**Introduction to JDBC**

JDBC or Java Database Connectivity is a Java API to connect and execute the query with the database. It is a specification from Sun microsystems that provides a standard abstraction (API or Protocol) for java applications to communicate with various databases. It provides the language with java database connectivity standards. It is used to write programs required to access databases. JDBC, along with the database driver, can access databases and spreadsheets.

JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

- o   JDBC-ODBC Bridge Driver,
- o   Native Driver,
- o   Network Protocol Driver, and
- o   Thin Driver

We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database. It is like Open Database Connectivity (ODBC) provided by Microsoft.
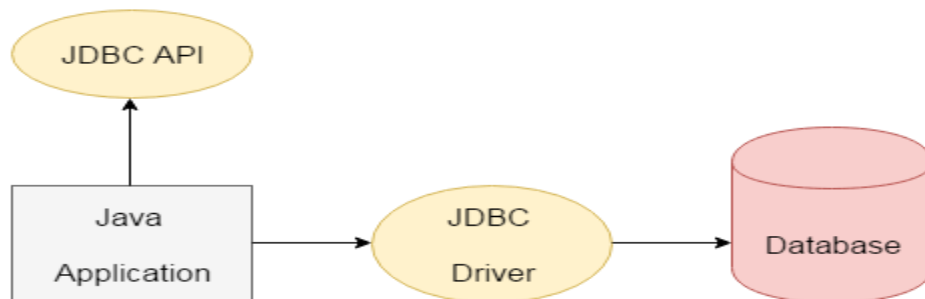
The **java.sql** package contains classes and interfaces for JDBC API. A list of popular *interfaces* of JDBC API are given below:

- o   Driver interface
- o   Connection interface
- o   Statement interface
- o   PreparedStatement interface
- o   CallableStatement interface
- o   ResultSet interface
- o   ResultSetMetaData interface
- o   DatabaseMetaData interface

o   RowSet interface

A list of popular *classes* of JDBC API are given below:
  o   DriverManager class
  o   Blob class
  o   Clob class
  o   Types class

Before JDBC, ODBC API was the database API to connect and execute the query with the database. But, DBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language). We can use JDBC API to handle database using Java program and can perform the following activities:

1. Connect to the database
2. Execute queries and update statements to the database
3. Retrieve the result received from the database.

JDBC Drivers

JDBC-ODBC bridge driver

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 function calls. This is now discouraged because of thin driver.
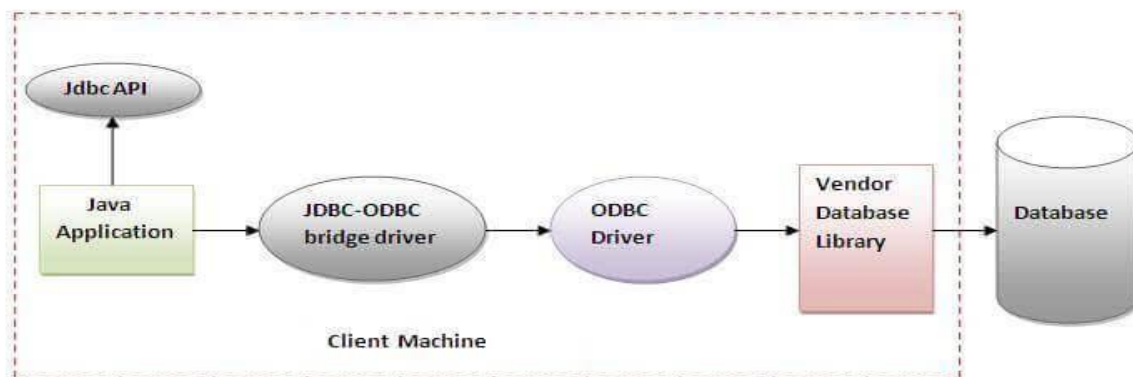

