INDEX

ADVANCED JAVA

With Core Java knowledge we can develop Stand Alone Applications.

The Applications which are running on a Single Machine are called *Stand Alone Applications.*
Eg: Calculator, MS Word
Any Core Java Application
If we want to develop Web Applications then we should go for Advanced Java.

The Applications which are providing Services over the Web are called *Web Applications.*

Eg: durgasoftvideos.com, gmail.com, facebook.com, durgasoft.com

In Java we can develop Web Applications by using the following Technologies...

- JDBC
- Servlets
- JSP's

Where ever Presentation Logic is required i.e. to display something to the End User then we should go for JSP i.e. JSP meant for View Component.

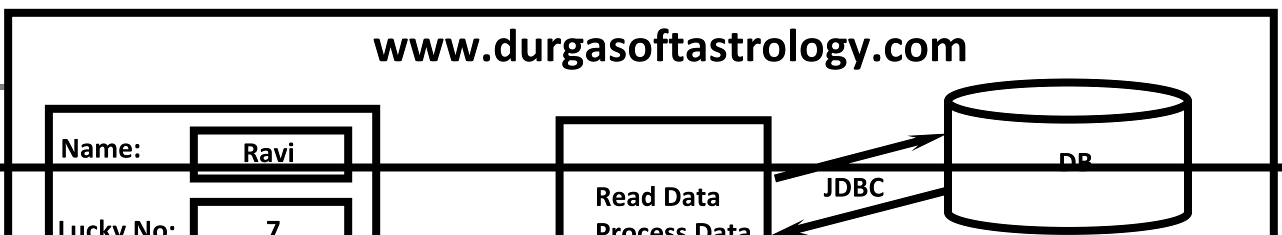
Eg: display login page
display inbox page
display error page
display result page
etc..

Where ever some Processing Logic is required then we should go for Servlet i.e. Servlet meant for Processing Logic/ Business Logic. Servlet will always work internally.

Eg: Verify User
Communicate with Database
Process End User's Data
etc..

From Java Application (Normal Java Class OR Servlet) if we want to communicate with Database then we should go for JDBC.

Eg: To get Astrology Information from Database
To get Mails Information from Database



In Java there are 3 Editions are available

1. Java Standard Edition (JSE | J2SE)
2. Java Enterprise Edition (JEE | J2EE)
3. Java Micro Edition (JME | J2ME)

JDBC is the Part of JSE
Servlets and JSP's are the Part of JEE

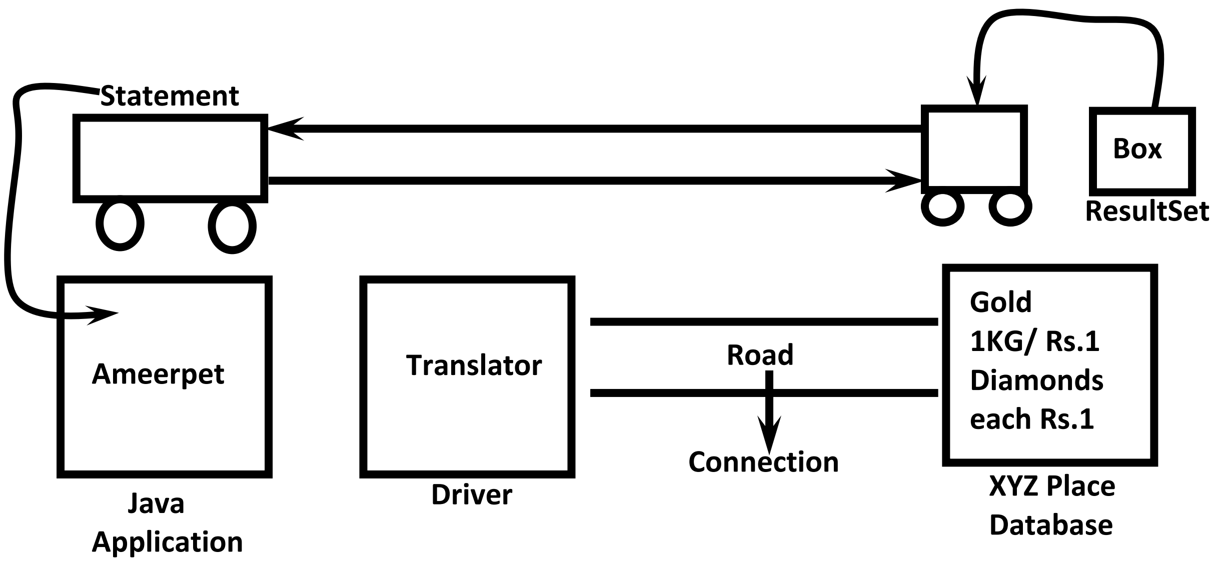
Current Version of JSE is Java 1.8
Current Version of JDBC is: 4.2 V

Current Version of JEE is 7.0
Current Version of Servlets: 3.1 V
Current Version of JSP is: 2.3 V

Current Versions Are:

JDBC 4.2 V
Servlets 3.1 V
JSP's 2.3 V

JDBC in Simple Way



Driver (Translator):

To convert Java specific calls into Database specific calls and Database specific calls into Java calls.

Connection (Road):

By using Connection, Java Application can communicate with Database.

Statement (Vehicle):

By using Statement Object we can send our SQL Query to the Database and we can get Results from Database.

ResultSet:

ResultSet holds Results of SQL Query.

Steps for JDBC Application:

1. Load and Register Driver
2. Establish Connection between Java Application and Database
3. Create Statement Object
4. Send and Execute SQL Query
5. Process Results from ResultSet
6. Close Connection

Demo Program:

```java
import java.sql.*;
public class JdbcDemo
{
    public static void main(String[] args) throws Exception
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con=DriverManager.getConnection("jdbc:odbc:demodsn","scott","tiger");
        Statement st = con.createStatement();
        ResultSet rs= st.executeQuery("select * from employees");
        while(rs.next())
        {
            System.out.println(rs.getInt(1)+".."+rs.getString(2)+".."+rs.getDouble(3)+"..."+rs.getString(4));
        }
        con.close();
    }
}
```

Storage Areas

As the Part of our Applications, we required to store our Data like Customers Information, Billing Information, Calls Information etc..

