* Introduction to JDBC :-

→ In today's scenario, many enterprise level applications need to intract with databases ~~for~~ storing information.

→ For this purpose, We used an API (Application program-ming Interface) i.e ODBC (open Database connectivity).

→ The ODBC API was the database API to connect and execute query with the database. But, ODBC API uses ODBC driver which is writtem in C language (i.e platform dependent and unsecured).

→ That is why java has defined its own API, called JDBC (Java Database connectivity), that uses JDBC drivers (Wri-tten in Java language).

→ The JDBC drivers are more compatible with Java Applns to provide database communication.

→ JDBC is a Java API to connect and execute query with the database. JDBC API uses jdbc drivers to connect with the database.

→ JDBC supports a wide level of portability and JDBC is simple and easy to use.

→ In JDBC API, a programmer needs a specific driver to connect to specific database.

RDBMS	Driver
Oracle	oracle.jdbc.driver.OracleDriver
MySQL	com.mysql.jdbc.Driver
SyBase	com.sybase.jdbc.SybDriver
SQLServer	com.microsoft.jdbc.SqlServer
DB2	com.ibm.db2.jdbc.net.DB2Driver

* List of some popular Drivers.*

* JDBC Architecture :-

The main function of the JDBC is to provide a standard abstraction for java applications to communication with databases.

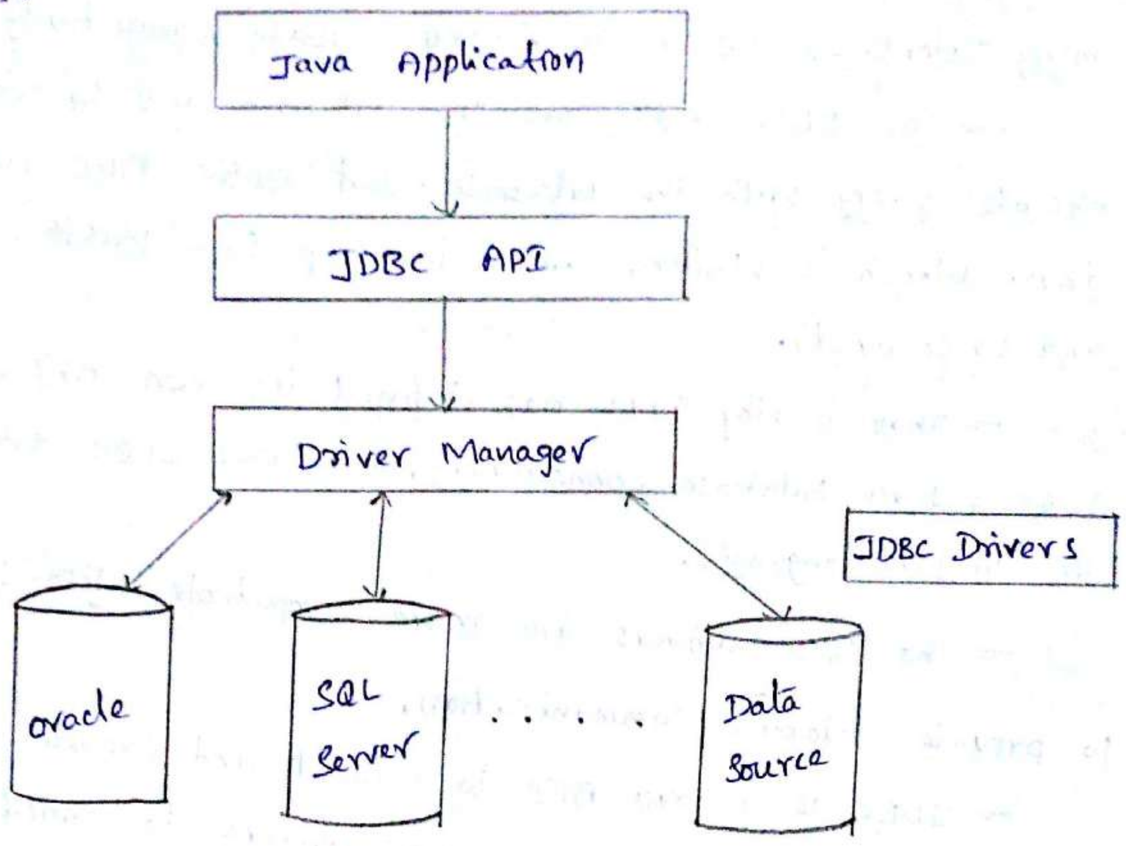


Fig:- The JDBC Architecture

As shown in figure, The Java application that wants to communicate with a database has to be programmed using JDBC API.

