**Asutosh Dash-Daily Assignment**

**Sprint 1 Day 4**

1. Basic JDBC Connection Setup

Question: Write a Java method that establishes a connection to a MySQL database using JDBC. The method should handle any exceptions and return the Connection object.

Solution:

package com.nisum;  
  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
import java.sql.Statement;  
  
public class Example {  
 public static void main(String[] args) throws SQLException {  
 String url="jdbc:mysql://localhost:3306/jdbc";  
 String username="root";  
 String password="root";  
 Connection connection = DriverManager.*getConnection*(url,username,password);  
 Statement statement=connection.createStatement();  
 statement.executeUpdate("INSERT INTO PRODUCT VALUES (1 ,'CHOCO',1000)");  
 System.*out*.println(connection);  
 }  
}

1. Executing Simple Queries

Question: Create a method that takes a student ID as a parameter and retrieves that student's information from a database table named 'students'. The method should return a Student object,.

Solution:

package com.nisum;  
  
import java.sql.\*;  
  
public class Example4 {  
  
 public static void main(String[] args) {  
 Student student = *getStudentById*(1); // Change ID as needed  
 if (student != null) {  
 System.*out*.println(student);  
 }  
 }  
  
 // Method to retrieve student by ID  
 public static Student getStudentById(int studentId) {  
 String url = "jdbc:mysql://localhost:3306/jdbc";  
 String username = "root";  
 String password = "root";  
  
 String query = "SELECT \* FROM students WHERE id = ?";  
  
 try (  
 Connection connection = DriverManager.*getConnection*(url, username, password);  
 PreparedStatement pstmt = connection.prepareStatement(query)  
 ) {  
 pstmt.setInt(1, studentId);  
 ResultSet rs = pstmt.executeQuery();  
  
 if (rs.next()) {  
 int id = rs.getInt("id");  
 String name = rs.getString("name");  
 String dob = rs.getString("dob");  
 String major = rs.getString("major");  
 String admissionDate = rs.getString("admission\_date");  
 double gpa = rs.getDouble("gpa");  
  
 return new Student(id, name, dob, major, admissionDate, gpa);  
 } else {  
 System.*out*.println("No student found with ID: " + studentId);  
 }  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
  
 return null;  
 }  
  
 // Inner Student class  
 static class Student {  
 private int id;  
 private String name;  
 private String dob;  
 private String major;  
 private String admissionDate;  
 private double gpa;  
  
 public Student(int id, String name, String dob, String major, String admissionDate, double gpa) {  
 this.id = id;  
 this.name = name;  
 this.dob = dob;  
 this.major = major;  
 this.admissionDate = admissionDate;  
 this.gpa = gpa;  
 }  
  
 @Override  
 public String toString() {  
 return "Student{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 ", dob='" + dob + '\'' +  
 ", major='" + major + '\'' +  
 ", admissionDate='" + admissionDate + '\'' +  
 ", gpa=" + gpa +  
 '}';  
 }  
 }  
}

1. Insert, Update, and Delete Operations

Question: Create methods for inserting a new student, updating an existing student's information, and deleting a student from the 'students' table. Each method should return a boolean indicating success or failure.

Explanation:

We will use executeUpdate() to update the data in the database and which will return number of rows updated then we will use executeQuery() to read the data from the database.

Solution:

package com.nisum;  
  
import java.sql.\*;  
import java.util.HashMap;  
  
public class Example1 {  
 public static void main(String[] args) throws SQLException {  
 String url = "jdbc:mysql://localhost:3306/jdbc";  
 String username = "root";  
 String password = "root";  
  
 try (  
 Connection connection = DriverManager.*getConnection*(url, username, password);  
 Statement statement = connection.createStatement()  
 ) {  
  
 int rowsUpdated = statement.executeUpdate("UPDATE product SET price=10 WHERE name='CHOCO'");  
 System.*out*.println("Rows updated: " + rowsUpdated);  
  
  
 ResultSet res = statement.executeQuery("SELECT \* FROM product");  
  
  
 while (res.next()) {  
 int id = res.getInt("id");  
 String name = res.getString("name");  
 int price = res.getInt("price");  
  
 System.*out*.println("ID: " + id + ", Name: " + name + ", Price: " + price);  
 }  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
}

1. Transaction Management

Question: Write a method that transfers credits from one student to another as part of a peer tutoring program. This should be done as a single transaction that either completely succeeds or fails.