Figure- JDBC-ODBC Bridge Driver

Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java. It requires database-specific coding to be done in the middle tier.
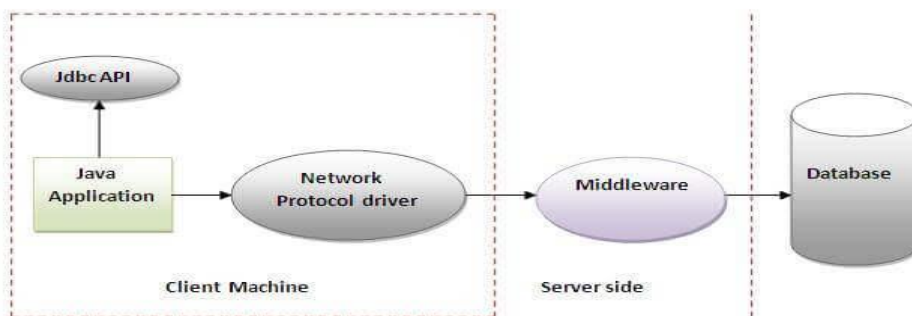

Figure- Network Protocol Driver

Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language. In this No software is required at client side or server side.
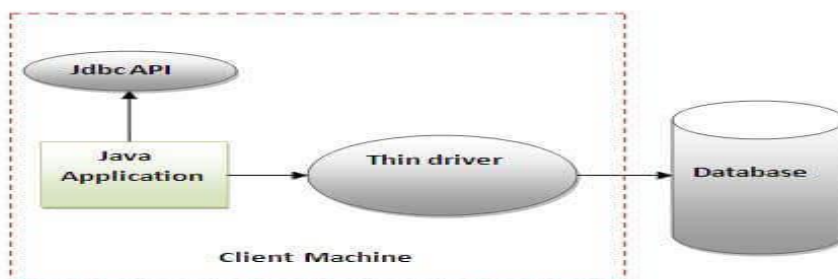

Figure- Thin Driver

Java Database Connectivity with 5 Steps

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

- o   Register the Driver class
- o   Create connection
- o   Create statement
- o   Execute queries
- o   Close connection

1) Register the driver class

The **forName()** method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of forName() method

1.   **public static void** forName(String className) **throws** ClassNotFoundException

Example to register the OracleDriver class

Here, Java program is loading oracle driver to establish database connection.

1.   Class.forName("oracle.jdbc.driver.OracleDriver");

2) Create the connection object

The **getConnection()** method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

1.   1) **public static** Connection getConnection(String url)**throws** SQLException
2.   2) **public static** Connection getConnection(String url,String name,String password) **throws** SQLException

Example to establish connection with the Oracle database

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

3) Create the Statement object

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

Syntax of createStatement() method

1. **public** Statement createStatement()**throws** SQLException

Example to create the statement object

1. Statement stmt=con.createStatement();

---

4) Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery**() method**

1. **public** ResultSet executeQuery(String sql)**throws** SQLException

Example to execute query

1. ResultSet rs=stmt.executeQuery("select * from emp");
2.
3. **while**(rs.next()){
4. System.out.println(rs.getInt(1)+" "+rs.getString(2));
5. }

---

5) Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method

1. **public void** close()**throws** SQLException

Example to close connection

1. con.close();

It avoids explicit connection closing step.

Java Database Connectivity with Oracle

To connect java application with the oracle database, we need to follow 5 following steps. In this example, we are using Oracle 10g as the database. So we need to know following information for the oracle database:

1. **Driver class:** The driver class for the oracle database is **oracle.jdbc.driver.OracleDriver**.
2. **Connection URL:** The connection URL for the oracle10G database is

jdbc:oracle:thin:@localhost:1521:xe where jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and XE is the Oracle service name. You may get all these information from the tnsnames.ora file.

3. **Username:** The default username for the oracle database is **system**.
4. **Password:** It is the password given by the user at the time of installing the oracle database.

Example to Connect Java Application with Oracle database

In this example, we are connecting to an Oracle database and getting data from **emp** table. Here, **system** and **oracle** are the username and password of the Oracle database.

```java
import java.sql.*;
class OracleCon{
public static void main(String args[]){
try{
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");
//step2 create  the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");

//step3 create the statement object
Statement stmt=con.createStatement();
//step4 execute query
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));

//step5 close the connection object
con.close();

}catch(Exception e){ System.out.println(e);}

}
}
```

// DML Commands

```
import java.sql.*;
class FetchRecord{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
Statement stmt=con.createStatement();
 //stmt.executeUpdate("insert into emp765 values(33,'Irfan',50000)");
//int result=stmt.executeUpdate("update emp765 set name='Vimal',salary=10000 where id=33");
int result=stmt.executeUpdate("delete from emp765 where id=33");
System.out.println(result+" records affected");
con.close();
}}
```