Standard Steps for developing JDBC Application

1. Load and register Driver Class
2. Establish Connection between Java Application and Database
3. Create Statement Object
4. Send and execute SQL Query
5. Process Result from ResultSet
6. Close Connection

Step 1: Load and Register Driver Class

JDBC API is a Set of Interfaces defined by Java Vendor.
Database Vendor is responsible to provide Implementation. This Group of Implementation Classes is nothing but "Driver Software".

We have to make this Driver Software available to our Java Program. For this we have to place corresponding Jar File in the Class Path.

Note:

Type-1 Driver is available as the Part of JDK and hence we are not required to set any Class Path explicitly.

Every Driver Software is identified by some special Class, which is nothing but Driver Class.

For Type-1 Driver, the corresponding Driver Class Name is

```
sun.jdbc.odbc.JdbcOdbcDriver
```

We can load any Java Class by using *Class.forName()* Method. Hence by using the same Method we can load Driver Class.

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Whenever we are loading Driver Class automatically Static Block present in that Driver Class will be executed.

```
1)  class JdbOdbcDriver
2)  {
3)     static
4)     {
5)        JdbOdbcDriver driver= new JdbOdbcDriver();
6)        DriverManager.registerDriver(driver);
7)     }
8)  }
```

Because of this Static Block, whenever we are loading automatically registering with *DriverManager* will be happened. Hence we are not required to perform this activity explicitly.

If we want to register explicitly without using *Class.forName()* then we can do as follows by using *registerDriver()* Method of *DriverManager* Class.

```
JdbOdbcDriver driver= new JdbOdbcDriver();
DriverManager.registerDriver(driver);
```

Note: From JDBC 4.0 V (Java 1.6 V) onwards Driver Class will be loaded automatically from Class Path and we are not required to perform this step explicitly.

Step-2: Establish Connection between Java Application and Database

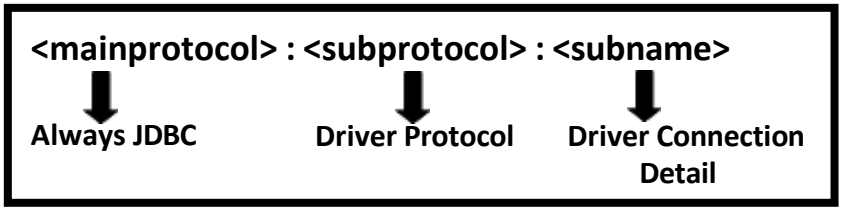
Once we loaded and registered Driver, by using that we can establish Connection to the Database. For this *DriverManager* Class contains *getConnection()* Method.

```
public static Connection getConnection(String jdbcurl, String username, String pwd) throws
SQLException
```

Eg: Connection con= DriverManager.getConnection(jdbcurl,username,pwd);

"Jdbcurl" represents URL of the Database.
***username* and *pwd* are Credentials to connect to the Database.**

JDBC URL Syntax:



For Type-1 Driver, JDBC URL is: `jdbc:odbc:demodsn`

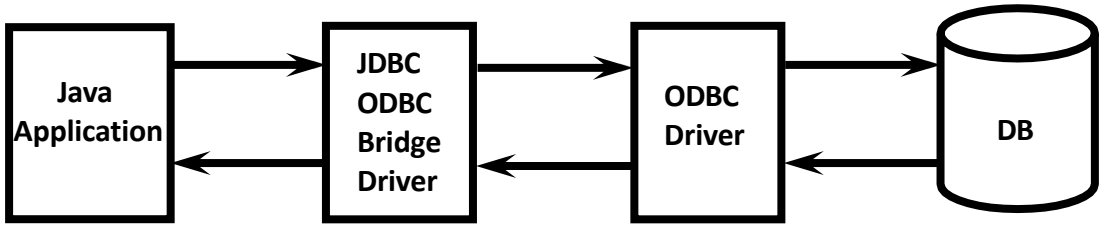
Eg: Connection con= DriverManager.getConnection("jdbc:odbc:demodsn","scott","tiger");

Note:

DriverManager will use Driver Class internally to connect with Database.
DriverManager Class *getConnection()* Method internally calls Driver Class *connect()* Method.

DSN (Data Source Name) for Type-1 Driver:

Internally Type-1 Driver uses ODBC Driver to connect with Database.



