Experiment 6

Student Name: Vikash Singh **UID: 22BCS10491**
Branch: BE-CSE **Section/Group: 903-B**
Semester: 6th **Date of Performance:**
Subject Name: Project based Learning Java **Subject Code: 22CSH-359**

Problem :- 1(Easy-Level)

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.Objective:

· Use DriverManager and Connection objects to establish a connection.

· Retrieve and display all records from a table named Employee with columns EmpID, Name, and Salary.

3. Algorithm:

· Define database connection details (URL, username, password).

· Establish a connection using DriverManager.

· Create a statement to execute the SQL query.

· Retrieve data using ResultSet and display it.

· Close the connection.

4.Implementation :

```
import java.sql.*;

public class MySQLDatabaseConnection {

    public static void main(String[] args) {

        // Database credentials
```

```
        String url = "jdbc:mysql://localhost:3306/your_database"; // Replace
'your_database' with your DB name

        String user = "your_username"; // Replace with your MySQL username

        String password = "your_password"; // Replace with your MySQL password


        // SQL query to fetch data from Employee table

        String query = "SELECT EmpID, Name, Salary FROM Employee";


        try (

            // Establishing the connection

            Connection conn = DriverManager.getConnection(url, user, password);

            // Creating a statement

            Statement stmt = conn.createStatement();

            // Executing the query

            ResultSet rs = stmt.executeQuery(query)

        ) {

            // Displaying the results

            System.out.println("EmpID | Name | Salary");

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

            while (rs.next()) {

                int empId = rs.getInt("EmpID");

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

```
            double salary = rs.getDouble("Salary");
            System.out.println(empId + " | " + name + " | " + salary);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
  }
}
```

5.Output:

```
EmpID | Name | Salary
------------------------
101   | John Doe   | 50000.0
102   | Jane Smith | 60000.0
103   | Alice Brown| 55000.0
```

Problem:- 2(Medium-level)

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.Objective:

· Implement menu-driven options for each CRUD operation.

· Ensure transaction handling to maintain data integrity.

· Use JDBC to interact with the MySQL database.

3.Algorithm:

· Establish a database connection.

· Implement methods for inserting, retrieving, updating, and deleting product records.

· Use a loop-driven menu to allow user selection.

· Use transaction handling to ensure data consistency.

· Close resources properly after execution.

4.Implementation/Code:

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


public class ProductCRUD {
    static final String URL = "jdbc:mysql://localhost:3306/your_database";
    static final String USER = "your_username";
    static final String PASSWORD = "your_password";


    public static void main(String[] args) {
        try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD))
{
            Scanner scanner = new Scanner(System.in);
            boolean exit = false;
            while (!exit) {
                System.out.println("\n1. Create Product\n2. Read Products\n3. Update
Product\n4. Delete Product\n5. Exit\nChoose an option: ");
                int choice = scanner.nextInt();
                scanner.nextLine();
                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: exit = true; break;
                default: System.out.println("Invalid choice, try again.");
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

    private static void createProduct(Connection conn, Scanner scanner) throws
SQLException {
        System.out.print("Enter Product Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter Price: ");
        double price = scanner.nextDouble();
        System.out.print("Enter Quantity: ");
        int quantity = scanner.nextInt();

        String query = "INSERT INTO Product (ProductName, Price, Quantity) VALUES
(?, ?, ?)";
        try (PreparedStatement pstmt = conn.prepareStatement(query)) {
            conn.setAutoCommit(false);
            pstmt.setString(1, name);
            pstmt.setDouble(2, price);
            pstmt.setInt(3, quantity);
            pstmt.executeUpdate();
```

```
            conn.commit();
            System.out.println("Product added successfully.");
        } catch (SQLException e) {
            conn.rollback();
            e.printStackTrace();
        }
    }

    private static void readProducts(Connection conn) throws SQLException {
        String query = "SELECT * FROM Product";
        try (Statement stmt = conn.createStatement(); ResultSet rs =
stmt.executeQuery(query)) {
            System.out.println("ProductID | ProductName | Price | Quantity");
            while (rs.next()) {
                System.out.println(rs.getInt("ProductID") + " | " + rs.getString("ProductName")
+ " | " + rs.getDouble("Price") + " | " + 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();
        scanner.nextLine();
        System.out.print("Enter new Product Name: ");
        String name = scanner.nextLine();
        System.out.print("Enter new Price: ");
        double price = scanner.nextDouble();
```

```
        System.out.print("Enter new Quantity: ");
        int quantity = scanner.nextInt();

        String query = "UPDATE Product SET ProductName=?, Price=?, Quantity=?
WHERE ProductID=?";
        try (PreparedStatement pstmt = conn.prepareStatement(query)) {
            conn.setAutoCommit(false);
            pstmt.setString(1, name);
            pstmt.setDouble(2, price);
            pstmt.setInt(3, quantity);
            pstmt.setInt(4, id);
            pstmt.executeUpdate();
            conn.commit();
            System.out.println("Product updated successfully.");
        } catch (SQLException e) {
            conn.rollback();
            e.printStackTrace();
        }
    }

    private static void deleteProduct(Connection conn, Scanner scanner) throws
SQLException {
        System.out.print("Enter Product ID to delete: ");
        int id = scanner.nextInt();

        String query = "DELETE FROM Product WHERE ProductID=?";
        try (PreparedStatement pstmt = conn.prepareStatement(query)) {
            conn.setAutoCommit(false);
```

```
            pstmt.setInt(1, id);
            pstmt.executeUpdate();
            conn.commit();
            System.out.println("Product deleted successfully.");
        } catch (SQLException e) {
            conn.rollback();
            e.printStackTrace();
        }
    }
}
```

