DATABASE ACCESS

JDBC

by

Tek Raj Chhetri

Introduction: to JDBC:

JDBC™ is a specification from Sunmicrosystems that provides a standard abstraction for java applications to communicate with different database. It is used to write the program required to access the database. JDBC, along with drivers, is capable of accessing database and spread-sheets.
It is an platform-independent interface between a relational database and java programming language which allows java programs to execute SQL statements, retrive results etc.

Characteristics of JDBC

i) Supports wide level of portability.
ii) Provides java interface that are compatible with java applications. These providers are also responsible for providing driver services.
iii) Provides higher level API for application programmers.
iv) Provides JDBC API for java applications.

Components of JDBC:

JDBC has four main components with which it communicate with database. They are:-

(i) The JDBC API:
It provides the various methods
[illegible] effective communic-

to connect to a clint ~~database~~ application. It provides the two main packages, java.sql, and javax.sql, to interact with database.

These packages are available in both J2SE and J2EE, platform which conforms write once run any where any time.

(ii) The JDBC Driver Manager: It loads the database specific drivers in an application to establish a connection with the database. It is also used to select the most appropriate database specific driver from the previously loaded driver when new connection to the database is established.

It is also used to make the database-specific calls to the database to process the user requests.

(iii) The JDBC test Suite: It evaluates the JDBC driver for its compatibility with Java EE. The JDBC test suite is used to test operations being performed by JDBC drivers.

(iv) The JDBC-ODBC bridge: It connects the database driver to the database. The bridge translates the JDBC method calls to the ODBC function calls, and is used to implement JDBC for any database for which an ODBC is available.

It can be availed by importing sun.jdbc.odbc package.

Working of JDBC:

The working of the JDBC can be explained as:-

1) First of all Java application establishes connection with data Source.
2) Then Java application invokes classes and interface from JDBC driver for sending queries to data Source.
3) The JDBC driver connects to corresponding database and retrives the result.
4) These results are based on SQL statements which are then returned to Java application.
5) Java application then uses the retrived information for further processing.

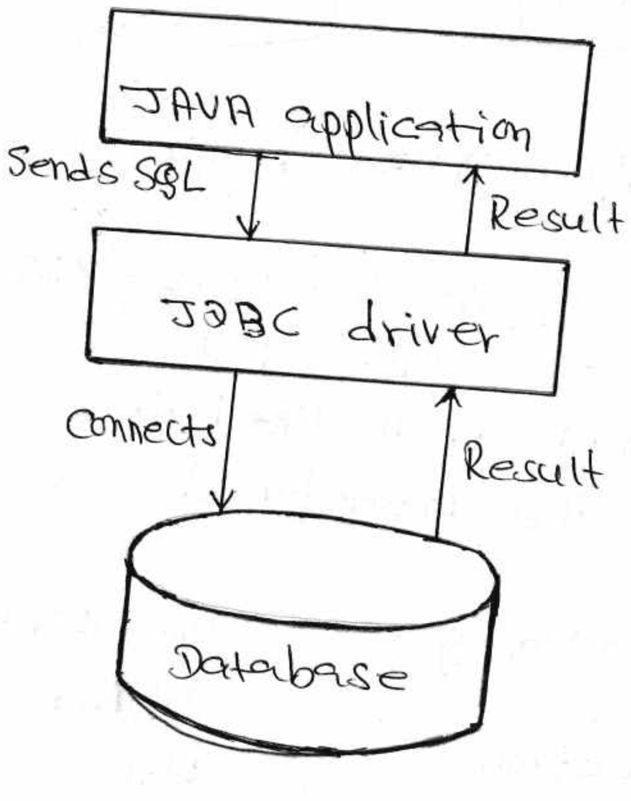


fig: Role of JDBC.

JDBC Architecture:

The figure below shows the architecture of JDBC.