ODBC Driver needs Database Name & its Location to connect with Database.

ODBC Driver collect this Information from DSN i.e. internally ODBC Driver will use DSN to get Database Information (DSN Concept applicable only for Type-1 Driver)

There are 3 Types of DSN

1. User DSN
2. System DSN
3. File DSN

1) User DSN:

It is the non-sharable DSN and available only for Current User.

2) System DSN:

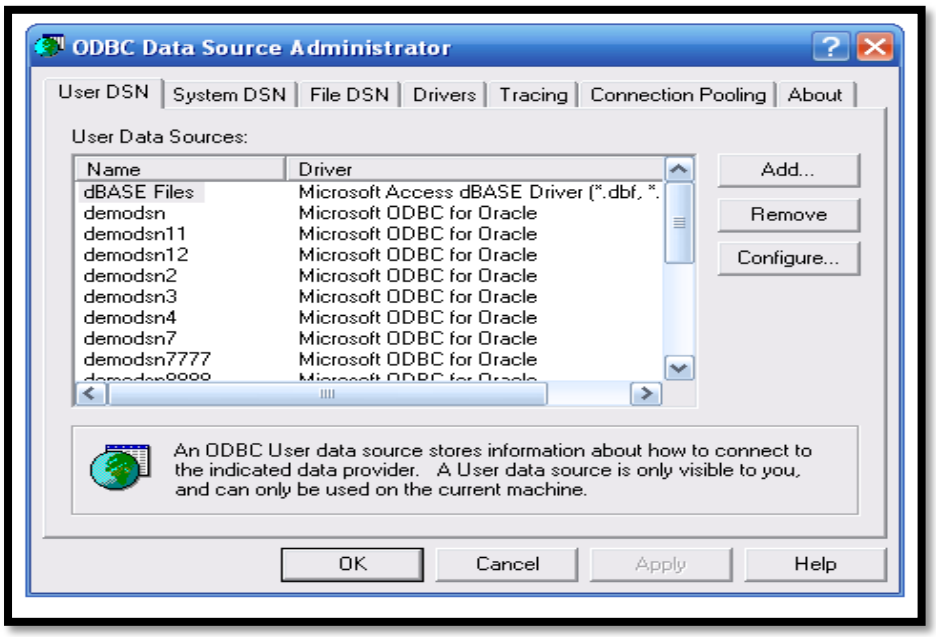
It is the sharable DSN and it is available for all Users who can access that System.
It is also known as Global DSN.

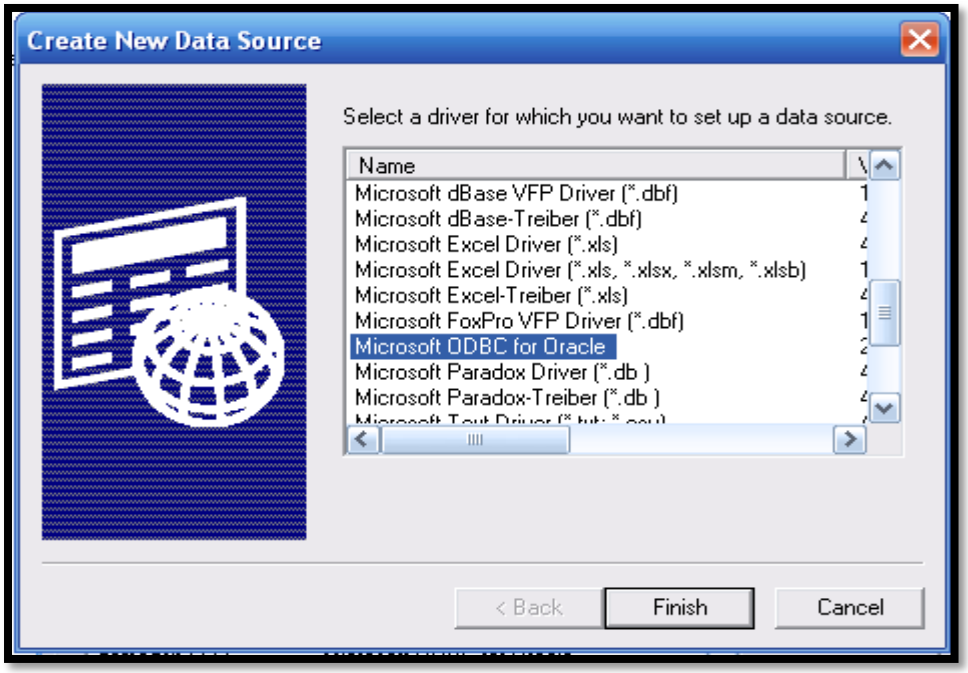
3) File DSN:

