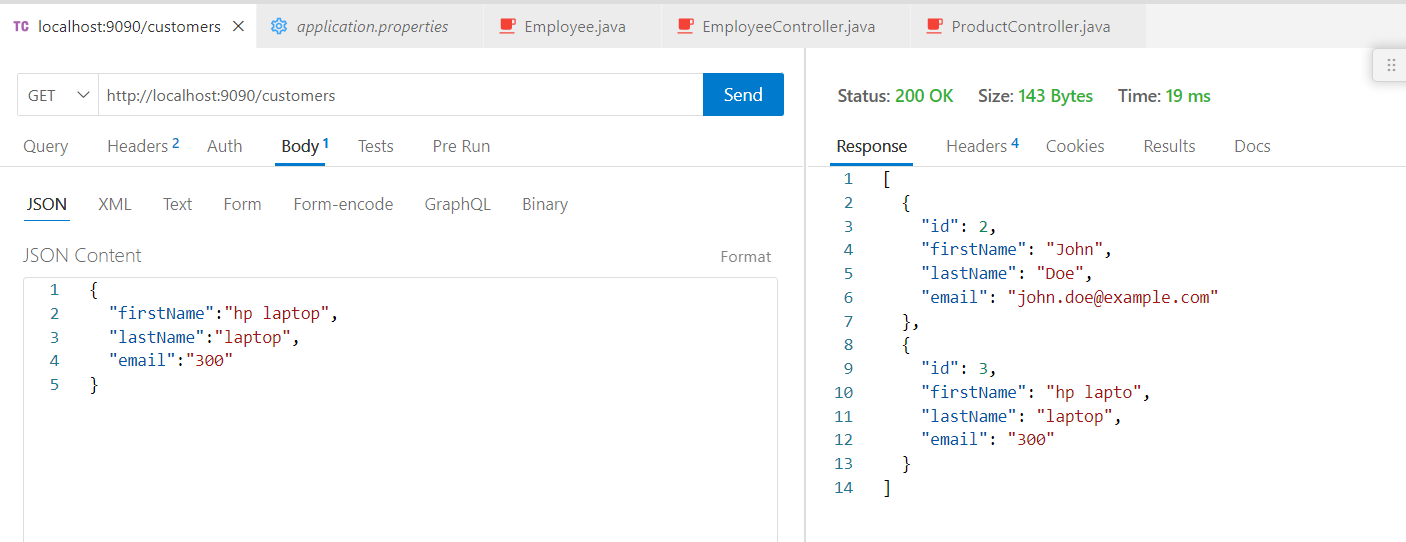
* **Focus on the simplicity achieved through auto-configuration for JPA and repositories.**
* **Implement basic CRUD operations using JPA repositories.**

****

****





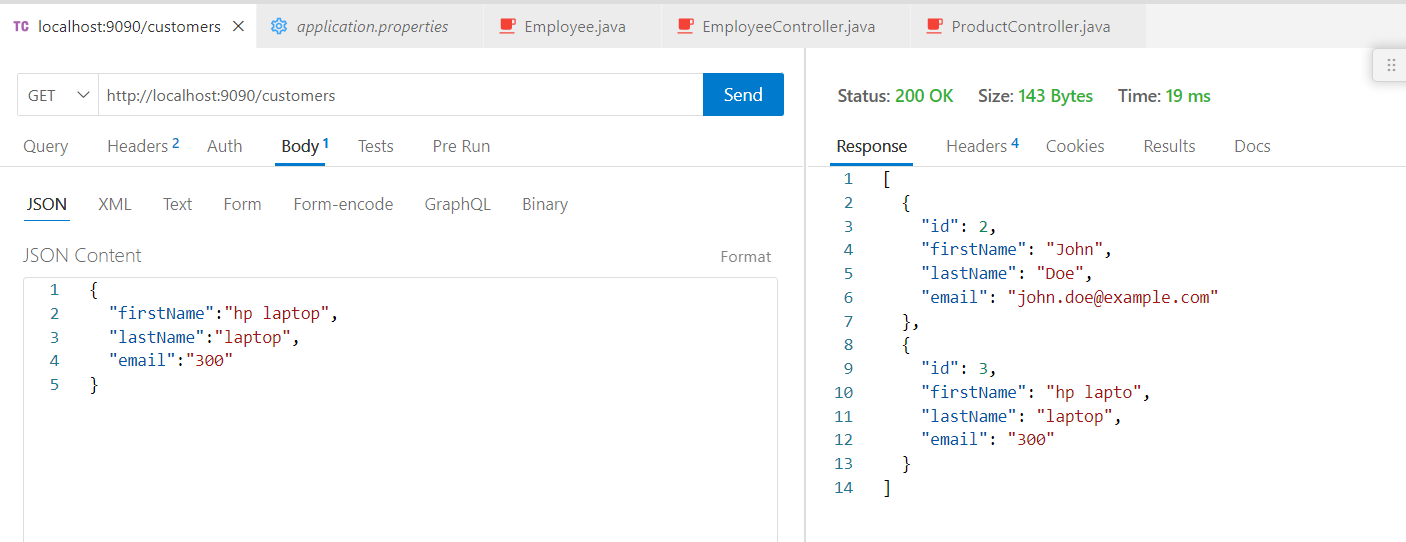


**Practical NO 5.3:**

**Develop a Spring Boot application with a RESTful API that exposes an endpoint to retrieve a list of products.**

* **Utilize Spring MVC annotations like @RestController and @GetMapping.**
* **Implement a service layer to interact with a product repository (in-memory or database).**
* **Return the list of products in JSON format using @ResponseBody.**

**Answer:-**

****

**Practical NO 6.1:**

**Write a Hibernate program to create the product table (product id,product name,product category,product price) and delete the specific product record.(using through the product id).**

