

Lab Sheet : Java Database Connectivity (JDBC)

1. Set Up MySQL Database

```
CREATE DATABASE employee_db;
```

```
USE employee_db;
```

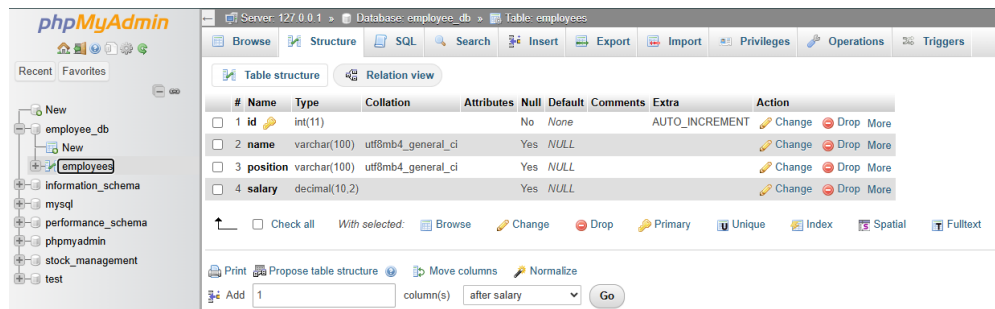
```
CREATE TABLE employees (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    name VARCHAR(100),  
    position VARCHAR(100),  
    salary DECIMAL(10, 2) );
```

-- Insert some sample data

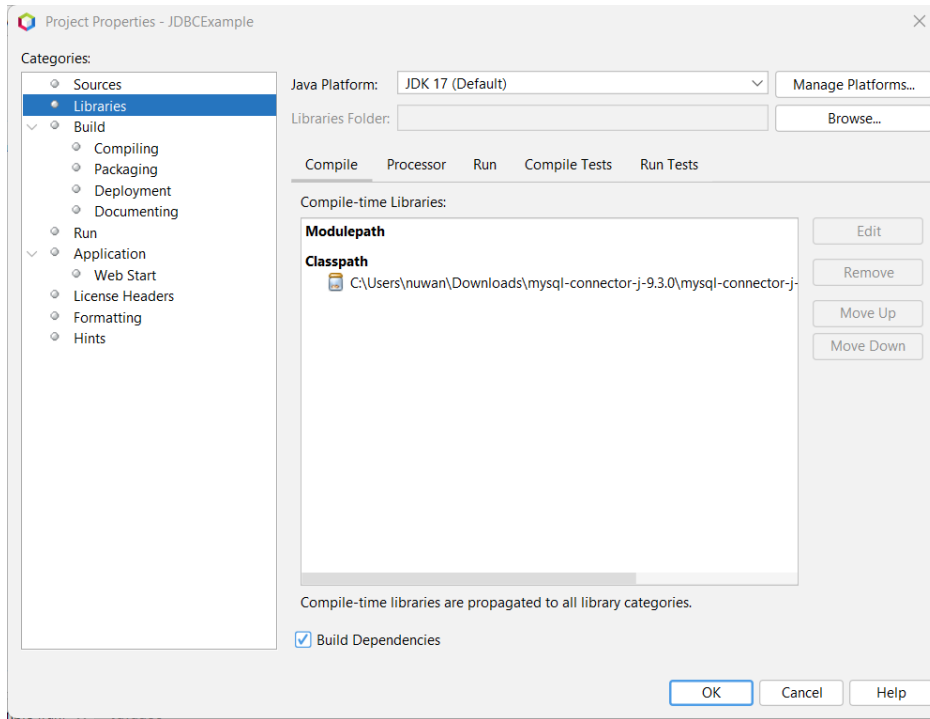
```
INSERT INTO employees (name, position, salary) VALUES ('John Doe', 'Software Engineer', 75000);
```

```
INSERT INTO employees (name, position, salary) VALUES ('Jane Smith', 'HR Manager', 65000);
```

```
INSERT INTO employees (name, position, salary) VALUES ('Steve Brown', 'Team Lead', 85000);
```