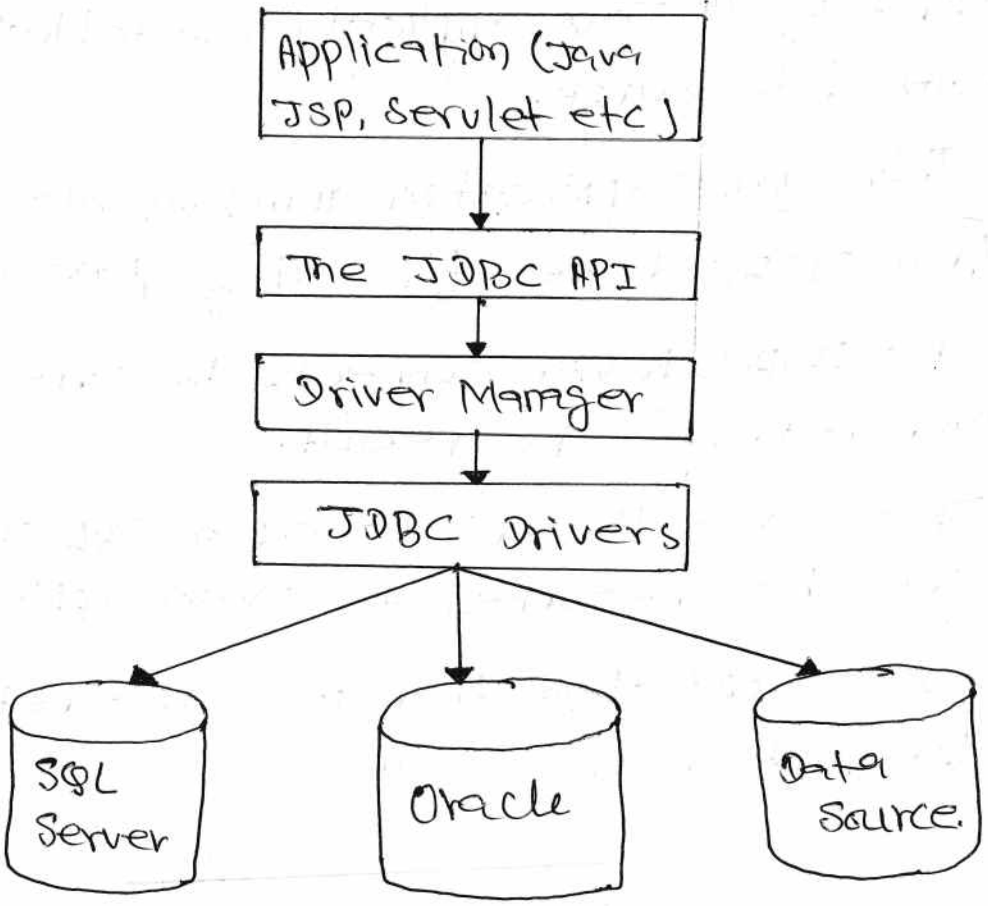


fig: JDBC Architecture.

As shown in the figure the JDBC API provides classes and interface to handle database specific calls from user, made with the help of application. For this it uses the predefined classes and interface like

~~JDBC~~ DriverManager
Driver
Connection
Statement
PreparedStatement

ResultSetMetaData
SqlData
Blob
Clob.

The next layer, DriverManager, plays an important role in JDBC architecture. It uses some database specific drivers to effectively connect enterprise applications to database.

The JDBC driver supports the data sources such as Oracle, and SQL has to be added in Java application for JDBC support, which can be done dynamically in the run time. The dynamic plugging of the JDBC driver ensures that the java application is vendor independent.

JDBC Drivers:

There are different types of the JDBC drivers. They are:-

i) Type-1 Driver:

The Type-1 Driver acts as a bridge between JDBC and other database connectivity mechanisms, such as ODBC. This helps the java programmers to use JDBC and develop java applications to communicate with existing data source. The driver converts the JDBC calls into ODBC calls and redirects the request to ODBC driver. It is included in the package sun.jdbc.odbc.

Eg: Sun JDBC-ODBC drivers.