The JDBC Driver is required to process the SQL requests and generate the results.

The JDBC driver has to be plays an important role in the JDBC architecture. The Driver Manager uses some specific drivers to effectively connect with specific databases.

→ JDBC Driver is a software component that enables java application to intract with the database.

There are 4 types of JDBC drivers, those are

→ Type -1 Driver (JDBC - ODBC bridge driver)
→ Type - 2 Driver (partial JDBC driver)
→ Type - 3 Driver (pure java driver for middleware)
→ Type - 4 Driver (pure java driver with direct database connection)

* Type-1 Driver (JDBC - ODBC bridge driver) :-

The type -1 driver acts as a bridge between JDBC and other database connectivity mechanisms such as ODBC.

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC- ODBC bridge driver converts JDBC method calls into the ODBC method calls.

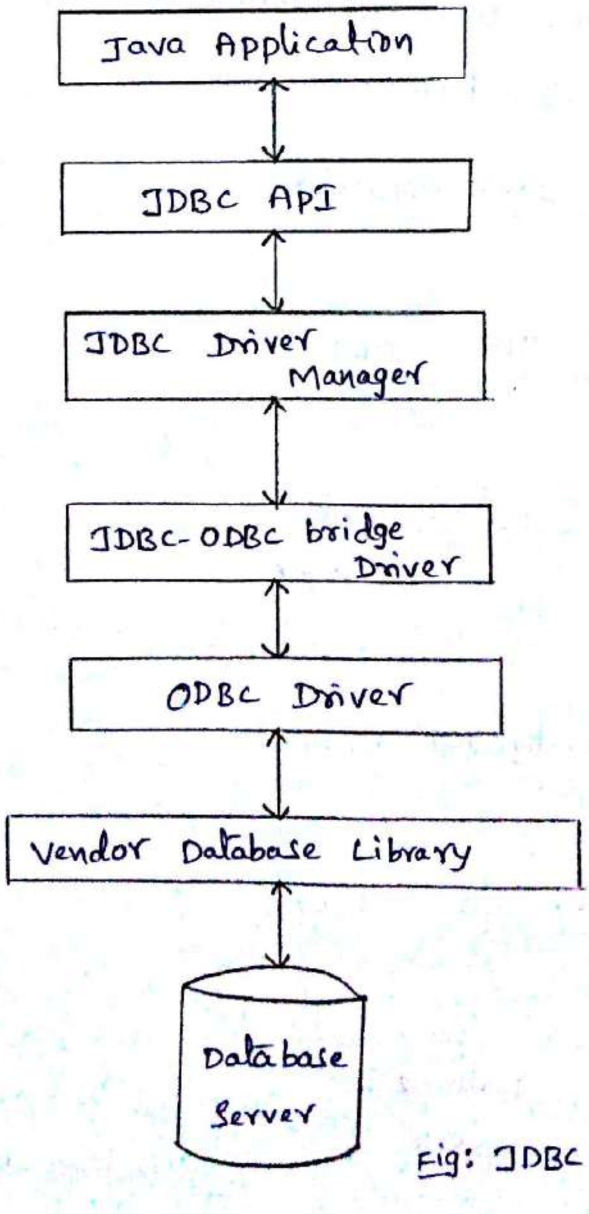


Fig: JDBC - ODBC Bridge Driver

Advantages:

* Easy to use.
* Can be easily connected to any database.

Disadvantages:

* performance degraded because large number of trans-lations (i.e JDBC calls to ODBC calls).
* The ODBC driver needs to be installed on the client machine.

* Type-2 Driver (partial JDBC driver) :-

The type-2 driver uses the client-side libraries of the database. So This driver is also called as Native-API driver.