2. Set Up NetBeans Project



3. Establish JDBC Connection

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class DatabaseConnection {
    private static final String URL =
"jdbc:mysql://localhost:3306/employee_db"; // Database URL
    private static final String USER = "root"; // Your MySQL username
    private static final String PASSWORD = "password"; // Your MySQL password
    public static Connection getConnection() throws SQLException {
        try {
            // Load the JDBC driver
            Class.forName("com.mysql.cj.jdbc.Driver");
            // Return the database connection
            return DriverManager.getConnection(URL, USER, PASSWORD);
        } catch (ClassNotFoundException | SQLException e) {
            System.out.println("Connection failed: " + e.getMessage());
            throw new SQLException("Failed to establish connection."); } } }
```

```

4  */
5  package DatabaseConnection;
6
7  import java.sql.Connection;
8  import java.sql.DriverManager;
9  import java.sql.SQLException;
10
11 public class DatabaseConnection {
12
13     private static final String URL = "jdbc:mysql://localhost:3306/employee_db"; // Database URL
14     private static final String USER = "root"; // Your MySQL username
15     private static final String PASSWORD = "";
16
17     // Your MySQL password
18
19     public static Connection getConnection() throws SQLException {
20         try {
21             // Load the JDBC driver
22             Class.forName("com.mysql.cj.jdbc.Driver");
23
24             // Return the database connection
25             return DriverManager.getConnection(url: URL, user: USER, password: PASSWORD);
26
27         } catch (ClassNotFoundException | SQLException e) {
28             System.out.println("Connection failed: " + e.getMessage());
29             throw new SQLException(reason: "Failed to establish connection.");
30         }
31     }
32 }
33
Output - JDBCExample (run) x Javadoc
run:
BUILD SUCCESSFUL (total time: 0 seconds)
>>

```

4. Perform CRUD Operations

```

import DatabaseConnection.DatabaseConnection;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class EmployeeDAO {

    // Create an employee
    public static void addEmployee(String name, String position, double salary) {
        String sql = "INSERT INTO employees (name, position, salary) VALUES (?, ?, ?)";
        try (Connection conn = DatabaseConnection.getConnection();
            PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setString(1, name);
            stmt.setString(2, position);
            stmt.setDouble(3, salary);
            int rowsAffected = stmt.executeUpdate();
            System.out.println("Employee added successfully. Rows affected: " +
rowsAffected);

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

```

```

// Read all employees
public static List<Employee> getAllEmployees() {
    List<Employee> employees = new ArrayList<>();
    String sql = "SELECT * FROM employees";

    try (Connection conn = DatabaseConnection.getConnection();
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(sql)) {

        while (rs.next()) {
            Employee employee = new Employee(
                rs.getInt("id"),
                rs.getString("name"),
                rs.getString("position"),
                rs.getDouble("salary")
            );
            employees.add(employee);
        }

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

    return employees;
}

// Update an employee's information
public static void updateEmployee(int id, String name, String position, double salary) {
    String sql = "UPDATE employees SET name = ?, position = ?, salary = ? WHERE
id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
        PreparedStatement stmt = conn.prepareStatement(sql)) {

        stmt.setString(1, name);
        stmt.setString(2, position);
        stmt.setDouble(3, salary);
        stmt.setInt(4, id);
        int rowsAffected = stmt.executeUpdate();
        System.out.println("Employee updated successfully. Rows affected: " +
rowsAffected);

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

```

```

    }

    // Delete an employee
    public static void deleteEmployee(int id) {
        String sql = "DELETE FROM employees WHERE id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
            PreparedStatement stmt = conn.prepareStatement(sql)) {

            stmt.setInt(1, id);
            int rowsAffected = stmt.executeUpdate();
            System.out.println("Employee deleted successfully. Rows affected: " +
                rowsAffected);

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

```

The screenshot shows an IDE with several tabs open: ThreadPooExample.java, ThreadLifecycleExample.java, index.html, JDBCExample.java, DatabaseConnection.java, and EmployeeDAO.java. The main editor displays the code for the `deleteEmployee` method, which is identical to the code block above. The code includes comments and uses try-with-resources for the database connection and statement. The IDE's output window at the bottom shows a successful build message: "BUILD SUCCESSFUL (total time: 1 second)".

5. Create Employee.java Class

```

public class Employee {
    private int id;
    private String name;
    private String position;
    private double salary;
}

```

```

public Employee(int id, String name, String position, double salary) {
    this.id = id;
    this.name = name;
    this.position = position;
    this.salary = salary;
}