The figure below shows the architecture of

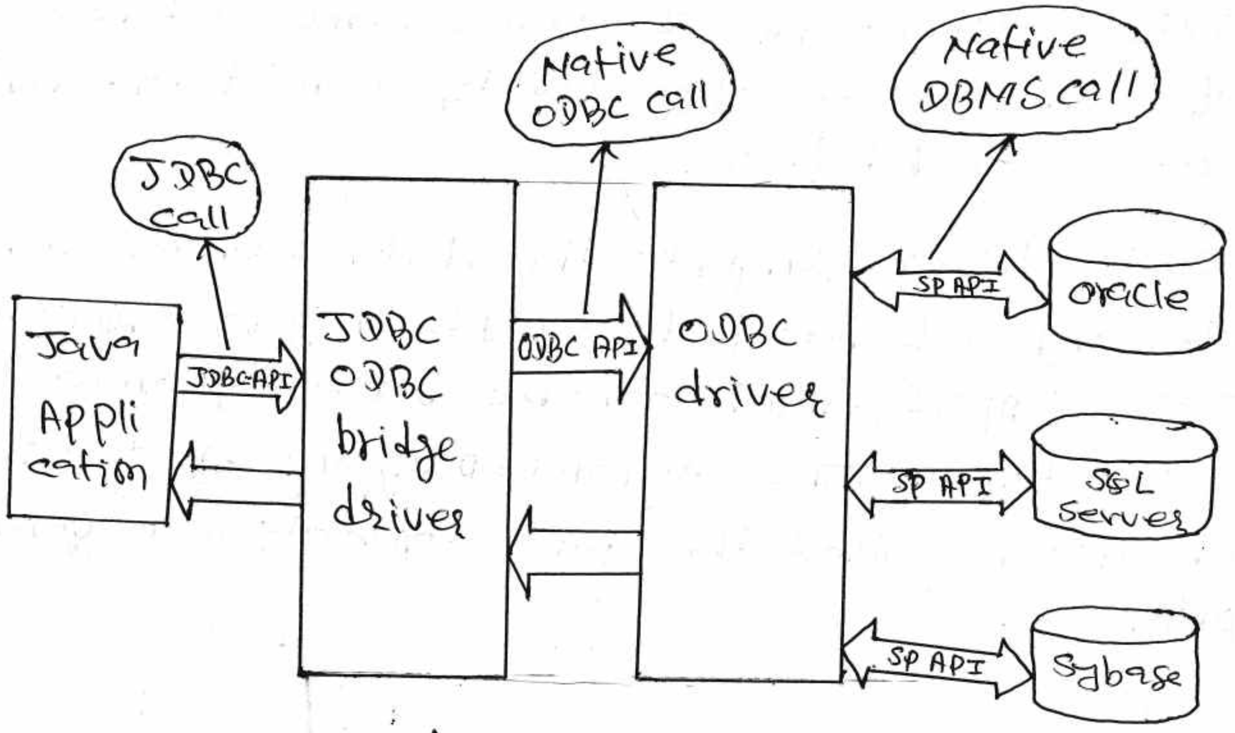


fig: Architecture of JDBC Type-1

The SP API refers to the API used to make Native DBMS calls. Here the following steps are involved in establishing connection between java application and data source:-

(i) The JDBC makes call to JDBC-ODBC bridge to access a data source.

(ii) It converts the JDBC call to equivalent ODBC call to ODBC driver.

(iii) The ODBC driver completes the request and sends request to JDBC-ODBC bridge driver.

(iv) The JDBC-ODBC converts the response to JDBC standards and display the result to the requesting java application.

Advantages:

[illegible]

supported by ODBC.

(iii) Represents a vendor independent drivers

Disadvantages:

(i) Decrease the execution speed.

(ii) Depends on ODBC driver; and therefore, java applications also become indirectly dependent on ODBC drivers.