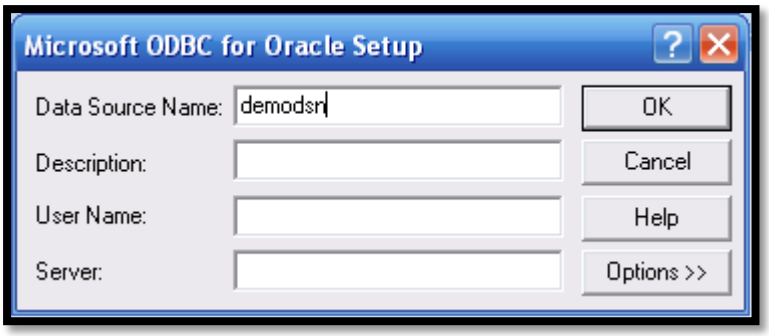
It is exactly same as User DSN but will be stored in a File with .dsn Extension.

Steps to configure DSN:

Start ➔ Settings ➔ Control Panel ➔ Performance and Maintenance ➔ Administrative Tools ➔ Data Sources (ODBC) ➔ Add ➔ Microsoft ODBC for Oracle ➔ Finish







For Windows 7 OR 8 OR 10:

C:\Windows\Syswow64 OR System32\Odbcad32.Exe ➔ Add ➔ Microsoft ODBC For Oracle ➔ Finish

Write A Java Program To Establish Connection To The Oracle Database By Using Type-1 Driver?

```
1)  import java.sql.*;
2)  public class DbConnectDemo1
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
7)        Connection con=DriverManager.getConnection("jdbc:odbc:demodsn7","scott","tiger"
    );
8)        if(con != null)
9)        {
10)          System.out.println("Connection established Successfully");
11)       }
12)       else
13)       {
14)          System.out.println("Connection not established");
15)       }
16)    }
17) }
```

In the above Program Line 1 is Optional, because from JDBC 4.0V/ Java 1.6V onwards Driver Class will be loaded automatically from the Class Path based on "jdbcurl".

Note:

To Compile and Run above Program we are not required to Place/Set any Jar File in the Class Path, because Type-1 Driver is available by default as the Part of JDK.

***Q. Connection is an interface, then how we can get Connection Object?

We are not getting Connection Object and we are getting its Implementation Class Object.

This Implementation Class is available as the Part of Driver Software. Driver Software Vendor is responsible to provide Implementation Class.

We can print corresponding Class Name as follows SOP(con.getClass().getName());
o/p: sun.jdbc.odbc.JdbcOdbcConnection

Q.What Is The Advantage Of Using Interface Names In Our Application Instead Of Using Implementation Class Names?

Interface Reference can be used to hold implemented Class Object. This Property is called Polymorphism.

Connection ➔ sun.jdbc.odbc.JdbcOdbcConnection ➔ Type-1
Connection ➔ orcale.jdbc.OracleT4Connection ➔ Type-2

In JDBC Programs, Interface Names are fixed and these are provided by JDBC API. But Implementation Classes are provided by Driver Software Vendor and these Names are varied from Vendor to Vendor.

If we Hard Code Vendor provided Class Names in our Program then the Program will become Driver Software Dependent and won't work for other Drivers.

If we want to change Driver Software then Total Program has to rewrite once again, which is difficult. Hence it is always recommended to use JDBC API provided Interface Names in our Application.

Step-3: Creation of Statement Object

Once we established Connection between *Java Application* and *Database*, we have to prepare SQL Query and we have to send that Query to the Database. Database Engine will execute that Query and send Result to Java Application.

To send SQL Query to the Database and to bring Results from Database to Java Application some Vehicle must be required, which is nothing but Statement Object.

We can create Statement Object by using *createStatement()* Method of Connection Interface.

```
public Statement  createStatement();
```

Eg: Statement st = con.createStatement();

Step-4: Prepare, Send and Execute SQL Query

According to Database Specification, all SQL Commands are divided into following Types...