This driver converts JDBC method calls into native calls of the database API. It is not written entirely in java, so it is called as partial JDBC driver.

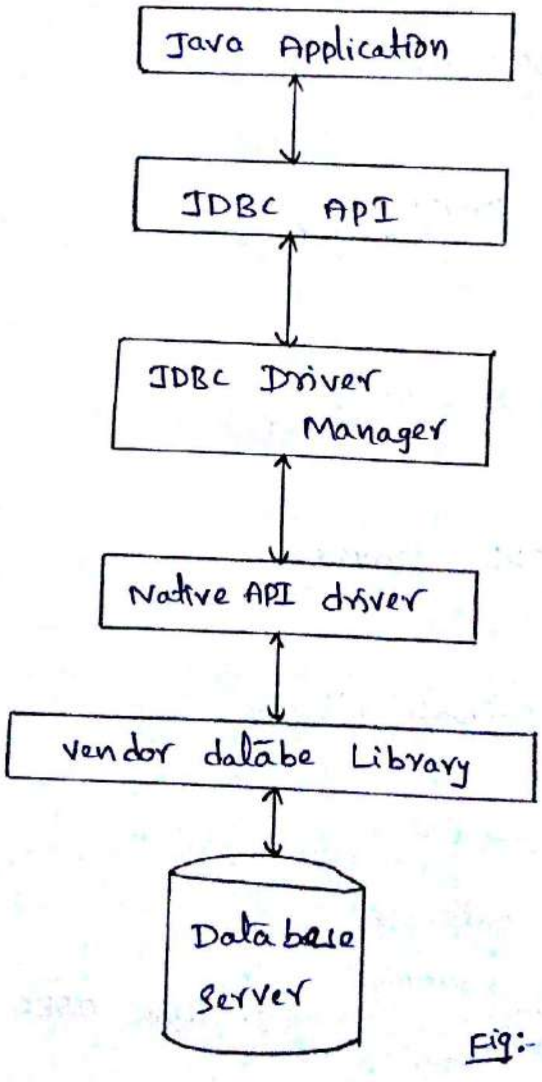


Fig:- Native API driver

Advantages:

* performance upgraded than JDBC-ODBC bridge driver.
* suitable to use with server-side applications.

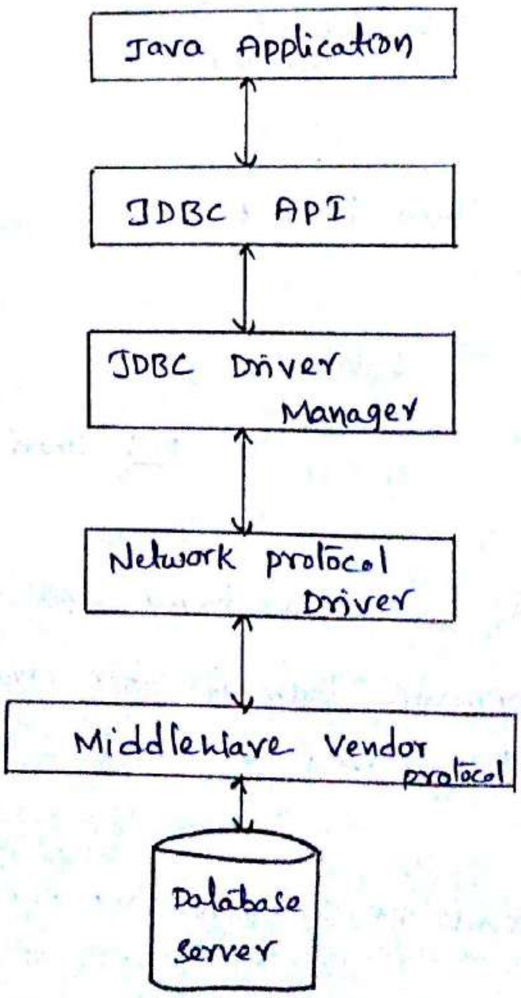
Disadvantages:

* This Native driver needs to installed on the each client Machine
* The vendor client library needs to be installed on client Machine
* It may increase the cost of the application if the application needs to run on different platforms.

* Type-3 Driver (pure Java driver for middleware):-

The type-3 driver is completely implemented in Java, hence it is a pure Java JDBC driver.