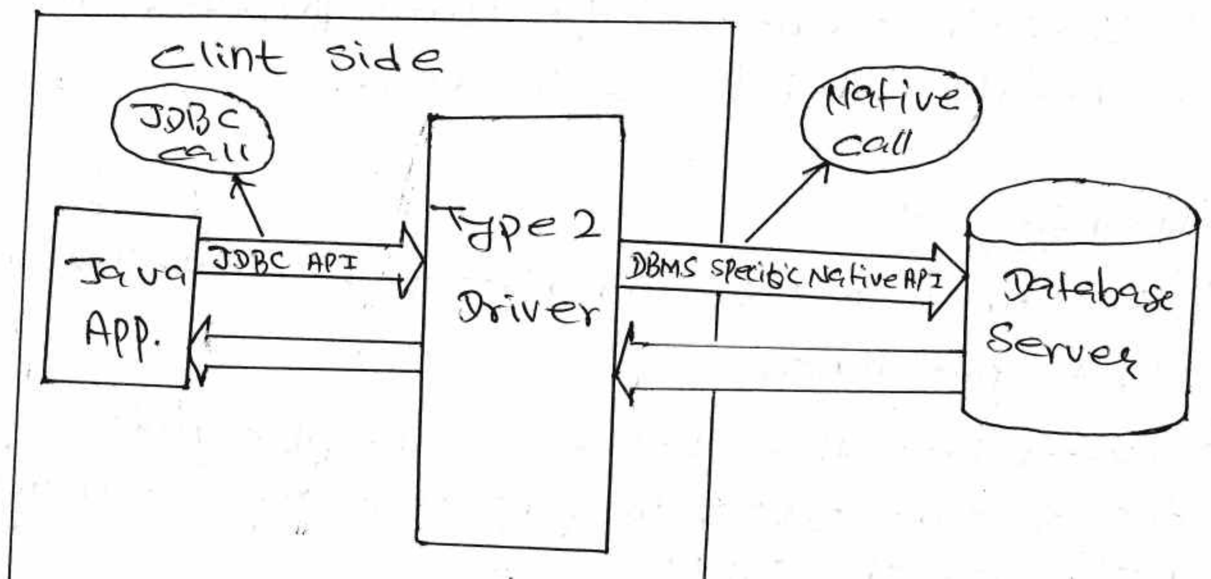
(iii) Requires ODBC binary code or clint library that must be installed on every clint.

(iv) Uses Java Native Interface to make ODBC calls.

(2) Type-2 Drivers (Java to Native API):

The JDBC call is converted to the database vendor specific native call with the help of Type-2 driver. In other words, this types of driver makes Java Native Interface call on specific native clint API, usually written in C and C++.

It follows the 2-tier architecture model.



'Here the Java application needs to communicate with the database is programmed using JDBC API. These calls are converted to database specific native call in the clint machine and request is dispatched to database specific native libraries. And these native libraries send the request to database server using native protocol.

This type of driver are implemented for specific database and usually delivered by DBMS vendor. It is recommended to use to use this type of driver with server side application. And it is also not mandatory to be implemented by DBMS vendor.

Eg: OCI (oracle call interface) driver.
weblogic OCI driver for Oracle.
Type-2 driver for sybase.

Advantages:

1) faster access to data compared to other types of drivers.
2) Contain additional features provided by specific database vendors.

Disadvantages:

1) Requires native libraries to be installed on clint machines.
2) Executes the database specific native functions on clint JVM, implying that any bug in Type-2 might crash JVM.

(3) Type-3 Driver (Java to network protocol / All Java Driver)

The Type-3 Driver translates the JDBC calls into a database server independent and middleware server specific calls. With the help of the middleware server, the translated JDBC calls are further translated into database server specific calls.

It follows the 3-tier architecteure.