To store this Data, we required Storage Areas. There are 2 types of Storage Areas.

1) Temporary Storage Areas
2) Permanent Storage Areas

Temporary Storage Areas:

These are the Memory Areas where Data will be stored temporarily.
Eg: All JVM Memory Areas (like Heap Area, Method Area, Stack Area etc).
Once JVM shutdown all these Memory Areas will be cleared automatically.

Permanent Storage Areas:

Also known as Persistent Storage Areas.
Here we can store Data permanently.
Eg: File Systems, Databases, Data warehouses, Big Data Technologies etc

File Systems:

File Systems can be provided by Local operating System.
File Systems are best suitable to store very less Amount of Information.

Limitations:

1) We cannot store huge Amount of Information.
2) There is no Query Language support and hence operations will become very complex.
3) There is no Security for Data.
4) There is no Mechanism to prevent duplicate Data. Hence there may be a chance of Data Inconsistency Problems.

To overcome the above Problems of File Systems, we should go for Databases.

Databases:

1) We can store Huge Amount of Information in the Databases.
2) Query Language Support is available for every Database and hence we can perform Database Operations very easily.
3) To access Data present in the Database, compulsory *username* and *pwd* must be required. Hence Data is secured.

4) Inside Database Data will be stored in the form of Tables. While developing Database Table Schemas, Database Admin follow various Normalization Techniques and can implement various Constraints like Unique Key Constrains, Primary Key Constraints etc which prevent Data Duplication. Hence there is no chance of Data Inconsistency Problems.

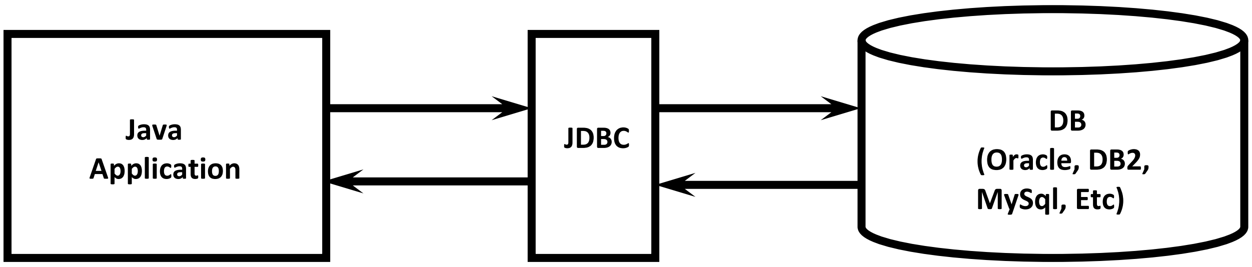
Limitations of Databases:

1) Database cannot hold very Huge Amount of Information like Terabytes of Data.
2) Database can provide support only for Structured Data (Tabular Data OR Relational Data) and cannot provide support for Semi Structured Data (like XML Files) and Unstructured Data (like Video Files, Audio Files, Images etc)

To overcome this Problems we should go for more Advanced Storage Areas like Big Data Technologies, Data warehouses etc..

JDBC

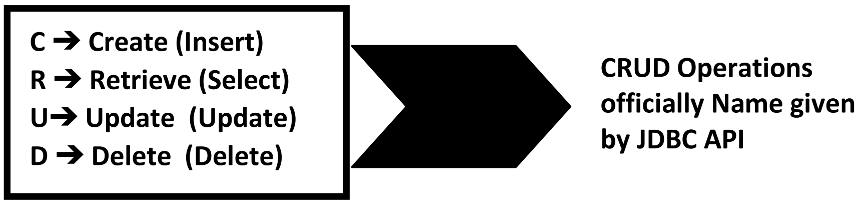
- JDBC is a Technology, which can be used to communicate with Database from Java Application.



- JDBC is the Part of Java Standard Edition (J2SE|JSE)
- JDBC is a Specification defined by Java Vendor (Sun Micro Systems) and implemented by Database Vendors.
- Database Vendor provided Implementation is called "Driver Software".

JDBC Features:

1) JDBC API is Standard API. We can communicate with any Database without rewriting our Application i.e. it is Database Independent API.
2) JDBC Drivers are developed in Java and hence JDBC Concept is applicable for any Platform. i.e. JDBC Is Platform Independent Technology.
3) By using JDBC API, we can perform basic CRUD Operations very easily.



These Operations also known as CURD/ SCUD Operations (Ameerpet People created Terminology)

We can also perform Complex Operations (like Inner Joins, Outer Joins, calling Stored Procedures etc) very easily by using JDBC API.

4) JDBC API supported by Large Number of Vendors and they developed multiple Products based on JDBC API.

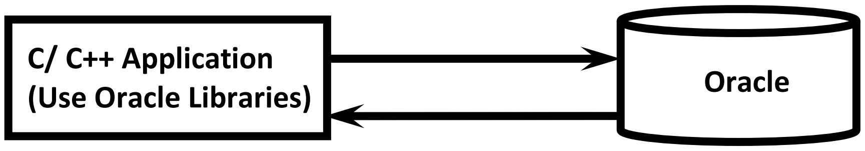
List of supported Vendors we can check in the link
http://www.oracle.com/technetwork/java/index-136695.html

JDBC Versions:

- JDBC 3.0 is Part J2SE 1.4
- No Update in Java SE 5.0
- JDBC 4.0 is Part Java SE 6.0
- JDBC 4.1 is Part Java SE 7.0
- JDBC 4.2 is Part Java SE 8.0

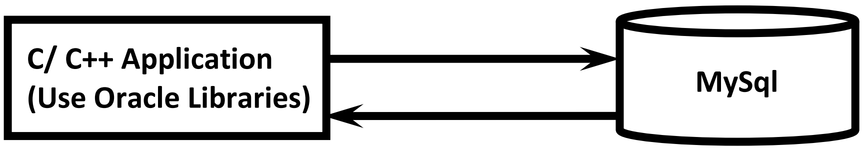
Evolution of JDBC:

- If we want to communicate with Database by using C OR C++, compulsory we have to use database specific Libraries in our Application directly.



