Prac 3) Write a program to implement CRUD operation in JDBC

What is JDBC?

JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

- Making a connection to a database.
- Creating SQL or MySQL statements.
- Executing SQL or MySQL queries in the database.
- Viewing & Modifying the resulting records.

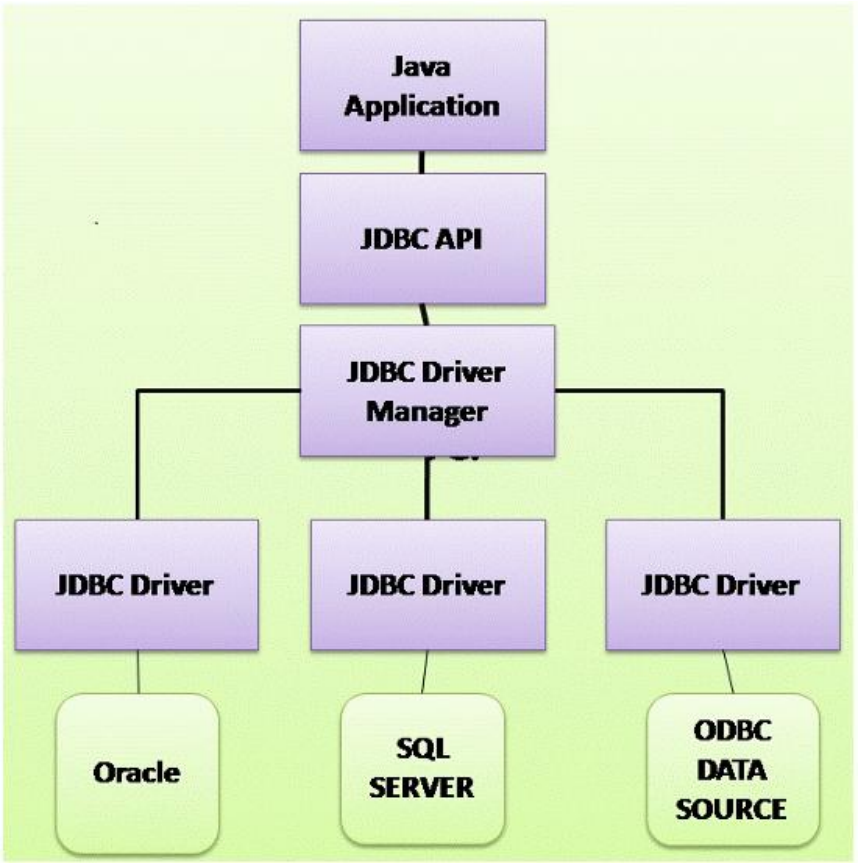
JDBC Architecture

The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers –

- JDBC API – This provides the application-to-JDBC Manager connection.
- JDBC Driver API – This supports the JDBC Manager-to-Driver Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases.

The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases.



Common JDBC Components

- **DriverManager** – This class manages a list of database drivers. Matches connection requests from the java application with the proper database driver using communication sub protocol. The first driver that recognizes a certain subprotocol under JDBC will be used to establish a database Connection.
- **Driver** – This interface handles the communications with the database server. You will interact directly with Driver objects very rarely. Instead, you use DriverManager objects, which manages objects of this type. It also abstracts the details associated with working with Driver objects.
- **Connection** – This interface with all methods for contacting a database. The connection object represents communication context, i.e., all communication with database is through connection object only.
- **Statement** – You use objects created from this interface to submit the SQL statements to the database. Some derived interfaces accept parameters in addition to executing stored procedures.
- **ResultSet** – These objects hold data retrieved from a database after you execute an SQL query using Statement objects. It acts as an iterator to allow you to move through its data.
- **SQLException** – This class handles any errors that occur in a database application.

There are following six steps involved in building a JDBC application:

Step 1: Import the packages: Needs that coder include the packages including the JDBC classes needed for database programming. Often, we will use "import java.sql.*;" as suffice.

Step 2: Register the JDBC driver: Requires that you initialise a driver so you can open a communication channel with the database.

The most common approach to register a driver is to use Java's **Class.forName()** method, to dynamically load the driver's class file into memory, which automatically registers it. This method is preferable because it allows you to make the driver registration configurable and portable.

Step 3: Open a connection: Requires using the DriverManager.getConnection( ) method to create a Connection object, which represents a physical connection with the database.

RDBMS	JDBC driver name	URL format
MySQL	com.mysql.jdbc.Driver	**jdbc:mysql://**hostname/databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	**jdbc:oracle:thin:@**hostname:port Number:databaseName
DB2	COM.ibm.db2.jdbc.net.DB2Driver	**jdbc:db2:**hostname:port Number/databaseName
Sybase	com.sybase.jdbc.SybDriver	**jdbc:sybase:Tds:**hostname: port Number/databaseName

Step 4: Execute a query: Requires using an object of type Statement for building and submitting an SQL statement to the database.

```
stmt = conn.createStatement( );
```

Once you've created a Statement object, you can then use it to execute an SQL statement with one of its three execute methods.

- **boolean execute (String SQL)**: Returns a boolean value of true if a ResultSet object can be retrieved; otherwise, it returns false. Use this method to execute SQL DDL statements or when you need to use truly dynamic SQL.
- **int executeUpdate (String SQL)** – Returns the number of rows affected by the execution of the SQL statement. Use this method to execute SQL statements for which you expect to get a number of rows affected - for example, an INSERT, UPDATE, or DELETE statement.
- **ResultSet executeQuery (String SQL)** – Returns a ResultSet object. Use this method when you expect to get a result set, as you would with a SELECT statement.

Step 5: Extract data from result set: Requires that you use the appropriateResultSet.get( ) method to retrieve the data from the result set.

A ResultSet object maintains a cursor that points to the current row in the result set. The term "result set" refers to the row and column data contained in a ResultSet object.

The methods of the ResultSet interface can be broken down into three categories −

- **Navigational methods** − Used to move the cursor around.
- **Get methods** − Used to view the data in the columns of the current row being pointed by the cursor.
- **Update methods** − Used to update the data in the columns of the current row. The updates can then be updated in the underlying database as well.

Step 6: Clean up the environment: Requires explicitly closing all database resources versus relying on the JVM's garbage collection.

```
conn.close();
```

(Write your own project structure here)

Code :

```
package abc;

import java.sql.*;

public class DEMO_JDBC {
    public static void main(String[] args) throws SQLException {
    Connection conn =
DriverManager.getConnection("jdbc:mysql://localhost:3306/studentdb","root","root");
    Statement stmt = conn.createStatement();

    String insertSQL = "INSERT INTO students (id, name, email) VALUES (1, 'Hemant',
'hem@example.com')";
    int insertedRows = stmt.executeUpdate(insertSQL);
    System.out.println("Insert operation affected " + insertedRows + " row(s).");

    //read
    String selectSQL = "SELECT * FROM students";
    ResultSet rs = stmt.executeQuery(selectSQL);
    System.out.println("\n--- Students Table ---");
    while (rs.next()) {
        rowCount++;
        System.out.println("ID: " + rs.getInt("id") +
                           ", Name: " + rs.getString("name") +
                           ", Email: " + rs.getString("email"));}

    // update
    String updateSQL = "UPDATE students SET name='Swami' WHERE id=1";
    int updatedRows = stmt.executeUpdate(updateSQL);
    System.out.println("Update operation affected " + updatedRows + " row(s).");

    //dele
    String deleteSQL = "DELETE FROM students WHERE id=1";
    int deletedRows = stmt.executeUpdate(deleteSQL);
    System.out.println("Delete operation affected " + deletedRows + " row(s).");

    }
}
```

Database :

CREATE DATABASE studentdb;

use studentdb;

CREATE TABLE students (id INT PRIMARY KEY, name VARCHAR(50), email VARCHAR(100));