1. **DDL (Data Definition Language) Commands:**
   Eg: Create Table, Alter Table, Drop Table Etc

2. **DML (Data Manipulation Language) Commands:**
   Eg: Insert, Delete, Update

3. **DQL (Data Query Language) Commands:**
   Eg: Select

4. **DCL (Data Control Language) Commands:**
   Eg: Alter Password, Grant Access Etc..

5. **Data Administration Commands**
   Eg: Start Audit
   Stop Audit

6. **Transactional Control Commands**
   Commit, Rollback, Savepoint Etc

According to Java Developer Point of View, all SQL Operations are divided into 2 Types...

1. **Select Operations (DQL)**
2. **Non-Select Operations (DML, DDL Etc)**

Basic SQL Commands

1) To Create a Table:

```
Create table movies (no number, name varchar2(20),hero varchar2(20),heroine varchar2(20));
```

2) To Drop/Delete Table:

```
drop table movies;
```

3) To Insert Data:

```
insert into movies values(1,'bahubali2','prabhas','anushka');
```

4) To Delete Data:

```
delete from movies where no=3;
```

5) To Update Data:

```
update movies set heroine='Tamannah' where no=1;
```

Select Operations and Non-Select Operations

Select Operations:

Whenever we are performing Select Operation then we will get a Group of Records as Result.

Eg: `select * from movies;`

Non-Select Operations:

Whenever we are performing Non-Select Operation then we will get Numeric Value that represents the Number of Rows affected.

Eg: `update movies set heroine='Tamannah' where no=1;`

Once we create Statement Object, we can call the following Methods on that Object to execute our Queries.

1. executeQuery()
2. executeUpdate()
3. execute()

1) executeQuery() Method:

We can use this Method for Select Operations.

Because of this Method Execution, we will get a Group of Records, which are represented by ResultSet Object.

Hence the Return Type of this Method is ResultSet.

```
public ResultSet executeQuery(String sqlQuery) throws SQLException
```

Eg: `ResultSet rs = st.executeQuery("select * from movies");`

2) executeUpdate() Method:

We can use this Method for Non-Select Operations (Insert|Delete|Update)
Because of this Method Execution, we won't get a Group of Records and we will get a Numeric Value represents the Number of Rows effected. Hence Return Type of this Method is int

```
public int executeUpdate(String sqlQuery)throws SQLException
```

Eg: int rowCount = st.executeUpdate("delete from employees where esal>100000");
SOP("The number of employees deleted:"+rowCount);

3) execute() method:

We can use this Method for both Select and Non-Select Operations.

If we don't know the Type of Query at the beginning and it is available dynamically at runtime then we should use this execute() Method.

```
public boolean execute(String sqlQuery)throws SQLException
```

Eg:

```
1)  boolean b = st.execute("dynamically provided query");
2)
3)  if(b==true)//select query
4)  {
5)     ResultSet rs=st.getResultSet();
6)     //use rs to get data
7)  }
8)  else// non-select query
9)  {
10)    int rowCount=st.getUpdateCount();
11)    SOP("The number of rows effected:"+rowCount);
12) }
```

executeQuery() Vs executeUpdate() Vs execute():

1. If we know the Type of Query at the beginning and it is always Select Query then we should use "*executeQuery()* Method".

2. If we know the Type of Query at the beginning and it is always Non-Select Query then we should use *executeUpdate()* Method.

3. If we don't know the Type of SQL Query at the beginning and it is available dynamically at Runtime (May be from *Properties File* OR From *Command Prompt* Etc) then we should go for execute() Method.

Note:

Based on our Requirement we have to use corresponding appropriate Method.

- st.executeQuery();
- st.executeUpdate();

- st.execute();
- st.getResultSet();
- st.getUpdateCount();

Case-1: executeQuery() Vs Non-Select Query

Usually we can use executeQuery() Method for Select Queries. If we use for Non-Select Queries then we cannot expect exact Result. It is varied from Driver to Driver.

```
ResultSet rs =st.executeQuery("delete from employees where esal>100000");
```

For Type-1 Driver we will get SQLException. But for Type-4 Driver we won't get any Exception and Empty ResultSet will be returned.