5. Output:

```
1. Create Product
2. Read Products
3. Update Product
4. Delete Product
5. Exit
Choose an option:
```

Problem:- 3(Hard-level)

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.Objective:

· Implement the MVC pattern with a Student model, Controller for database operations, and a View for user interaction.

· Use JDBC to interact with a database table storing student information.

· Provide menu-driven options for CRUD operations.

· Ensure data integrity through transaction handling.

3.Algorithm:

· Define the Student class with fields: StudentID, Name, Department, and Marks.

· Create a database connection using JDBC.

· Implement a StudentController class to handle database operations.

· Develop a menu-driven interface in the View class for user interaction.

· Use prepared statements for secure and efficient database operations.

· Execute CRUD operations based on user input.

4.Implementation/Code:

```
import java.sql.*;
import java.util.*;
// Model Class
class Student {
    private int studentID;
    private String name;
    private String department;
    private double marks;

    public Student(int studentID, String name, String department, double marks) {
        this.studentID = studentID;
        this.name = name;
        this.department = department;
```

```
        this.marks = marks;
    }

    public int getStudentID() { return studentID; }
    public String getName() { return name; }
    public String getDepartment() { return department; }
    public double getMarks() { return marks; }
}

// Controller Class
class StudentController {
    private static final String URL = "jdbc:mysql://localhost:3306/your_database";
    private static final String USER = "your_username";
    private static final String PASSWORD = "your_password";

    public void addStudent(Student student) throws SQLException {
        String query = "INSERT INTO Student (StudentID, Name, Department, Marks)
VALUES (?, ?, ?, ?)";
        try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
             PreparedStatement pstmt = conn.prepareStatement(query)) {
            conn.setAutoCommit(false);
            pstmt.setInt(1, student.getStudentID());
            pstmt.setString(2, student.getName());
            pstmt.setString(3, student.getDepartment());
            pstmt.setDouble(4, student.getMarks());
            pstmt.executeUpdate();
            conn.commit();
            System.out.println("Student added successfully.");
```

```
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void viewStudents() throws SQLException {
        String query = "SELECT * FROM Student";
        try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(query)) {
            System.out.println("StudentID | Name | Department | Marks");
            while (rs.next()) {
                System.out.println(rs.getInt("StudentID") + " | " + rs.getString("Name") + " | " +
rs.getString("Department") + " | " + rs.getDouble("Marks"));
            }
        }
    }

    public void updateStudent(int studentID, String name, String department, double
marks) throws SQLException {
        String query = "UPDATE Student SET Name=?, Department=?, Marks=? WHERE
StudentID=?";
        try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
             PreparedStatement pstmt = conn.prepareStatement(query)) {
            conn.setAutoCommit(false);
            pstmt.setString(1, name);
            pstmt.setString(2, department);
            pstmt.setDouble(3, marks);
            pstmt.setInt(4, studentID);
```

```
            pstmt.executeUpdate();
            conn.commit();
            System.out.println("Student updated successfully.");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void deleteStudent(int studentID) throws SQLException {
        String query = "DELETE FROM Student WHERE StudentID=?";