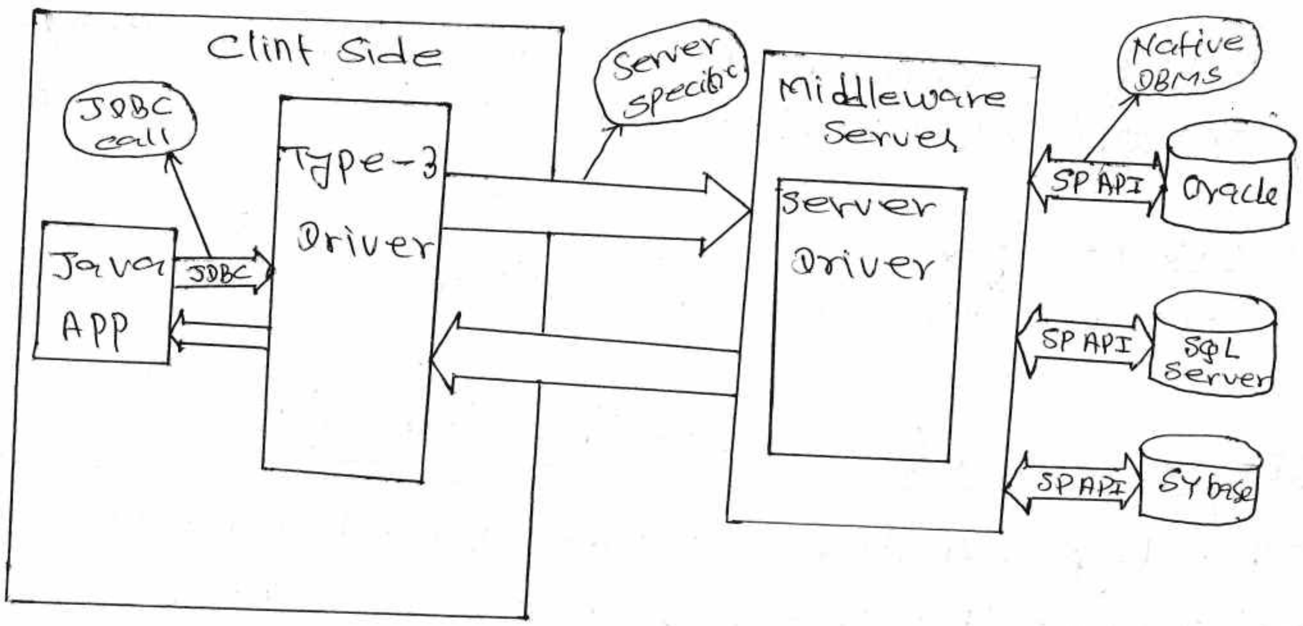


fig: Architecture of Type-3 Drivers.

It translates the JDBC calls to middleware server specific calls. After that it communicates with middleware server over a socket. And the middleware server converts these calls into database specific calls.

It is also known as "net-protocol fully Java technology-enabled" or "net protocol drivers."

Additional functionality such as pool management, performance improvement and connection availability can be added.

It is recommended to be used with applets and is more useful in enterprise applications.

Advantages:

i) Serves as all java driver & is auto downloadable.

2) Don't require any native library to be installed on clint machine.

3) Ensures database independency.

4) Doesnot provide database details i.e username, passwords, location, automatically configured in the middleware server.

5) Provides switching between database without changing clint side driver classes.

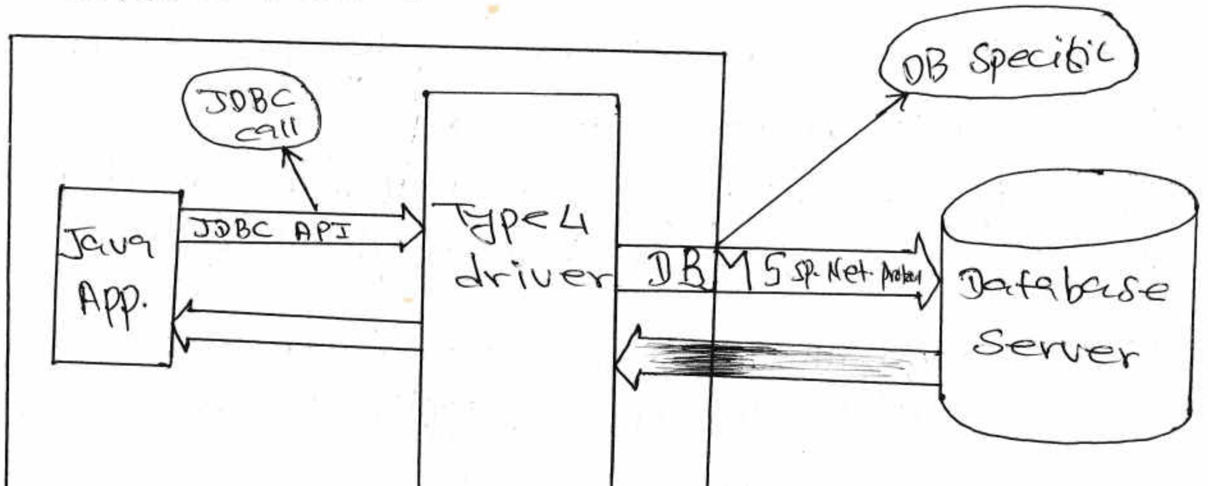
Disadvantage:

1) It is slower and Costlier.