**Practical NO 6.2:**

**Write a Hibernate program to update the product price data from product table.(Using HQL).**

**Practical NO 6.3:**

**Write a Hibernate Program for product information and display the information by selecting the details from product database table.**

**Answer:-**

**Product.java**

package com.example;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

// @Table(name = "product")

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int productId;

private String productName;

private String productCategory;

private double productPrice;

// Getters and Setters

public int getProductId() {

return productId;

}

public void setProductId(int productId) {

this.productId = productId;

}

public String getProductName() {

return productName;

}

public void setProductName(String productName) {

this.productName = productName;

}

public String getProductCategory() {

return productCategory;

}

public void setProductCategory(String productCategory) {

this.productCategory = productCategory;

}

public double getProductPrice() {

return productPrice;

}

public void setProductPrice(double productPrice) {

this.productPrice = productPrice;

}

}

**ProductCrud.java**

package com.example;

import org.hibernate.Session;

import org.hibernate.Transaction;

import java.util.List;

import jakarta.persistence.EntityManager;

import jakarta.persistence.EntityTransaction;

import jakarta.persistence.Query;

import jakarta.persistence.TypedQuery;

public class ProductCRUD {

public static void main(String[] args) {

// Create a new product object

Product product = new Product();

product.setProductName("Laptop");

product.setProductCategory("Electronics");

product.setProductPrice(1200.00);

// Add product to the database

addProduct(product);

updateProductPrice(1, 299.99f);

displayProducts();

// Delete product by ID

deleteProductById(1); // Assuming the product ID is 1

}

// Method to add a product

public static void addProduct(Product product) {

Session session = HibernateUtil.getSessionFactory().getCurrentSession();

Transaction transaction = null;

try {

// Start a transaction

transaction = session.beginTransaction();

// Use persist() to save the product object

session.persist(product);

// Commit the transaction

transaction.commit();

System.out.println("Product added successfully!");

} catch (Exception e) {

if (transaction != null)

transaction.rollback();

e.printStackTrace();

} finally {

session.close();

}

}

// Method to delete product by ID

// Method to delete product by ID

public static void deleteProductById(int productId) {

Session session = HibernateUtil.getSessionFactory().getCurrentSession();

Transaction transaction = null;

try {

// Start a transaction

transaction = session.beginTransaction();

// Get the product object by its ID

Product product = session.get(Product.class, productId);

if (product != null) {

// Use remove() (JPA standard) to delete the product

session.remove(product);

System.out.println("Product deleted successfully!");

} else {

System.out.println("Product not found!");

}

// Commit the transaction

transaction.commit();

} catch (Exception e) {

if (transaction != null)

transaction.rollback();

e.printStackTrace();

} finally {

session.close();

}

}

/// update

public static void updateProductPrice(int productId, float newPrice) {

EntityManager entityManager = HibernateUtil.getEntityManagerFactory().createEntityManager();

EntityTransaction transaction = entityManager.getTransaction();

try {

transaction.begin(); // Start the transaction

// JPQL query for updating product price

String jpql = "UPDATE Product p SET p.productPrice = :newPrice WHERE p.productId = :productId";

Query query = entityManager.createQuery(jpql); // Create query without type parameter

query.setParameter("newPrice", newPrice);

query.setParameter("productId", productId);

// Execute update

int result = query.executeUpdate();

if (result > 0) {

System.out.println("Product price updated successfully!");

} else {

System.out.println("Product not found!");

}

transaction.commit(); // Commit the transaction

} catch (Exception e) {

if (transaction.isActive()) {

transaction.rollback();

}

e.printStackTrace();

} finally {

entityManager.close(); // Close the entity manager

}

}

public static void displayProducts() {

EntityManager entityManager = HibernateUtil.getEntityManagerFactory().createEntityManager();

EntityTransaction transaction = entityManager.getTransaction();

try {

transaction.begin();

// JPQL query to fetch all product details

String jpql = "SELECT p FROM Product p";

TypedQuery<Product> query = entityManager.createQuery(jpql, Product.class);

List<Product> products = query.getResultList();

// Display product information

if (products.isEmpty()) {

System.out.println("No products found!");

} else {

System.out.println("Product Details:");

for (Product product : products) {

System.out.println("ID: " + product.getProductId() +

", Name: " + product.getProductName() +

", Category: " + product.getProductCategory() +

", Price: " + product.getProductPrice());

}

}

transaction.commit();

} catch (Exception e) {

if (transaction.isActive()) {

transaction.rollback();

}

e.printStackTrace();

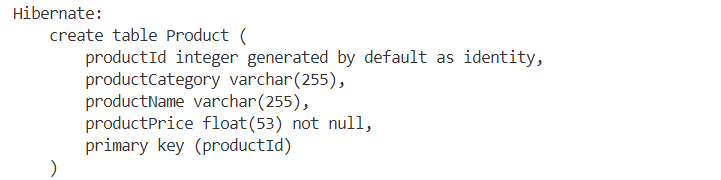
} finally {

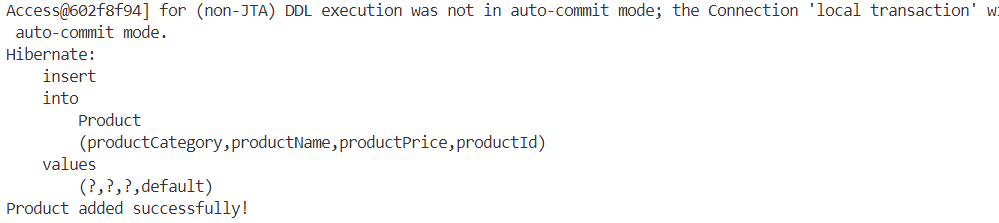
entityManager.close();

}

}

}

****

****