        try (Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
             PreparedStatement pstmt = conn.prepareStatement(query)) {
            conn.setAutoCommit(false);
            pstmt.setInt(1, studentID);
            pstmt.executeUpdate();
            conn.commit();
            System.out.println("Student deleted successfully.");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

// View Class
public class StudentManagement {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        StudentController controller = new StudentController();
```

```
        boolean exit = false;
        while (!exit) {
            System.out.println("\n1. Add Student\n2. View Students\n3. Update Student\n4.
Delete Student\n5. Exit\nChoose an option: ");
            int choice = scanner.nextInt();
            scanner.nextLine();
            switch (choice) {
                case 1:
                    System.out.print("Enter Student ID: ");
                    int id = scanner.nextInt();
                    scanner.nextLine();
                    System.out.print("Enter Name: ");
                    String name = scanner.nextLine();
                    System.out.print("Enter Department: ");
                    String dept = scanner.nextLine();
                    System.out.print("Enter Marks: ");
                    double marks = scanner.nextDouble();
                    try {
                        controller.addStudent(new Student(id, name, dept, marks));
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                    break;
                case 2:
                    try {
                        controller.viewStudents();
                    } catch (SQLException e) {
                        e.printStackTrace();
```

```
            }
            break;
        case 3:
            System.out.print("Enter Student ID to update: ");
            int updateID = scanner.nextInt();
            scanner.nextLine();
            System.out.print("Enter new Name: ");
            String newName = scanner.nextLine();
            System.out.print("Enter new Department: ");
            String newDept = scanner.nextLine();
            System.out.print("Enter new Marks: ");
            double newMarks = scanner.nextDouble();
            try {
                controller.updateStudent(updateID, newName, newDept, newMarks);
            } catch (SQLException e) {
                e.printStackTrace();
            }
            break;
        case 4:
            System.out.print("Enter Student ID to delete: ");
            int deleteID = scanner.nextInt();
            try {
                controller.deleteStudent(deleteID);
            } catch (SQLException e) {
                e.printStackTrace();
            }
            break;
        case 5:
```

```
                exit = true;
                break;
            default:
                System.out.println("Invalid choice, try again.");
        }
    }
    scanner.close();
  }
}
```

6. Output:

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

7. Learning Outcomes:

· **JDBC Integration** – Understand how to establish database connections and execute SQL queries using JDBC.

· **MVC Architecture** – Learn to separate concerns using Model (Student class), View (menu-driven UI), and Controller (database operations).

· **CRUD Operations** – Gain hands-on experience in performing Create, Read, Update, and Delete operations on a database.

· **Transaction Handling** – Ensure data integrity using commit, rollback, and auto-commit mechanisms.

· **Security and Exception Handling** – Use **prepared statements** to prevent SQL injection and handle database errors effectively.

· **Scalability and Maintainability** – Develop structured and reusable code, making the application scalable and easy to maintain.