4) Type-4 Driver (Java to Database Protocol):

The Type-4 driver is pure Java driver, which implements the database protocol to interact directly with database. It doesnot require any native database library to retrive records from database.

Type-4 Driver translates JDBC calls to database specific network calls. And follows 2 tier architecture model.



Type-4 Driver prepares DBMS specific network message and then communicates with database server over a Socket. This type of driver is light weight and generally known as thin driver.

Type-4 Driver uses database specific proprietery protocol for communication and implemented by DBMS vendor.

It can also be used in server side and when we want autodownloadable for clint side.

Eg: Thin driver for Oracle, Weblogic & Mssqlserver4 for MS SQL.

Advantages:

1) Serves as pure Java driver and is auto downloadable.

2) Doesn't require any native library to be installed on clint.

3) Uses database server specific protocol.

4) Doesn't require middleware server.

Disadvantage:

1) It is vendor dependent.

Connection Interface:

The connection interface is used to connect the java application with particular database. After creating the connection with the database we can execute SQL statements for that particular connection using object of connection and retrive the result.

to the database temporarily or permanently.

The Some methods are given below:

Method	Description.
void close()	This method frees the object of type Connection from database and other JDBC resource.
void commit()	This method makes all the changes made since last commit or rollback permanent. It throws SQLException.
Statement createStatement()	This method creates an object type Statement for sending SQL statements to database. It throws SQLException.
boolean isClosed()	returns true if connection is close else return false.
CallableStatement prepareCall(String s)	This method creates an object of type Callable Statement for calling stored procedure from database. It throws SQLException.
PreparedStatement prepareStatement(String s)	This method creates an object of type PrepareStatement for sending dynamic SQL statements to database It throws SQLException.
void rollback()	This method undoes all changes made to database.

JDBC API:

If any java application wants to connect with the database then there are various interfaces and classes available in java.sql package.

Depending on requirement these classes and interface can be used.

Some of them are:

class or Interface	Description.
java.sql.Connection	Creates connection with specific database.
java.sql.DriverManager	The task of driver manager is to manage database driver.
java.sql.Statement	It executes SQL statement for particular connection and retrives result.
java.sql.PreparedStatement	It allows programmer to create prepared SQL statements.
java.sql.CallableStatement	It executes stored procedure.
java.sql.ResultSet	It provides methods to get result row by row generated by select statements.

The Prepared Statement Interface:

The prepared statement is used to execute a dynamic

IN parameter:

In Some situation where we need to pass different value to an query then such values can be specified as a "?" in the query and the actual values can be passed using set XXX() method at the time of execution.

Syntax

setXXX (integer data, XXX value);

where XXX means data type as per the value we want to pass in query.

Eg:

```
String query ="Select * from Data where ID=? and
 Name = ? ";
PreparedStatement ps = con.prepareStatement(query);
        ps.SetInt (1, 1);
        ps. SetString ( 2, " Tek Raj ");
```

The prepared statement interface has Several methods to execute the parameterized SQL statements and retrive appropriate results as per query sent to database.

Some of them are:

Method	Description.
void close()	It frees the object of type

boolean execute()	This method executes dynamic query in the object of type Prepared statement. The ~~get~~ getResult() method is used to retrive the result.
ResultSet executeQuery()	This method is used to execute the dynamic query in the object of type Prepared Statement and returns the object of type ResultSet.
Int executeUpdate()	This method executes SQL Statement in the object of type prepared statement. The SQL statement may be SQL insert, update and delete statement.
ResultSetMetaData getMetaData()	The ResultsetMetaData means a data about the data of ResultSet. This method retrives an object of type ResultSetMeta-Data that contains information about the columns of ResultSet object that will be return when query is execute.
int getMaxRows()	This method returns the maximum number of rows those are generated by executeQuery() method.

