Experiment-7

Student Name: Prithvi Singh
UID: 22BCS12368
Branch: CSE
Section/Group: IOT-618/A
Semester: 6th
Date of Performance: 21/03/25
Subject Name: Java Lab
Subject Code: 22CSH-359

Problem-1 (Easy)

1. Aim:

Create a Java program to connect to a MySQL database and fetch data from a single table. The program should:
Use DriverManager and Connection objects.
Retrieve and display all records from a table named Employee with columns EmpID, Name, and Salary.

2. Implementation/Code:

```
import java.sql.*;


public class SQLiteConnection {
    public static void main(String[] args) {
        String url = "jdbc:sqlite:employees.db"; // Your SQLite database file


        try {
            // Load SQLite JDBC Driver
            Class.forName("org.sqlite.JDBC");


            // Establish Connection
            Connection conn = DriverManager.getConnection(url);
```

```
            System.out.println("Connected to SQLite database!");

            // Create Employee Table if it doesn't exist
            String createTableSQL = "CREATE TABLE IF NOT EXISTS Employee ("
                  + "EmpID INTEGER PRIMARY KEY AUTOINCREMENT, "
                  + "Name TEXT NOT NULL, "
                  + "Salary REAL NOT NULL"
                  + ");";

            Statement stmt = conn.createStatement();
            stmt.execute(createTableSQL);
            System.out.println("Employee table is ready.");

            // Insert Sample Data (if table was empty)
            String insertSQL = "INSERT INTO Employee (Name, Salary) "
                        + "SELECT 'Alice', 50000.00 WHERE NOT EXISTS (SELECT 1
FROM Employee);";
            stmt.execute(insertSQL);

            // Query Employee Table
            ResultSet rs = stmt.executeQuery("SELECT * FROM Employee");

            // Display Results
            System.out.println("Employee Details:");
            while (rs.next()) {
               System.out.println("EmpID: " + rs.getInt("EmpID") +
                           ", Name: " + rs.getString("Name") +
                           ", Salary: " + rs.getDouble("Salary"));
```

```
            }

            // Close resources
            rs.close();
            stmt.close();
            conn.close();
            System.out.println("Database operation completed.");

        } catch (ClassNotFoundException e) {
            System.out.println("SQLite JDBC Driver not found. Ensure sqlite-jdbc-3.49.1.0.jar is in classpath.");
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

3. Output:

```
Connected to SQLite database!
Employee table is ready.
Employee Details:
EmpID: 1, Name: Alice, Salary: 50000.0
Database operation completed.
```

Problem-2 (Medium)

1. Aim:

Build a program to perform CRUD operations (Create, Read, Update, Delete) on a database table Product with
columns:
ProductID, ProductName, Price, and Quantity.
The program should include:
Menu-driven options for each operation.
Transaction handling to ensure data integrity.

2. Implementation/Code:

```
import java.sql.*;
import java.util.Scanner;

public class ProductCRUD {
   static final String URL = "jdbc:sqlite:products.db"; // SQLite database file

   public static void main(String[] args) {
      try (Connection conn = DriverManager.getConnection(URL);
           Scanner scanner = new Scanner(System.in)) {

         Class.forName("org.sqlite.JDBC");
         createTable(conn); // Ensure the table exists

         while (true) {
            System.out.println("\n--- Product CRUD Menu ---");
            System.out.println("1. Create Product");
            System.out.println("2. Read Products");
            System.out.println("3. Update Product");
            System.out.println("4. Delete Product");
            System.out.println("5. Exit");
            System.out.print("Enter choice: ");
            int choice = scanner.nextInt();
```

```
                switch (choice) {
                    case 1:
                        createProduct(conn, scanner);
                        break;
                    case 2:
                        readProducts(conn);
                        break;
                    case 3:
                        updateProduct(conn, scanner);
                        break;
                    case 4:
                        deleteProduct(conn, scanner);
                        break;
                    case 5:
                        System.out.println("Exiting...");
                        return;
                    default:
                        System.out.println("Invalid choice, try again.");
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    private static void createTable(Connection conn) throws SQLException {
        String sql = "CREATE TABLE IF NOT EXISTS Product ("
                + "ProductID INTEGER PRIMARY KEY AUTOINCREMENT, "
                + "ProductName TEXT NOT NULL, "
                + "Price REAL NOT NULL, "
                + "Quantity INTEGER NOT NULL);";
        try (Statement stmt = conn.createStatement()) {
            stmt.execute(sql);
        }
    }
    private static void createProduct(Connection conn, Scanner scanner) throws
SQLException {
        System.out.print("Enter Product Name: ");
        scanner.nextLine(); // Consume newline
```

```
        String name = scanner.nextLine();
        System.out.print("Enter Price: ");
        double price = scanner.nextDouble();
        System.out.print("Enter Quantity: ");
        int quantity = scanner.nextInt();