Solution:

package com.nisum;  
  
import java.sql.\*;  
  
public class Example5 {  
 public static void main(String[] args) throws SQLException {  
 String url="jdbc:mysql://localhost:3306/jdbc";  
 String username="root";  
 String password="root";  
 Connection connection = DriverManager.*getConnection*(url,username,password);  
 Statement statement=connection.createStatement();  
 ResultSet res=statement.executeQuery("Select \* from accounts");  
 while(res.next()){  
 System.*out*.println(res.getInt("id")+" "+res.getString("name")+" "+res.getString("credits"));  
 }  
 connection.setAutoCommit(false);  
 try{  
 statement.executeUpdate("update accounts set balance = credits-100 where name='John'");  
 System.*out*.println("Transaction commited");  
 System.*out*.println("------------------");  
 if(true)throw new RuntimeException("error");  
 statement.executeUpdate("update accounts set balance = credicts+100 where name='Jane'");  
 connection.commit();  
 System.*out*.println("Transaction commited");  
  
 }catch (Exception e){  
 System.*out*.println("rolled back");  
 connection.rollback();  
 }  
  
 }  
}

1. Advanced Query with Joins

Question: Write a method that retrieves all courses taken by a given student, including the course name, instructor, and grade. This requires joining multiple tables: 'students', 'enrollments', and 'courses'.

-- Students table

CREATE TABLE students (

id INT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

date\_of\_birth DATE,

major VARCHAR(50),

admission\_date DATE,

gpa DECIMAL(3,2)

);

-- Courses table

CREATE TABLE courses (

id INT PRIMARY KEY,

course\_code VARCHAR(10) UNIQUE NOT NULL,

course\_name VARCHAR(100) NOT NULL,

department VARCHAR(50) NOT NULL,

credits INT NOT NULL,

instructor VARCHAR(100) NOT NULL,

max\_capacity INT

);

-- Enrollments table (junction table to handle many-to-many relationship)

CREATE TABLE enrollments (

student\_id INT,

course\_id INT,

semester VARCHAR(20) NOT NULL, -- e.g., "Fall 2024"

enrollment\_date DATE NOT NULL,

grade VARCHAR(2), -- e.g., "A", "B+", "C", etc.

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

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

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

);

Explanation:

We will create the tables and fill the coulmns mannually in sql workbench then we will use executeQuery(),to execute the SQL cmd to join the tables and give the required data from the database .

Solution:

package com.nisum;  
  
import java.sql.\*;  
  
public class Example3 {  
 public static void main(String[] args) throws SQLException {  
 String url="jdbc:mysql://localhost:3306/jdbc";  
 String username="root";  
 String password="root";  
 Connection connection = DriverManager.*getConnection*(url,username,password);  
 Statement statement=connection.createStatement();  
 ResultSet res=statement.executeQuery("Select c.coursesname AS Course\_name,c.insstructor,e.grade from students s join enrollments e on s.id=e.student\_id join courses c on e.course\_id=c.id");  
 while(res.next()){  
 String courseName = res.getString("Course\_name");  
 String instructor = res.getString("insstructor");  
 String grade=res.getString("grade");  
 System.*out*.println("Course\_Name: "+courseName+" Instructor: "+instructor+" Grade: "+grade);  
 }  
  
// System.out.println(connection);  
 }  
}

1. Dynamic Query Building with PreparedStatement

Question: Create a method that builds a dynamic search query for products based on optional filter criteria (category, minimum price, maximum price). Use PreparedStatement to handle the varying number of parameters.

Solution:

package com.nisum;  
  
import java.sql.\*;  
import java.util.ArrayList;  
import java.util.List;  
  
public class ProductSearch {  
  
 public static void main(String[] args) {  
 *searchProducts*("Electronics", 100.0, 500.0); // Example call  
 *searchProducts*(null, 200.0, null); // Only min price  
 *searchProducts*(null, null, null); // No filters  
 }  
  
 public static void searchProducts(String category, Double minPrice, Double maxPrice) {  
 String url = "jdbc:mysql://localhost:3306/jdbc";  
 String user = "root";  
 String pass = "root";  
  
 StringBuilder query = new StringBuilder("SELECT \* FROM products WHERE 1=1");  
 List<Object> params = new ArrayList<>();  
  
 if (category != null) {  
 query.append(" AND category = ?");  
 params.add(category);  
 }  
 if (minPrice != null) {  
 query.append(" AND price >= ?");  
 params.add(minPrice);  
 }  
 if (maxPrice != null) {  
 query.append(" AND price <= ?");  
 params.add(maxPrice);  
 }  
  
 try (  
 Connection conn = DriverManager.*getConnection*(url, user, pass);  
 PreparedStatement pstmt = conn.prepareStatement(query.toString())  
 ) {  
 for (int i = 0; i < params.size(); i++) {  
 pstmt.setObject(i + 1, params.get(i));  
 }  
  
 ResultSet rs = pstmt.executeQuery();  
 while (rs.next()) {  
 int id = rs.getInt("id");  
 String name = rs.getString("name");  
 String cat = rs.getString("category");  
 double price = rs.getDouble("price");  
 System.*out*.println(id + " | " + name + " | " + cat + " | $" + price);  
 }  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
}