- In the above Diagram C OR C++ Application uses Oracle specific Libraries directly.

- The Problem in this Approach is, if we want to migrate Database to another Database then we have to rewrite Total Application once again by using new Database specific Libraries.

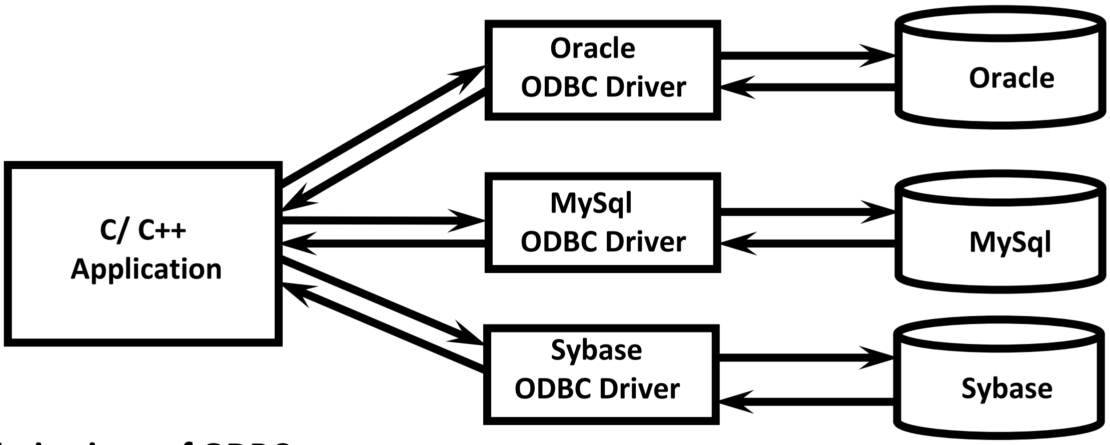


- The Application will become Database Dependent and creates Maintenance Problems.

- To overcome this Problem, Microsoft People introduced "ODBC" Concept in 1992. It is Database Independent API.

- With ODBC API, Application can communicate with any Database just by selecting corresponding ODBC Driver.

- We are not required to use any Database specific Libraries in our Application. Hence our Application will become Database Independent.



Limitations of ODBC:

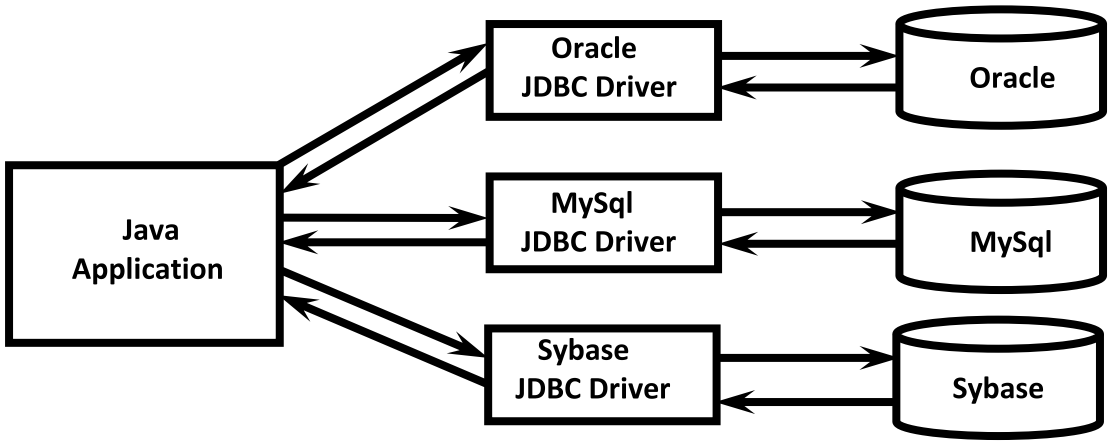
1) ODBC Concept will work only for Windows Machines. It is Platform Dependent Technology.
2) ODBC Drivers are implemented in C Language. If we use ODBC for Java Applications, then Performance will be down because of internal conversions from Java to C and C to Java.

- Because of above Reasons, ODBC Concept is not suitable for Java Applications.

- For Java Applications, SUN People introduced JDBC Concept.

- JDBC Concept Applicable for any Platform. It is Platform Independent Technology.

- JDBC Drivers are implemented in Java. If we use JDBC for Java Applications, then internal Conversions are not required and hence there is no Effect on Performance.