Case-2: executeUpdate() Vs Select Query

Usually we can use *executeUpdate()* Method for Non-Select Queries. But if we use for Select Queries then we cannot expect the Result and it is varied from Driver to Driver.

```
int rowCount=st.executeUpdate("select * from employees");
```

For Type-1 Driver we will get *SQLException* where as for Type-4 Driver we won't get any Exception and simply returns the Number of Rows selected.

Case-3: executeUpdate() Vs DDL Queries

If we use *executeUpdate()* Method for DDL Queries like Create Table, Alter Table, Drop Table Etc, then Updated Record Count is not applicable. The Result is varied from Driver to Driver.

```
int rowCount=st.executeUpdate("create table employees(eno number,ename varchar2(20)");
```

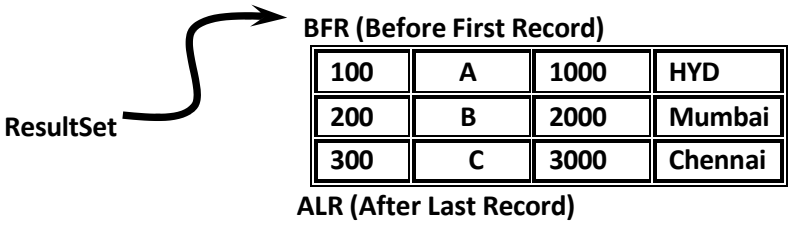
For Type-1 Driver, we will get -1 and For Type-4 Driver, we will get 0

```
st.executeUpdate("create table employees(eno number,ename varchar2(20)");
```

Step-5: Process Result from ResultSet

After executing Select Query, Database Engine will send Result back to Java Application. This Result is available in the form of ResultSet.

i.e. ResultSet holds Result of executeQuery() Method, which contains a Group of Records.
By using ResultSet we can get Results.



ResultSet is a Cursor always locating Before First Record (BFR).
To check whether the next Record is available OR not, we have to use *rs.next()* Method.

```
public  boolean next()
```

This Method Returns True if the next Record is available, otherwise returns False.

```
1)  while(rs.next())
2)  {
3)     read data from that record
4)  }
```

If next Record is available then we can get Data from that Record by using the following Getter Methods.

1. getXxx(String columnName)
2. getXxx(int columnIndex)

Like getInt(), getDouble(), getString() etc..

Note:
In JDBC, Index is always one based but not Zero based i.e. Index of First Column is 1 but not 0.

```
1)  while(rs.next())
2)  {
3)    SOP(rs.getInt("ENO")+".."+rs.getString("ENAME")+".."+rs.getDouble("ESAL")+".."+rs.getS
    tring("EADDR"));
4)          OR
5)    SOP(rs.getInt(1)+".."+rs.getString(1)+".."+rs.getDouble(3)+".."+rs.getString(4));
6)   }
```

Note:

Readability wise it is recommended to use Column Names, but Performance wise it is recommended to use Column Index. (Because comparing Numbers is very easy than comparing String Values)

Hence if we are handling very large Number of Records then it is highly recommended to use Index.

If we know Column Name then we can find corresponding Index as follows...

```
int columnIndex=rs.findColumn(String columnName);
```

Conclusions:

1. ResultSet follows "Iterator" Design Pattern.
2. ResultSet Object is always associated with Statement Object.
3. Per Statement only one ResultSet is possible at a time. if we are trying to open another ResultSet then automatically first ResultSet will be closed.

Eg:

```
Statement st = con.createStatement();
RS rs1 = st.executeQuery("select * from movies");
RS rs2 = st.executeQuery("select * from employees");
```

In the above Example Rs1 will be closed automatically whenever we are trying to open Rs2.

Step 6: Close the Connection

After completing Database Operations it is highly recommended to close the Resources whatever we opened in reverse order of opening.

1. rs.close();

It closes the ResultSet and won't allow further processing of ResultSet

2. st.close();

It closes the Statement and won't allow sending further Queries to the Database.

3. con.close();

It closes the Connection and won't allow for further Communication with the Database.

Conclusions:

- **Per Statement only one ResultSet is possible at a time.**
- **Per Connection multiple Statement Objects are possible.**
- **Whenever we are closing Statement Object then automatically the corresponding ResultSet will be closed.**
- **Similarly, whenever we are closing Connection Object automatically corresponding Statement Objects will be closed.**
- **Hence we required to use only *con.close();***