The type-3 driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor -specific database protocol. so it is called as Network protocol driver.



Advantages:

* No client side library is required on client side.
* pure Java drivers and auto downloadable.

Disadvantages:

* Network support is required on client machine.
* This driver is costly compared to other drivers.

* Type -4 Driver (pure java driver with direct database connection):-

The type-4 driver is a pure java driver, which converts JDBC calls directly into the vendor-specific database protocol. that is why it is known as thin driver.

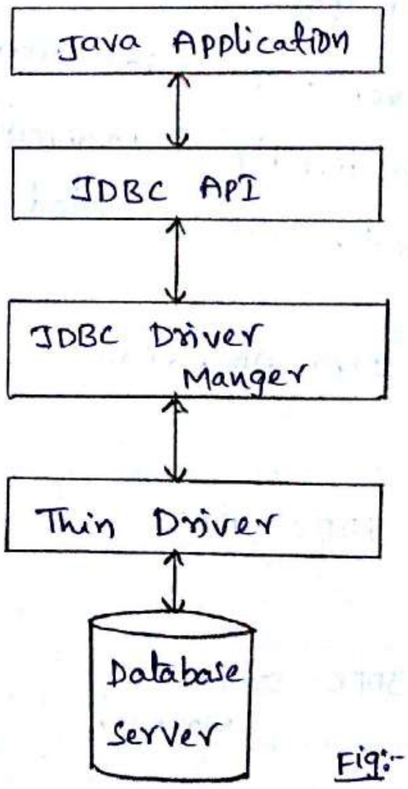


Fig:- Thin Driver.

Advantages:

* This driver is pure Java driver and auto downloadable.
* Better performance than all other drivers
* No software is required at client side or server side.

Disadvantages:

* Drivers depends on the Database.

* Database programming using JDBC :-

JDBC APIs are used by a Java application to communi-cate with a database.

In otherwords, we use JDBC connectivity code in Java application to communicate with a database.

There are 5 steps to connect any Java application with the database in java using JDBC. They are as follows:

<u>step 1</u> : Register the driver class

<u>step 2</u> : Creating connection

<u>step 3</u> : Creating statement

<u>step 4</u> : Executing SQL statements

<u>step 5</u> : closing connection.

* step 1 :- (Register the driver class)

In this step, we register the driver class with driver Manager by using forName() method of Class class.

<u>Syntax</u>: Class.forName(Driver class Name)

<u>Example</u>: Class.forName("oracle.jdbc.driver.OracleDriver");

* step 2 :- (Creating connection)

In this step, we can create a connection with database server by using getConnection() method of DriverManager class.

<u>Syntax</u>: getConnection(string url, string name, string pwd)

<u>Example</u>:

```
Connection con = DriverManager.getConnection(
                    "jdbc:oracle:thin:@localhost:1521:xe",
                              "system", "admin");
```

* step 3 :- (Creating statement)

After the connection made, we need to create the statement object to execute the SQL statements.

The createStatement() method of Connection interface is used to create statement. This statement object is responsible to execute SQL statements with the database.

```
Syntax:  createStatement()
Example:
        statement stmt = con.statement();
```

* **step 4:-** (Executing SQL statements)

After the statement object is created, it can be used to execute the SQL statements by using executeUpdate() (or) executeQuery() method of statement interface.

The executeQuery() method is only used to execute SELECT statements.

The executeUpdate() method is used to execute all SQL statements except SELECT statements.

```
Syntax:  executeQuery(String query)
         executeUpdate(String query)

Example:  // using executeQuery()
          String query = " Select * from emp";
          Resultset rs = stmt. executeQuery(query);
          // using executeUpdate()
          String query = "insert into emp values(504,'Madhu',29)";
           stmt. executeUpdate(query);
```

* **step 5:-** (closing the connection)

After executing all the SQL statements and obtaing the results, we need to close the connection and release the session.

The close() method of Connection interface is used to close the connection.

```
Syntax:-   close()
Example :-  con. close();
```

Scanned by CamScanner

*** Example :-** (connectivity with oracle database)

For connecting java application with the oracle database, we need to know following information to perform database connectivity with oracle.