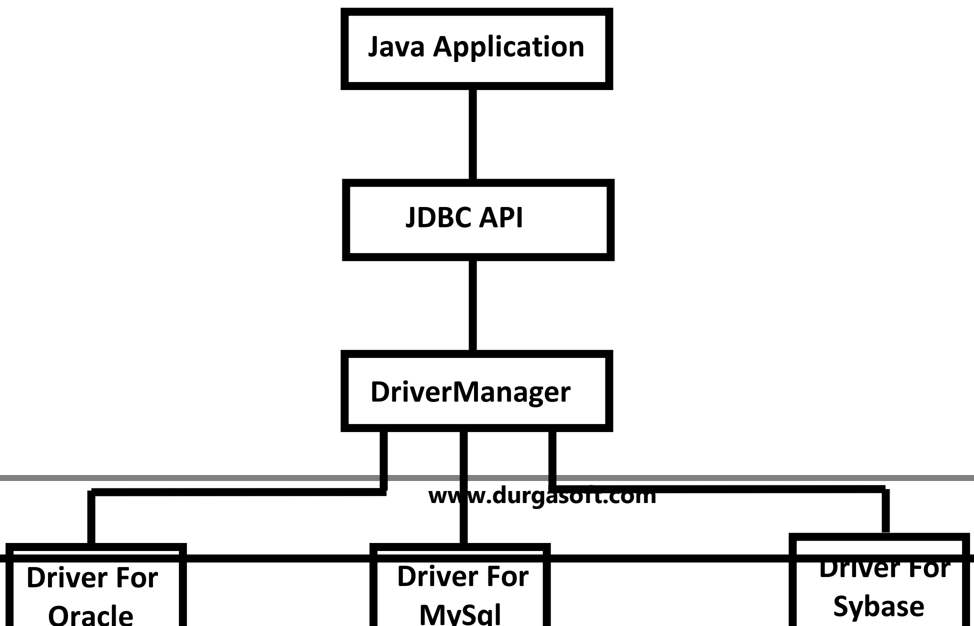
***<u>Note:</u>**

1) ODBC Concept is applicable for any Database and for any Language, but only for Windows Platform.
2) JDBC Concept is Applicable for any Platform and for any Database, but only for Java Language.

Differences Between JDBC and ODBC

ODBC	JDBC
1) ODBC Stands for Open Database Connectivity	1) JDBC Stands for Java Database Connectivity
2) Introduced by Microsoft.	2) Introduced by Sun Micro Systems.
3) We can Use ODBC for any Languages like C, C++, Java, Etc.	3) We can Use JDBC only for Java Language.
4) We can use ODBC only for Windows Platforms.	4) We can use JDBC for any Platform.
5) Mostly ODBC Drivers are developed in Native Languages like C OR C++.	5) Mostly JDBC Drivers are developed in Java.
6) For Java Applications, it is not recommended to use ODBC because Performance will be Down due to Internal Conversions and Application will become Platform Dependent.	6) For Java Applications, it is highly recommended to use JDBC because there is no Performance Problems and Platform Dependency Problems.

JDBC Architecture



- JDBC API provides DriverManager to our Java Application.

- Java Application can communicate with any Database with the help of DriverManager and Database Specific Driver.

DriverManager:

- It is the Key Component in JDBC Architecture.
- DriverManager is a Java Class present in *java.sql* Package.
- It is responsible to manage all Database Drivers available in our System.

- DriverManager is responsible to register and unregister Database Drivers.
  ```
  DriverManager.registerDriver(Driver);
  DriverManager.unregisterDriver(Driver);
  ```

- DriverManager is responsible to establish Connection to the Database with the help of Driver Software.
  ```
  Connection con = DriverManager.getConnection (jdbcurl, username, pwd);
  ```

Database Driver:

- It is the very Important Component of JDBC Architecture.
- Without Driver Software we cannot touch Database.
- It acts as Bridge between Java Application and Database.
- It is responsible to convert Java Calls into Database specific Calls and Database specific Calls into Java Calls.

Note:

1) Java Application is Database Independent but Driver Software is Database Dependent. Because of Driver Software only Java Application will become Database Independent.
2) Java Application is Platform Independent but JVM is Platform Dependent. Because of JVM only Java Application will become Platform Independent.

JDBC API

- JDBC API provides several Classes and Interfaces.
- Programmer can use these Classes and Interfaces to communicate with the Database.
- Driver Software Vendor can use JDBC API while developing Driver Software.
- JDBC API defines 2 Packages

1) java.sql Package:

It contains basic Classes and Interfaces which can be used for Database Communication.

Interfaces	Classes
1) Driver	1) DriverManager

2) Connection 3) Statement 4) PreparedStatement 5) CallableStatement 6) ResultSet 7) ResultSetMetaData 8) DataBaseMetaData	2) Date 3) Time 4) TimeStamp 5) Types

2) <u>javax.sql Package:</u>

It defines more advanced Classes and Interfaces which can be used for Database Communication.

There are multiple Sub Packages are also available

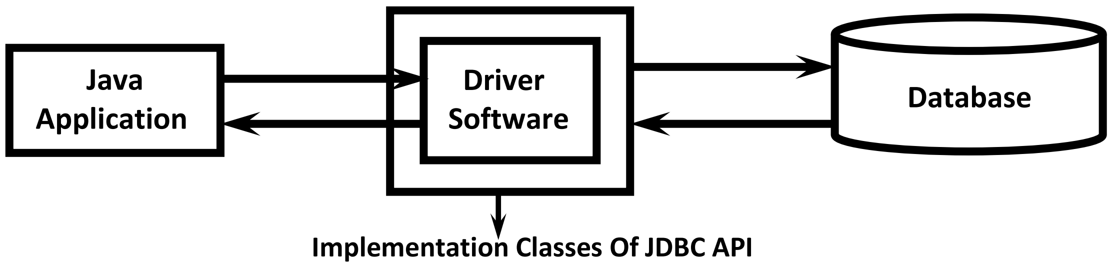
- **javax.sql.rowset;**
- **javax.sql.rowset.serial;**
- **javax.sql.rowset.spi;**

<u>Interfaces</u>	<u>Classes</u>
1) DataSource 2) RowSet 3) RowSetListener 4) ConnectionEventListener 5) StatementEventListener	1) ConnectionEvent 2) RowSetEvent 3) StatementEvent ::::::::::::::::::

- **Programmers are not responsible to provide Implementation for JDBC API Interfaces.**

- **Most of the times Database Vendor is responsible to provide Implementation as the Part of Driver Software.**

- **Every Driver Software is a Collection of Classes implementing various Interfaces of JDBC API, which can be used to communicate with a particular Database.**

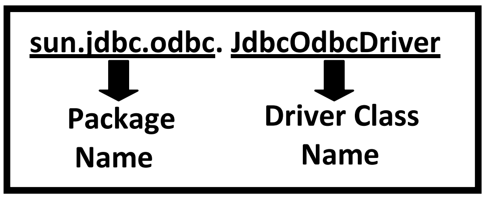


- **For Example, Driver Software of Oracle means Collection of Implementation Classes of JDBC API, which can be used to communicate with Oracle Database.**

- **Every Driver Software is identified with some Special Class which is nothing but Driver Class. It is the Implementation Class of Driver Interface present in. *java.sql* Package.**

- **As the Part of JDK, SUN People provided one Built-In Driver Software which implements JDBC API, which is nothing but Type-1 Driver (JDBC-ODBC Bridge Driver).**

- **The corresponding Driver Class Name is:**



Difference between Driver Interface, Driver Class and Driver Software:

1) Driver Interface:

This Interface present in *java.sql* Package.
This Interface acts as Requirement Specification to implement Driver Class.