        String sql = "INSERT INTO Product (ProductName, Price, Quantity) VALUES (?,
?, ?)";
        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, name);
            pstmt.setDouble(2, price);
            pstmt.setInt(3, quantity);
            pstmt.executeUpdate();
            System.out.println("Product added successfully!");
        }
    private static void readProducts(Connection conn) throws SQLException {
        String sql = "SELECT * FROM Product";
        try (Statement stmt = conn.createStatement(); ResultSet rs =
stmt.executeQuery(sql)) {
            System.out.println("\n--- Product List ---");
            while (rs.next()) {
                System.out.println("ID: " + rs.getInt("ProductID") +
                                   ", Name: " + rs.getString("ProductName") +
                                   ", Price: " + rs.getDouble("Price") +
                                   ", Quantity: " + rs.getInt("Quantity"));
            }
        }
    }
    private static void updateProduct(Connection conn, Scanner scanner) throws
SQLException {
        System.out.print("Enter Product ID to update: ");
        int id = scanner.nextInt();
        System.out.print("Enter new Price: ");
        double price = scanner.nextDouble();
        System.out.print("Enter new Quantity: ");
        int quantity = scanner.nextInt();
```

```
        String sql = "UPDATE Product SET Price = ?, Quantity = ? WHERE ProductID =
?";
        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setDouble(1, price);
            pstmt.setInt(2, quantity);
            pstmt.setInt(3, id);
            int rowsAffected = pstmt.executeUpdate();
            System.out.println(rowsAffected > 0 ? "Product updated successfully!" : "Product
not found.");
        }
    }
    private static void deleteProduct(Connection conn, Scanner scanner) throws
SQLException {
        System.out.print("Enter Product ID to delete: ");
        int id = scanner.nextInt();
        String sql = "DELETE FROM Product WHERE ProductID = ?";
        try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, id);
            int rowsAffected = pstmt.executeUpdate();
            System.out.println(rowsAffected > 0 ? "Product deleted successfully!" : "Product
not found.");
        }
    }
}
```

3. Output:

```
--- Product CRUD Menu ---
1. Create Product
2. Read Products
3. Update Product
4. Delete Product
5. Exit
Enter choice: 1
Enter Product Name: Apple
Enter Price: 500
Enter Quantity: 25
Product added successfully!
```

Problem-3 (Hard)

1. Aim:

Develop a Java application using JDBC and MVC architecture to manage student data.
The application should:
Use a Student class as the model with fields like StudentID, Name, Department, and Marks.
Include a database table to store student data.
Allow the user to perform CRUD operations through a simple menu-driven view.
Implement database operations in a separate controller class.

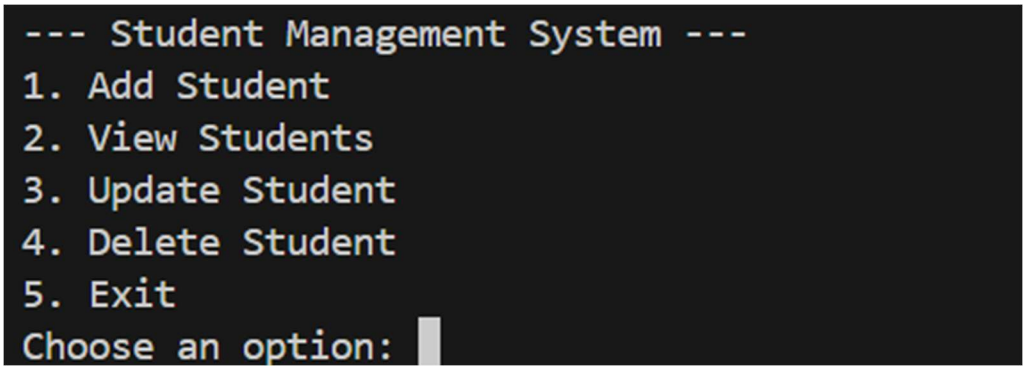
2. Implementation/Code:

```
import java.util.Scanner;

public class StudentManagementApp {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      StudentDAO dao = new StudentDAO();
      StudentView view = new StudentView();

      while (true) {
         System.out.println("\n--- Student Management System ---");
         System.out.println("1. Add Student");
         System.out.println("2. View Students");
         System.out.println("3. Update Student");
         System.out.println("4. Delete Student");
         System.out.println("5. Exit");
         System.out.print("Choose an option: ");
         int choice = scanner.nextInt();

         switch (choice) {
            case 1:
               scanner.nextLine(); // Consume newline
               System.out.print("Enter Name: ");
               String name = scanner.nextLine();
               System.out.print("Enter Department: ");
               String department = scanner.nextLine();
```

```
                System.out.print("Enter Marks: ");
                double marks = scanner.nextDouble();
                dao.addStudent(new Student(0, name, department, marks));
                break;

            case 2:
                view.displayStudents(dao.getAllStudents());
                break;

            case 3:
                System.out.print("Enter Student ID to update: ");
                int id = scanner.nextInt();
                scanner.nextLine(); // Consume newline
                System.out.print("Enter New Name: ");
                String newName = scanner.nextLine();
                System.out.print("Enter New Department: ");
                String newDepartment = scanner.nextLine();
                System.out.print("Enter New Marks: ");
                double newMarks = scanner.nextDouble();
                dao.updateStudent(id, newName, newDepartment, newMarks);
                break;

            case 4:
                System.out.print("Enter Student ID to delete: ");
                int deleteId = scanner.nextInt();
                dao.deleteStudent(deleteId);
                break;

            case 5:
                System.out.println("Exiting...");
                scanner.close();
                return;

            default:
                System.out.println("Invalid choice, please try again.");
        }
    }
}
```

```
}
```

3. Output:


--- Student Management System ---
1. Add Student
2. View Students
3. Update Student
4. Delete Student
5. Exit
Choose an option: