**Assignment No:5**

**Title:** Write a program to implement MySQL/Oracle database connectivity with any front-end language to implement Database navigation operations (add, delete, edit etc.)

**Problem Statement:** Create a program to connect MySQL/Oracle with a front-end language, enabling database navigation operations such as adding, deleting, and editing records.

**Objective:** • To Understand the workings of two-tier and three-tier architecture.

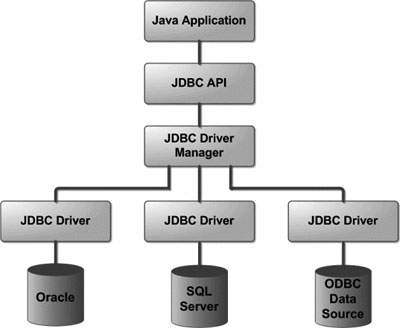
**Outcome:** Construct a two-tier and three-tier application using a suitable programming language and database.

**Tools Required:** Ubuntu OS, Eclipse, MySql, MySql\_Connector, Java.

**Theory:**

1. **What is JDBC?**

JDBC stands for **J**ava **D**ata**b**ase **C**onnectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.



## 2. Common JDBC Components

The JDBC API provides the following interfaces and classes −

**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 sub protocol 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.

## 3. What is JDBC Driver?

JDBC drivers implement the defined interfaces in the JDBC API, for interacting with your database server.

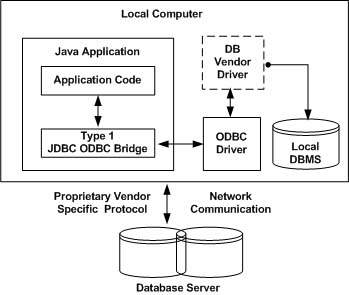
For example, using JDBC drivers enable you to open database connections and to interact with it by sending SQL or database commands then receiving results with Java.

The *Java.sql* package that ships with JDK, contains various classes with their behaviors defined and their actual implementations are done in third-party drivers. Third party vendors implements the *java.sql.Driver* interface in their database driver.

## Type 1: JDBC-ODBC Bridge Driver

In a Type 1 driver, a JDBC bridge is used to access ODBC drivers installed on each client machine. Using ODBC, requires configuring on your system a Data Source Name (DSN) that represents the target database.

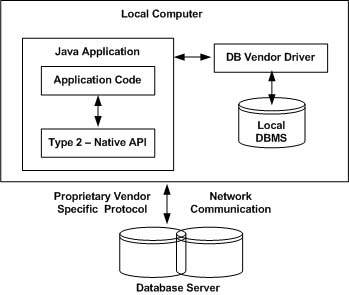
When Java first came out, this was a useful driver because most databases only supported ODBC access but now this type of driver is recommended only for experimental use or when no other alternative is available.



## Type 2: JDBC-Native API

In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database. These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor-specific driver must be installed on each client machine.

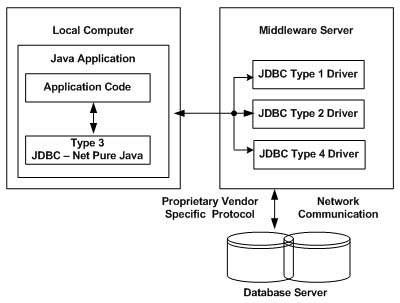
If we change the Database, we have to change the native API, as it is specific to a database and they are mostly obsolete now, but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead.



**Type 3: JDBC-Net pure Java**

In a Type 3 driver, a three-tier approach is used to access databases. The JDBC clients use standard network sockets to communicate with a middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.

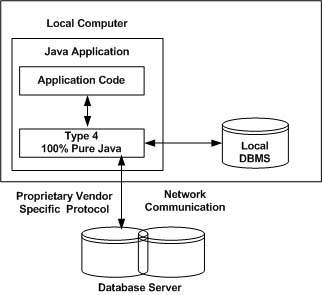
This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.



## Type 4: 100% Pure Java

In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.

kind of driver is extremely flexible, you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.



**4. Connecting Java Application to MySql database:**

We are using MySql as the database. So we need to know following informations for the mysql database:

1. **Driver class:**The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:**The connection URL for the mysql database is **jdbc:mysql://localhost:3306/sonoo** where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, we need to replace the sonoo with our database name.
3. **Username:**The default username for the mysql database is **root**.
4. **Password:**It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

**4.1 Steps:**

* ***Install MySQL and Set Up a Database:*** Ensure MySQL is installed and running. Create a database and table
* ***Add MySQL Connector JAR:*** Download **MySQL Connector/J** from [MySQL website](https://dev.mysql.com/downloads/connector/j/). Add the JAR file to your project’s classpath.
* ***Loads MySQL JDBC Driver:***

Class.forName("com.mysql.cj.jdbc.Driver");.

* ***Establishes a connection:***

DriverManager.getConnection(url, user, password);

* ***Statement:*** A Statement is used to execute SQL queries in Java. It is created from a Connection object and used to send SQL commands to the database.

Statement stmt = conn.createStatement();

* ***ResultSet:*** A ResultSet is used to store the data retrieved from the database when executing a SELECT query. It is returned by executing a query using Statement;

ResultSet rs = stmt.executeQuery("SELECT \* FROM users");

* ***Close connection:***

conn.close();

**4.2 Sample Code:**

|  |
| --- |
| import java.sql.\*;  public class JDBCExample  {  ***// JDBC driver name and database URL***  String JDBC\_DRIVER = "com.mysql.cj.jdbc.Driver";  String DB\_URL = "jdbc:mysql://localhost/STUDENTS";  ***// Database credentials***  String USER = "username";  String PASS = "password";    public static void main(String[] args)  {  Connection conn = null;  Statement stmt = null;  Try  {  ***//STEP 2: Register JDBC driver***  Class.forName("com.mysql.jdbc.Driver");  ***//STEP 3: Open a connection***  System.out.println("Connecting to database...");  conn = DriverManager.getConnection(DB\_URL, USER, PASS);  System.out.println("Connected successfully...");    ***//STEP 4: Execute a query***  System.out.println("Inserting records into the table...");  stmt = conn.createStatement();  String sql = "INSERT INTO Registration " +  "VALUES (100, 'Zara', 'Ali', 18)";  stmt.executeUpdate(sql);  sql = "INSERT INTO Registration " +  "VALUES (101, 'Mahnaz', 'Fatma', 25)";  stmt.executeUpdate(sql);  sql = "INSERT INTO Registration " +  "VALUES (102, 'Zaid', 'Khan', 30)";  stmt.executeUpdate(sql);  sql = "INSERT INTO Registration " +  "VALUES(103, 'Sumit', 'Mittal', 28)";  stmt.executeUpdate(sql);  System.out.println("Inserted records into the table...");    String sql = "SELECT id, first, last, age FROM Registration";  ResultSet rs = stmt.executeQuery(sql);  ***//STEP 5: Extract data from result set***  while(rs.next())  {  ***//Retrieve by column name***  int id = rs.getInt("id");  int age = rs.getInt("age");  String first = rs.getString("first");  String last = rs.getString("last");  ***//Display values***  System.out.print("ID: " + id);  System.out.print(", Age: " + age);  System.out.print(", First: " + first);  System.out.println(", Last: " + last);  }  }  catch(Exception e)  {  ***//Handle errors for Class.forName***  e.printStackTrace();  }  finally  {  ***//finally block used to close resources***  conn.close();  }  }  }  } |

**Conclusion:**

We have successfully implemented 2-tier application using java and Mysql.