2) Driver Class:

It is the Implementation Class of Driver Interface
Eg: sun.jdbc.odbc.jdbcodbcdriver

3) Driver Software:

- **It is the Collection of Implementation Classes of various Interfaces present in JDBC API.**
- **It acts as Bridge between Java Application and Database.**
- **It is responsible to convert Java Calls into Database specific Calls and Database specific Calls into Java Calls.**

- **Usually Driver Softwares are available in the Form of jar File.**
  Eg:
  - **ojdbc14.jar**
  - **ojdbc6.jar**
  - **ojdbc7.jar**
  - **mysql-connector.jar**

- **Driver Softwares can be provided by the following Vendors**
  - **Java Vendor (Until 1.7 Version Only)**
  - **Database Vendor**
  - **Third Party Vendor**

- Type-1 Driver (JDBC-ODBC Bridge Driver) provided by Java Vendor.
- Thin Driver provided by Oracle Database Vendor.
- Inet is a Third Party Vendor and providing several Driver Softwares for different Databases.

  Eg:
  - Inet Oraxo For Oracle Database
  - Inet Merlia For Microsoft SQL Server
  - Inet Sybelux For Sybase Database

Note: It is highly recommended to use Database Vendor provided Driver Softwares.

- While developing Driver Software, Vendors may use only Java OR Java with other Languages like C OR C++.

- If Driver Software is developed only in Java Language then such Type of Drivers are called Pure Java Drivers.

- If Driver Software developed with Java and other Languages, then such Type of Driver Softwares are called Partial Java Drivers.

Types of Drivers

While communicating with Database, we have to convert Java Calls into Database specific Calls and Database specific Calls into Java Calls. For this Driver Software is required. In the Market Thousands of Driver Softwares are available. But based on Functionality all Driver Software Drivers are divided into 4 Types.

1) Type-1 Driver (JDBC-ODBC Bridge Driver OR Bridge Driver)

2) Type-2 Driver (Native API-Partly Java Driver OR Native Driver)

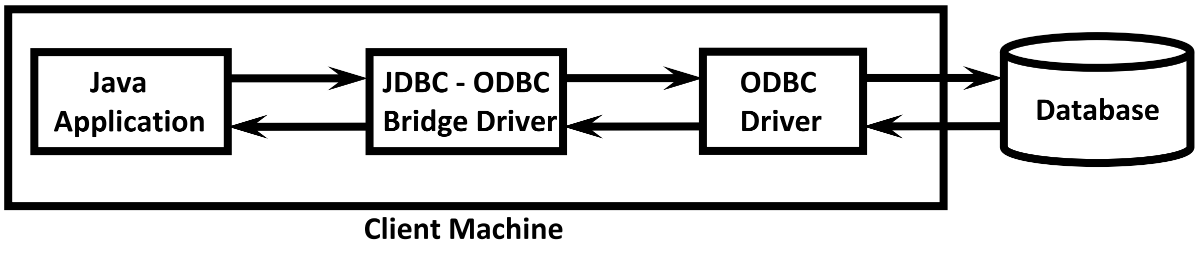
3) Type-3 Driver (All Java Net Protocol Driver OR Network Protocol Driver OR Middleware Driver)

4) Type-4 Driver (All Java Native Protocol Driver OR Pure Java Driver OR Thin Driver)

Note:
Progress Data direct Software Company introduced Type-5 Driver, but it is not Industry Recognized Driver.

Type-1 Driver:

Also known as *JDBC-ODBC Bridge Driver* OR *Bridge Driver.*



This Driver provided by Sun Micro Systems as the Part of JDK. But this Support is available until 1.7 Version only.

Internally this Driver will take Support of ODBC Driver to communicate with Database.

Type-1 Driver converts JDBC Calls (Java Calls) into ODBC Calls and ODBC Driver converts ODBC Calls into Database specific Calls.

Hence Type-1 Driver acts as Bridge between JDBC and ODBC.

Advantages :

1. It is very easy to use and maintain.

2. We are not required to install any separate Software because it is available as the Part of JDK.

3. Type-1 Driver won't communicates directly with the Database. Hence it is Database Independent Driver. Because of this migrating from one Database to another Database will become Easy.

Limitations:

1. It is the slowest Driver among all JDBC Drivers (Snail Driver), because first it will convert JDBC Calls into ODBC Calls and ODBC Driver converts ODBC Calls into Database specific Calls.

2. This Driver internally depends on ODBC Driver, which will work only on Windows Machines. Hence Type-1 Driver is Platform Dependent Driver.

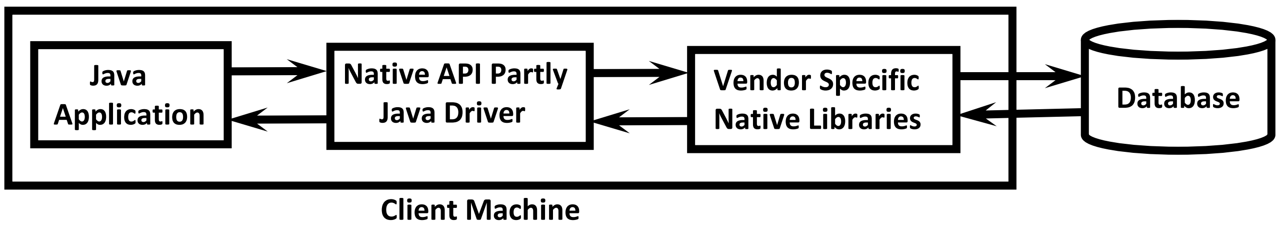
3. No Support from JDK 1.8 Version onwards.