Example programs:

(i) TYPE 1 Driver:

a. JDBC example ~~with access~~: ~~DSN~~:

create the DSN (data Source name).

The process to create DSN are:-

1. Go to control panel
2. Go to Administrative tool (XP) for (windows 7) System & security then Administrative tools.
3. Click on ODBC. Data Source.
4. Select MS Access Database and click on Add button.
5. Select Microsoft Access driver (*.mdb,*.accdb) and click on finish.
6. Type data source name and click on select.
7. Select particular database created at begining.

<u>Program to establish Connection</u>

```
import java.sql.*;
class typeone
{
   public static void main(String args[])
   { try{
      sun.jdbc.odbc.JdbcOdbcDriver obj = new
       sun.jdbc.odbc.JdbcOdbcDriver();

      ~~sun.jdbc.odbc.JdbcOdbc~~
      DriverManager.registerDriver(obj);

      Connection con = DriverManager.getConnection
        ("jdbc:odbc:sdata", "scott", "tiger");
                        → DSN name.
      if (con == null)
      {
```

```
    else
    {
          ~~conne~~
          System.out.println("Success");
    }
    con.close();
  }
  catch(Exception e)
  {
       System.out.println(e);
  }
 }
}
```

Here

```
sun.jdbc.odbc.JdbcOdbcDriver obj = new sun.jdbc.odbc.Jdbc
    OdbcDriver() ;
DriverManager.register(obj);
```

is used to load the driver.

The equivalent statements for this are:

~~ii)~~ `class.forName("sun.jdbc.odbc.JdbcOdbcDriver");`

And next is to pass Jdbc driver class at runtime.

```
        class.forName(args[0]).
```

And type the following while running program.

```
java typeone sun.jdbc.odbc.JdbcOdbcDriver
```

$\swarrow$

is passed as parameter of args[0]

Program to create, Insert and Retrive the result.

```
import java.sql.*;
import java.io.*;
import java.util.*;

class TypeOne
{
    public static void main(String args[])
    {
        Connection con = null;
        class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        System.out.println("Driver loaded");
        con = DriverManager.getConnection("jdbc:odbc:oradsn",
         "scott","tiger");
        if (con == null)
        {
            System.out.println("Connection error");
        }
        else
            System.out.println("Success");

        Statement st = con.createStatement();
        String query = "Create table student (Regno number,
                        Name varchar(10))";
         st.executeQuery(query);
         System.out.println("Table Created");
         String query1 = "Insert into student values(40,
System.out.println("value inserted");
ResultSet rs=st.executeQuery("Select * from Student");
System.out.println("values are");
while (rs.next())
{
    System.out.println(rs.getInt(1)+" "+rs.getString
        ( 2));
}
  rs.close();
  st.close();
  con.close();
}
}
```

In order to access the data using Microsoft Access we need not provide username & password, if not provided while creating DSN.

Prepared statement example: ~~using access~~:

```
import java.sql.*;
import java.io.*;
import java.util.*;
public class Prepare_Demo
{
  public static void main (String[] args)
  {
    try
    {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
PreparedStetement ps= con.prepare Statement("insert into
  Student (ID, Name) values (?,?)");

   ps.setInt (1,200);
   ps.SetString (2,"Hari");
   ps.executeUpdate();
   system.out.println(~~executed~~ "Inserted");
   con.close();
 }
 catch (Exception e)
 {
      System.out.println(e);
 }
}
}
```

(ii) TYPE 4 Driver :

Program to connect to the database using type 4 driver. using ORacle.

```
import java.sql.*;
import java.io.*;
class type4
{
   Public static void main(String args[])
   {
       Connection con = null;
```

```
Class.forName("oracle.jdbc.driver.OracleDriver");
System.out.println("Driver loaded");
con = Driver manager.getConnection("jdbc:oracle:thin:@
    localhost: 1521: XE", "tekraj", "laxman");
if (con == null)
{
    System.out.println("Connection error\n");
}
else
    System.out.println("Connection established");

Statement st = con.CreateStatement();
ResultSet rs = st.executeQuery("Select * from t1");
while(rs.next())
{
    System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
rs.close();
st.close();
con.close();
}
catch(SQLException se)
{
    se.printStackTrace();
}
Catch (ClassNotfound Exception e)
{
    e.printStackTrace();
```

```
        e1.printStackTrace();
    }
  }
}
```

Type 4 driver is called Oracle thin driver. It automatically comes when we install the Oracle software. Generally this driver is available in the form of jar or zip files. Technical name is Oracle thin driver. Mechanism is Type 4, vendor is Oracle coorporation.

