**Lab sheet 02**

Java JDBC Lab Practical using NetBeans IDE 8.2

1. Set Up MySQL Database

CREATE TABLE employees (

id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(100),

position VARCHAR(100),

salary DECIMAL(10, 2)

);

INSERT INTO employees (name, position, salary) VALUES (&#39;John Doe&#39;, &#39;Software

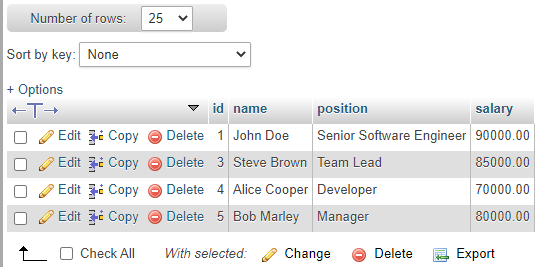
Engineer&#39;, 75000);

INSERT INTO employees (name, position, salary) VALUES (&#39;Jane Smith&#39;, &#39;HR

Manager&#39;, 65000);

INSERT INTO employees (name, position, salary) VALUES (&#39;Steve Brown&#39;, &#39;Team

Lead&#39;, 85000);



2.Create a DatabaseConnection.java

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 = ""; // 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.");

}

}

}

1. Create EmployeeDAO.java for CRUD Operations

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&#39;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();

}

}

1. Create a simple Employee.java POJO (Plain Old Java Object) to represent employee data.

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&#39;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();

}

}

1. Create a simple Employee.java POJO (Plain Old Java Object) to represent employee data.

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 + '}';

}

}

5.Create a Main.java class to test the CRUD operations.

import java.util.List;

public class JDBCExample {

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);

}

}