Note:

Because of above Limitations it is never recommended to use Type-1 Driver.

Type-2 Driver:

It is also known as *Native API -Partly Java Driver* OR *Native Driver.*


Java Application
Native API Partly Java Driver
Vendor Specific Native Libraries
Database
Client Machine

Type-2 Driver is exactly same as Type-1 Driver except that ODBC Driver is replaced with Vendor specific Native Libraries.

Type-2 Driver internally uses Vendor specific Native Libraries to Communicate with Database.

Native Libraries means the Set of Functions written in Non-Java (Mostly C OR C++).

We have to install Vendor provided Native Libraries on the Client Machine.

Type-2 Driver converts JDBC Calls into Vendor specific Native Library Calls, which can be understandable directly by Database Engine.

Advantages:

1. When compared with Type-1 Driver Performance is High, because it required only one Level Conversion from JDBC to Native Library Calls.

2. No need of arranging ODBC Drivers.

3. When compared with Type-1 Driver, Portability is more because Type-1 Driver is applicable only for Windows Machines.

Limitations:

1. Internally this Driver using Database specific Native Libraries and hence it is Database Dependent Driver. Because of this migrating from one Database to another Database will become Difficult.

2. This Driver is Platform Dependent Driver.

3. On the Client Machine compulsory we should install Database specific Native Libraries.

4. There is no Guarantee for every Database Vendor will provide This Driver.
(Oracle People provided Type-2 Driver but MySql People won't provide this Driver)

Eg: OCI (Oracle Call Interface) Driver is Type-2 Driver provided by Oracle.
OCI Driver internally uses OCI Libraries to communicate with Database.

OCI Libraries contain "C Language Functions"

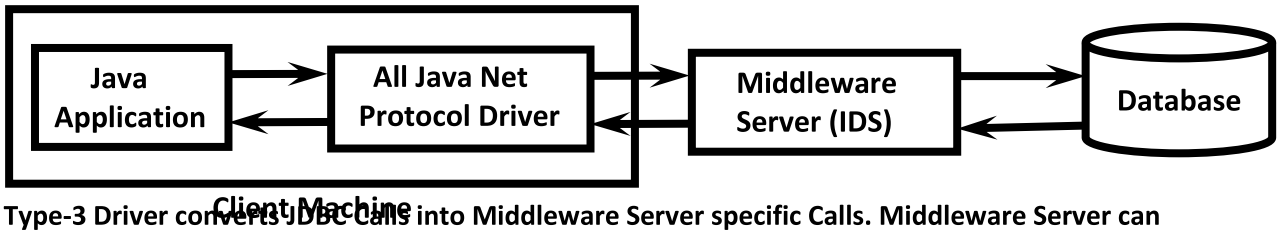
OCI Driver and corresponding OCI Libraries are available in the following Jar File. Hence we have to place this Jar File in the Class Path.

ojdbc14.jar ➔ Oracle 10g (Internally Uses Java 1.4V)
ojdbc6.jar ➔ Oracle 11g (Internally Uses Java 1.6V)
ojdbc7.jar ➔ Oracle 12c (Internally Uses Java 1.7V)

Note: The only Driver which is both Platform Dependent and Database Dependent is Type-2 Driver. Hence it is not recommended to use Type-2 Driver.

Type-3 Driver:

Also known as *All Java Net Protocol Driver* OR *Network Protocol Driver* OR *Middleware Driver*



Type-3 Driver converts JDBC Calls into Middleware Server specific Calls. Middleware Server can convert Middleware Server specific Calls into Database specific Calls.

Internally Middleware Server may use Type-1, 2 OR 4 Drivers to communicates with Database.

Advantages:

1. This Driver won't communicate with Database directly and hence it is Database Independent Driver.

2. This Driver is Platform Independent Driver.

3. No need of ODBC Driver OR Vendor specific Native Libraries

Limitations:

1. Because of having Middleware Server in the Middle, there may be a chance of Performance Problems.

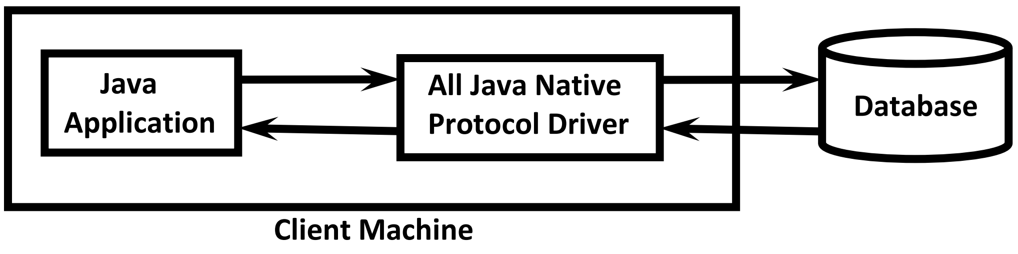
2. We need to purchase Middleware Server and hence the cost of this Driver is more when compared with remaining Drivers.

Eg: IDS Driver (Internet Database Access Server)

Note: The only Driver which is both Platform Independent and Database Independent is Type-3 Driver. Hence it is recommended to use.

Type-4 Driver:

Also known as *Pure Java Driver* OR *Thin Driver.*



This Driver is developed to talk with the Database directly without taking Support of *ODBC Driver* OR *Vendor Specific Native Libraries* OR *Middleware Server*.

This Driver uses Database specific Native Protocols to communicate with the Database.

This Driver converts JDBC Calls directly into Database specific Calls.