JDBC driver class is:

oracle.jdbc.driver.OracleDriver

JDBC Url :

<u>Syntax:</u>

Jdbc:oracle:thin:@ host name/ IP address : port no: Service ID/ SID

$\underbrace{\text{Jdbc:oracle:thin:@localhost}}_{\text{name of machine where oracle database software is installed.}}:\underbrace{1521}_{\text{Default port no. for oracle software}}:\underbrace{XE}_{\text{Service ID}};$

To know SID of Oracle type:

Select * from global_name;

JDBC - MySqL

1. Program to connect to database through MySql and retrive result.

```
import java.sql.*;
public class MysqlDemo
{
   public static void main(String []args)
   {
     try
     {
       Class.forName("com.mysql.jdbc.Driver");
       System.out.println("Driver loaded");
       Connection c = DriverManager.getConnection("
         Jdbc:mysql://localhost:3306/college","root","123");
       Statement s= c.createStatement();
       ResultSet rs = s.executeQuery("select * from
            data");
       System.out.println("ID \t Name");
       while (rs.next())
       {
          System.out.println(rs.get~~S~~Int(1)+"\t"+ rs.get
                String(2));
       }
   }
   catch(Exception e)
   {
```

Here college is the name of the database. and 3306 is the portnumber.

Callable Statement in MySQL :

For the callableStatement first of all we must have stored procedure in our database.

Steps to create stored procedure in MySQL:

- first create the database and make use of it.
- Example of stored procedure for addition of two number.

```
DELIMITER $$
    CREATE PROCEDURE 'adproc'(IN a INT, IN b INT,
     OUT c INT).
    BEGIN
        SET c=a+b;
    END $$
```

After this if we get message like Query ok, 0 rows affected, means stored procedure created successfully.

<u>JDBC program to create stored procedure to add two integer.</u>

```
import java.sql.*;
Public class callable
{
```

```
try
{
    Class.forName("com.mysql.jdbc.Driver");
    System.out.println("Driver loaded");
    Connection c= DriverManager.getConnection ("jdbc:
     mysql://localhost:3360/tekraj", "root" ,"123");
    System.out.println("Connection created");
    String q = "call adproc (?,?,?)";
    CallableStatement cs = c.prepareCall(q);
      cs.SetInt(1,10);
      cs.SetInt (2,20);
      cs.registerOutParameter(3, Types.INTEGER);
      cs.execute();
      int add = cs.getInt(3);
      System.out.println("addition ="+ add);
}
catch(Exception e)
{
      System.out.println(e);
}
  }
}
```

Stored Procedure to retrive name on passing ID.

① Stored procedure:

```
DELIMITER $$
   CREATE PROCEDURE 'ret'( IN SID INT, OUT NAME
          varchar(20) )
   BEGIN
```

Java Program:

```
import java.sql.*;
import ~~java.util~~ java.util.Scanner*;

Public class call_sql
{
    Public static void main(String args[])
    {
        int i;
        try
        {
            Scanner s= new Scanner (System.in);
            System.out.println ("Enter ID");
             i= s.nextInt();
            Class.forName("com.mysql.jdbc.Driver");
            Connection con= DriverManager.getConnection
             ("jdbc:mysql://localhost:3360/tekraj","A1","2");
            String q= "~~ret~~ call ret ( ?, ? )";
            CallableStatement cs= con.prepareCall(q);
             cs.setInt(1,i);
                                      Types
            cs.registerOutParameter (2, ^.VARCHAR);
            cs.execute();
            String nm= cs.getString (2);
            System.out.println("Name: " + nm);
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
```