// Getters and setters
public int getId() { return id; }
public void setId(int id) { this.id = id; }
public String getName() { return name; }
public void setName(String name) { this.name = name; }
public String getPosition() { return position; }
public void setPosition(String position) { this.position = position; }
public double getSalary() { return salary; }
public void setSalary(double salary) { this.salary = salary; }
@Override
public String toString() {
    return "Employee{id=" + id + ", name='" + name + "', position='" +
position + "', salary=" + salary + '}';
}
}

```

```

Source History
9  */
10 public class Employee {
11
12     private int id;
13     private String name;
14     private String position;
15     private double salary;
16     public Employee(int id, String name, String position, double salary) {
17         this.id = id;
18         this.name = name;
19         this.position = position;
20         this.salary = salary;
21     }
22     // Getters and setters
23     public int getId() { return id; }
24     public void setId(int id) { this.id = id; }
25     public String getName() { return name; }
26     public void setName(String name) { this.name = name; }
27     public String getPosition() { return position; }
28     public void setPosition(String position) { this.position = position; }
29     public double getSalary() { return salary; }
30     public void setSalary(double salary) { this.salary = salary; }
31     @Override
32     public String toString() {
33         return "Employee{id=" + id + ", name='" + name + "', position='" +
34 position + "', salary=" + salary + '}';
35     }
36 }
37
38
Output - JDBCExample (run) x Javadoc
run:
BUILD SUCCESSFUL (total time: 0 seconds)
>>

```

6. Test the Application

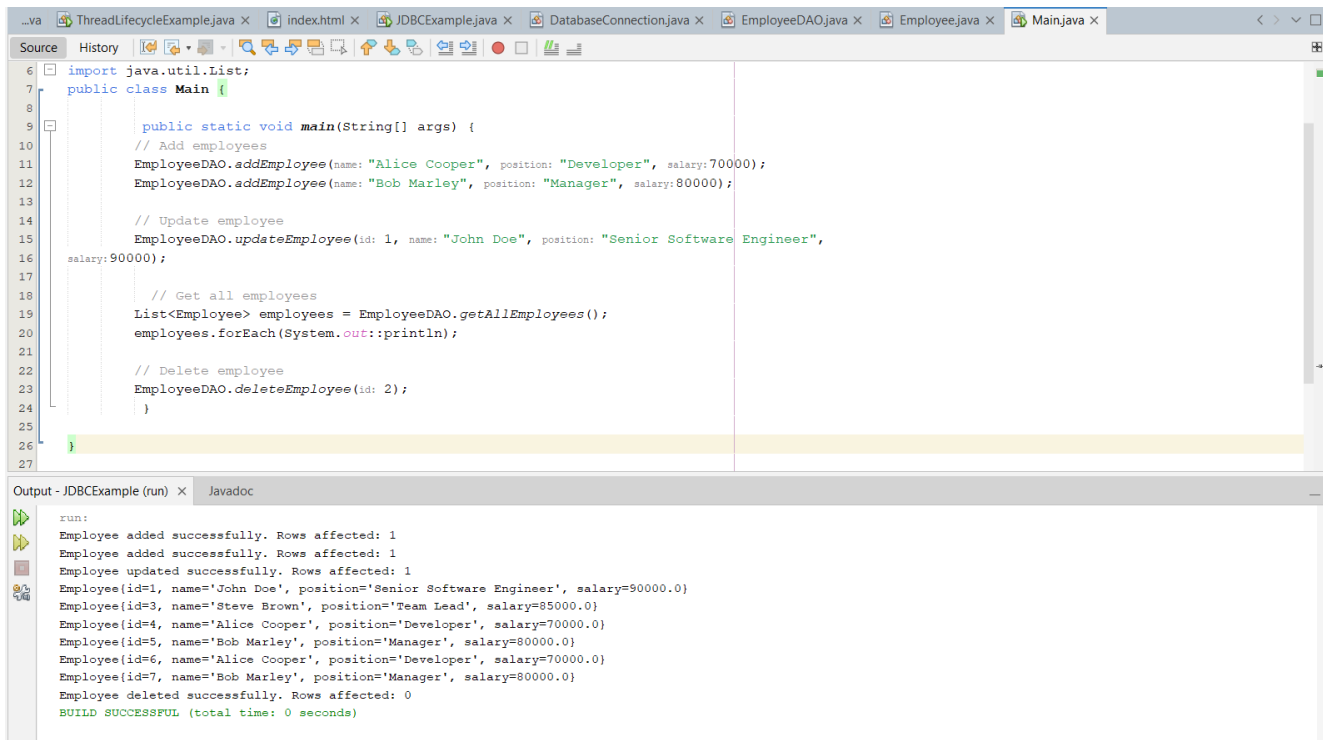
```
import java.util.List;
public class Main {

    public static void main(String[] args) {
        // Add employees
        EmployeeDAO.addEmployee("Alice Cooper", "Developer", 70000);
        EmployeeDAO.addEmployee("Bob Marley", "Manager", 80000);