This Driver developed only in Java and hence it is also known as Pure Java Driver. Because of this, Type-4 Driver is Platform Independent Driver.
This Driver won't require any Native Libraries at Client side and hence it is light weighted. Because of this it is treated as Thin Driver.

Advantages:

1. It won't require any Native Libraries, *ODBC Driver* OR *Middleware Server*

2. It is Platform Independent Driver

3. It uses Database Vendor specific Native Protocol and hence Security is more.

Limitation:

The only Limitation of this Driver is, it is Database Dependent Driver because it is communicating with the Database directly.

Eg: Thin Driver for Oracle
Connector/J Driver for MySQL

Note: It is highly recommended to use Type-4 Driver.

Java Application ➔ Type-1 Driver ➔ ODBC Driver ➔ DB
Java Application ➔ Type-2 Driver ➔ Vendor Specific Native Libraries ➔ DB
Java Application ➔ Type-3 Driver ➔ Middleware Server ➔ DB
Java Application ➔ Type-4 Driver ➔ DB

Which Driver should be used?

1. If we are using only one Type of Database in our Application then it is recommended to use Type-4 Driver.

<u>Eg:</u> Stand Alone Applications, Small Scale Web Applications

2. If we are using multiple Databases in our Application then Type-3 Driver is recommended to use.

<u>Eg:</u> Large Scale Web Applications and Enterprise Applications

3. If Type-3 and Type-4 Drivers are not available then only we should go for Type-2 Driver.

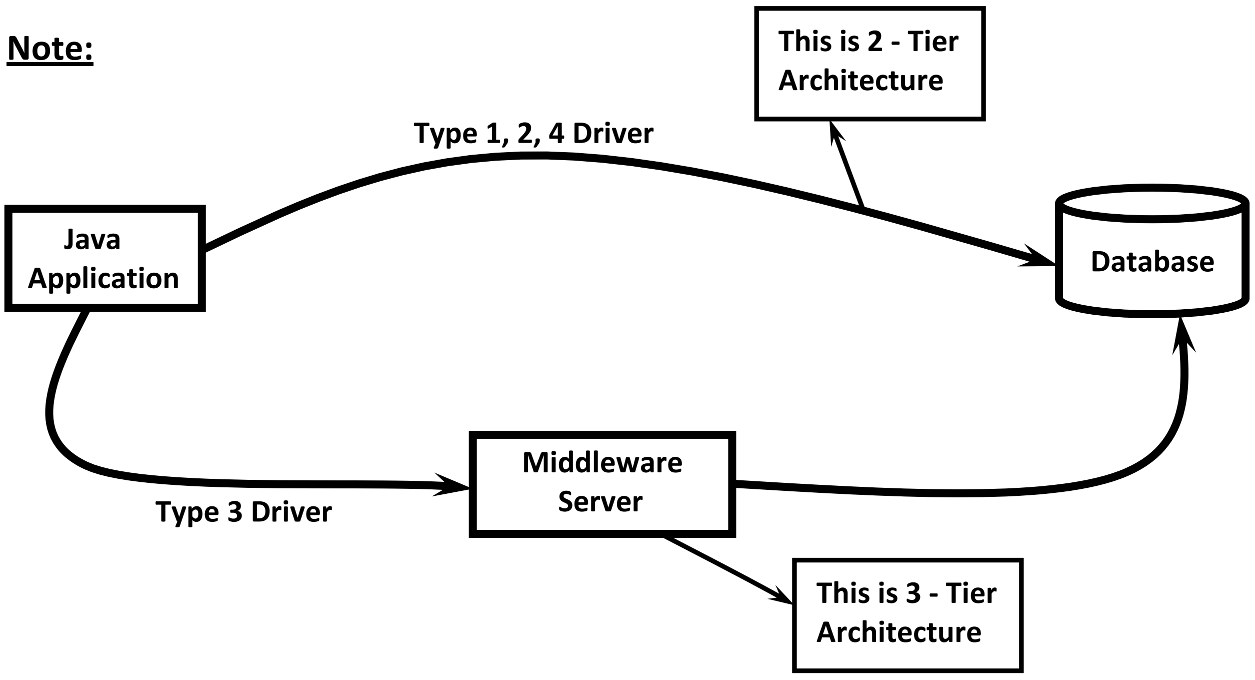
4. If no other Driver is available then only we should go for Type-1 Driver.

Differences between *Thin* and *Thick* Driver:

If Driver won't require any extra Component to communicate with Database, such type of Driver is called Thin Driver.
<u>Eg:</u> Type-4 Driver

If Driver require some extra Component (like *ODBC Driver* OR *Vendor specific Native Libraries* OR *Middleware Server*), such Type of Driver is called Thick Driver.
<u>Eg:</u> Type-1, Type-2 and Type-3 Drivers

Note:



Type-1, Type-2 and Type-4 Drivers follow 2-Tier Architecture.
Type-3 Driver follows 3-Tier Architecture.

Comparison Table of All JDBC Drivers

Property	Type - 1	Type - 2	Type - 3	Type - 4
1) Conversion	From JDBC Calls To ODBC Calls	From JDBC Calls To Native Library Calls	From JDBC Calls To Middleware Server Specific Calls	From JDBC Calls To Database Specific Calls
2) Implemented In	Only In Java	Java + Native Language	Only In Java	Only In Java
3) Architecture	2 - Tier	2 - Tier	3 - Tier	2 - Tier
4) Is It Platform Independent?	No	No	Yes	Yes
5) Is It Database Independent?	Yes	No	Yes	No
6) Is It Thin OR Thick?	Thick	Thick	Thick	Thin

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.

```
class JdbOdbcDriver
{
   static
   {
      JdbOdbcDriver driver= new JdbOdbcDriver();
      DriverManager.registerDriver(driver);
   }
}
```