In This example We are using oracle 10g as the database, so We need to know following information for the oracle database.

* **Driver class :** The driver class for oracle database is "oracle.jdbc.driver.OracleDriver".

* **Connection URL :** The connection URL for the oracle 10G database is "jdbc:oracle:thin:@localhost:1521:xe".

Where jdbc is the API, oracle is the database, thin is the driver, localhost is the servername on which oracle is running, 1521 is the port number and XE is the oracle service name.

* **username :** The default username for the oracle database is "system".

* **password :** password is given by the user at the time of installing the oracle database.

→ To connect java application with the oracle database ojdbc14.jar file is required to be loaded.

→ There are two ways to load the ojdbc14.jar file, We need to follow any one of two ways.

1. paste The ojdbc14.jar file in "java/jre/lib/ext" folder
2. Set classpath

Firstly, search the ojdbc14.jar file then go to "java/jre/lib/ext" folder and paste the jar file here.

(or)

Set class path : To set classpath, goto environment variable then click on new tab. In variable name write classpath and in variable value paste the path to ojdbc14.jar by appending ojdbc14.jar;.; as

"c:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar;.;".

* **Example:**

Let's first create a table and insert two or more records in oracle database.

```
SQL> create table emp(id number(10), name varchar2(40),
                                       age number(3));

SQL> insert into emp values(501, 'Madhu', 30);
SQL> insert into emp values(502, 'Hari', 32);
SQL> insert into emp values(503, 'Satti', 33);
```

* **program:** connect java application with oracle database for selecting or retriving data.

SelectData.java

```
import java.sql.*;
import java.util.*;
class SelectData
{
public static void main(String args[])
{
 Try
 {
 // Step 1: load the driver class
 Class.forName("oracle.jdbc.driver.OracleDriver");
 // Step 2: create the connection object
 Connection con = DriverManager.getConnection(
     "jdbc:oracle:thin:@localhost:1521:xe", "system", "admin");
 // step 3: create the statement object
  Statement stmt = con.createStatement();
 // step 4: execute query
  ResultSet rs = stmt.executeQuery("select * from emp");
  while (rs.next())
  {
  System.out.println(rs.getInt(1) + "    " + rs.getString(2) + "   "
                          + rs.getString(3));
  }
 // step 5: close the connection object
  con.close();
 }
 catch(Exception e) { System.out.println(e);}
}
}
```

Scanned by CamScanner

output:

```
D:\> javac SelectData.java
D:\> java SelectData
 501      Madhu      30
 502      Hari       32
 503      Satti      35
```

* program: connect Java application with oracle database for inserting data.

InsertData.java

```
import java.sql.*;
import java.util.*;
class InsertData
{
public static void main(String args[])
{
 try
 {
 Class.forName("oracle.jdbc.driver.oracleDriver");
 Connection con = DriverManager.getConnection("jdbc:oracle:thin
                    :@localhost:1521:xe","system","admin");

 Statement stmt = con.createStatement();
 stmt.executeUpdate("insert into emp values(504,'Ganesh',28)");
 System.out.println("Inserted...");
 con.close();
 }
 catch(Exception e)
 {
  System.out.println(e);
 }
}
}
```

output:

```
D:\> javac InsertData.java
D:\> java InsertData
Inserted...
```

* <u>program</u>: Java application with oracle database for update data.

Updatedata.java

```
import java.sql.*;
import java.util.*;
class Updatedata
{
public static void main(String args[])
{
 try
 {
  Class.forName("oracle.jdbc.driver.OracleDriver");
  Connection con = DriverManager.getConnection("jdbc:oracle:
               thin:@localhost:1521:xe", "system", "admin");
  Statement stmt = con.createStatement();
  stmt.executeUpdate("update emp set age=38 where id=503");
  System.out.println("updated....");
  con.close();
 }
 catch(Exception e)
 {
   System.out.printh("Exception is:" +e);
 }
 }
}
```

<u>output</u>:

```
D:/> javac Updatedata.java
D:/> java Updatedata
Updated....
```

* <u>DriverManager</u> <u>class</u> :-

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are avaliable and handles establishing a connection between a database and the appropri-ate driver.