        // Update employee
        EmployeeDAO.updateEmployee(1, "John Doe", "Senior Software Engineer",
90000);

        // Get all employees
        List<Employee> employees = EmployeeDAO.getAllEmployees();
        employees.forEach(System.out::println);

        // Delete employee
        EmployeeDAO.deleteEmployee(2);
    }
}
```



The screenshot shows an IDE with the following tabs: ThreadLifecycleExample.java, index.html, JDBCExample.java, DatabaseConnection.java, EmployeeDAO.java, Employee.java, and Main.java. The Main.java file is open, showing the code from the previous block. The output window at the bottom shows the following text:

```
run:
Employee added successfully. Rows affected: 1
Employee added successfully. Rows affected: 1
Employee updated successfully. Rows affected: 1
Employee(id=1, name='John Doe', position='Senior Software Engineer', salary=90000.0)
Employee(id=3, name='Steve Brown', position='Team Lead', salary=85000.0)
Employee(id=4, name='Alice Cooper', position='Developer', salary=70000.0)
Employee(id=5, name='Bob Marley', position='Manager', salary=80000.0)
Employee(id=6, name='Alice Cooper', position='Developer', salary=70000.0)
Employee(id=7, name='Bob Marley', position='Manager', salary=80000.0)
Employee deleted successfully. Rows affected: 0
BUILD SUCCESSFUL (total time: 0 seconds)
```

7. Run the Application

The screenshot shows a database management interface with a top navigation bar containing tabs: Browse, Structure, SQL, Search, Insert, Export, Import, Privileges, Operations, Tracking, and Triggers. The main content area displays a query result for the 'employees' table. A green status bar at the top indicates 'Showing rows 0 - 5 (6 total, Query took 0.0021 seconds.)'. Below this, the SQL query 'SELECT * FROM `employees`' is shown. A toolbar includes a 'Profiling' checkbox and links for 'Edit inline', 'Edit', 'Explain SQL', 'Create PHP code', and 'Refresh'. A control bar shows 'Show all', 'Number of rows: 25', 'Filter rows: Search this table', and 'Sort by key: None'. An 'Extra options' button is also present. The data is presented in a table with columns: id, name, position, and salary. Each row includes action icons for Edit, Copy, and Delete. Below the table, there are checkboxes for 'Check all' and 'With selected', along with 'Edit', 'Copy', 'Delete', and 'Export' buttons. A second control bar is identical to the one above. At the bottom, a 'Query results operations' section contains buttons for 'Print', 'Copy to clipboard', 'Export', 'Display chart', and 'Create view'. A 'Bookmark this SQL query' button is located at the very bottom.

Server: 127.0.0.1 » Database: employee_db » Table: employees

Showing rows 0 - 5 (6 total, Query took 0.0021 seconds.)

SELECT * FROM `employees`

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]

Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Extra options

| | id | name | position | salary |
|---|----|--------------|--------------------------|----------|
| <input type="checkbox"/> Edit Copy Delete | 1 | John Doe | Senior Software Engineer | 90000.00 |
| <input type="checkbox"/> Edit Copy Delete | 3 | Steve Brown | Team Lead | 85000.00 |
| <input type="checkbox"/> Edit Copy Delete | 4 | Alice Cooper | Developer | 70000.00 |
| <input type="checkbox"/> Edit Copy Delete | 5 | Bob Marley | Manager | 80000.00 |
| <input type="checkbox"/> Edit Copy Delete | 6 | Alice Cooper | Developer | 70000.00 |
| <input type="checkbox"/> Edit Copy Delete | 7 | Bob Marley | Manager | 80000.00 |

Check all | With selected: Edit Copy Delete Export

Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Query results operations

Print Copy to clipboard Export Display chart Create view

Bookmark this SQL query