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

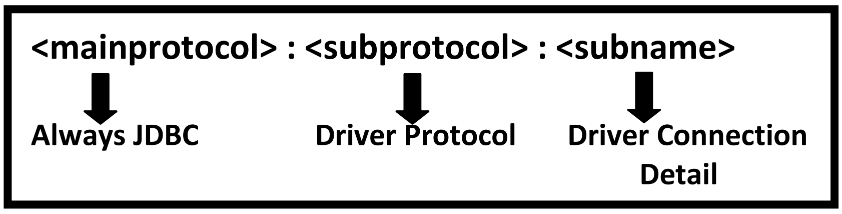
Once we loaded and registered Driver, by using that we can establish Connection to the Database. For this *DriverManager* Class contains *getConnection()* Method.

```
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`

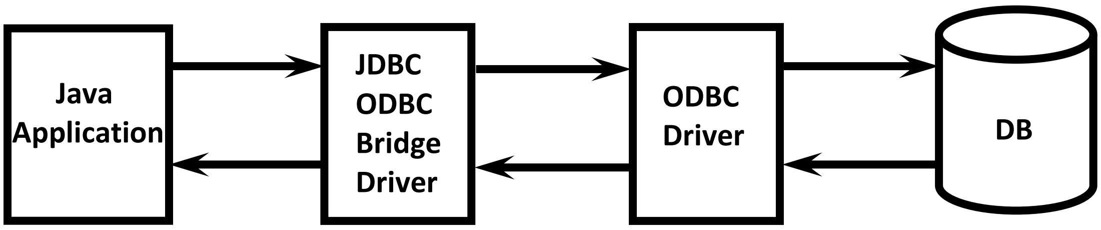
Eg: Connection con= DriverManager.getConnection("jdbc:odbc:demodsn","scott","tiger");

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:

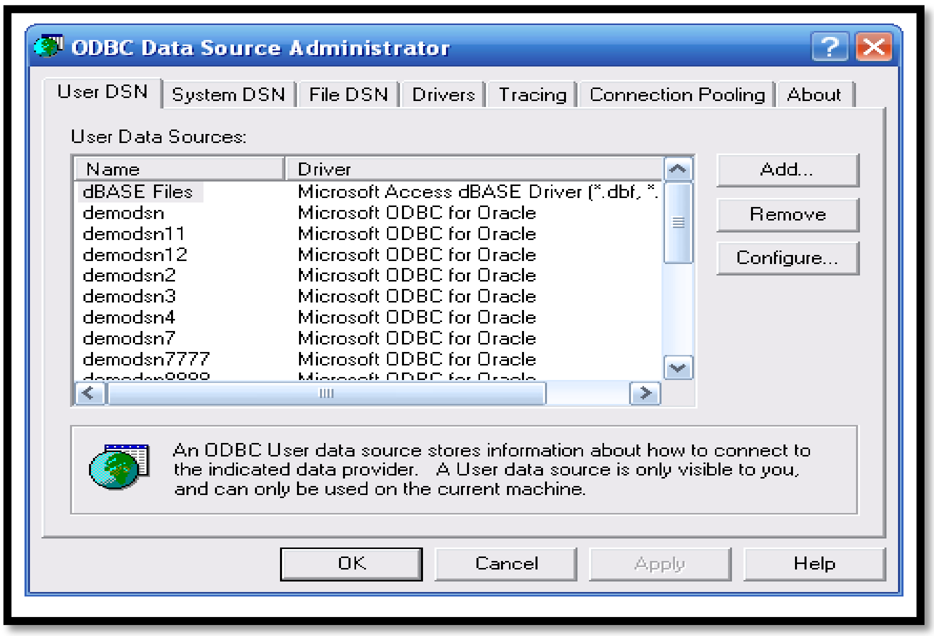
It is the sharable DSN and it is available for all Users who can access that System.
It is also known as Global DSN.

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

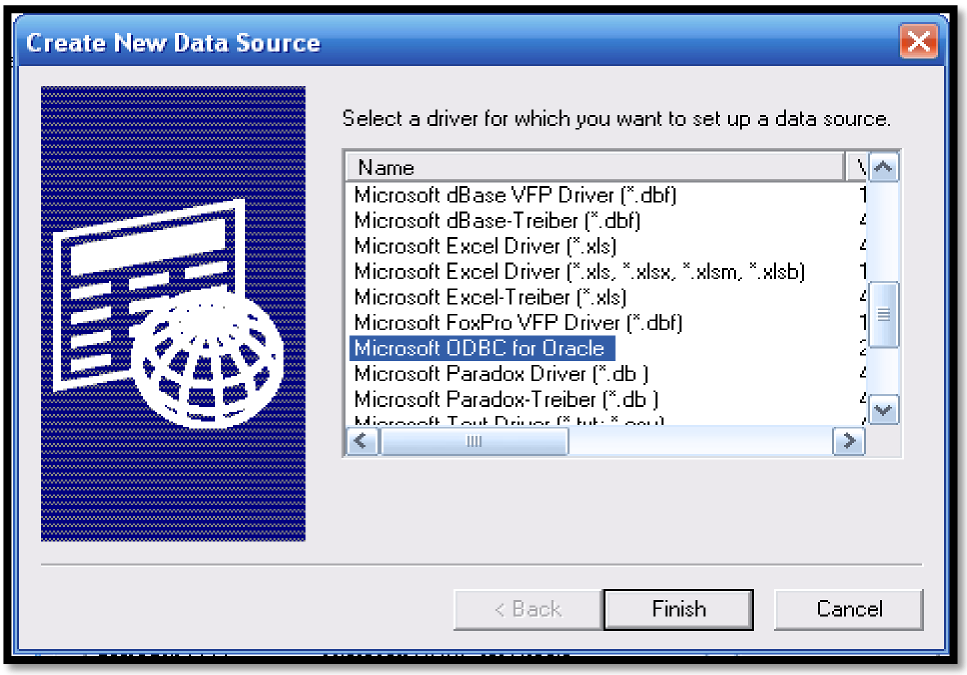

ODBC Data Source Administrator

User DSN | System DSN | File DSN | Drivers | Tracing | Connection Pooling | About

User Data Sources:

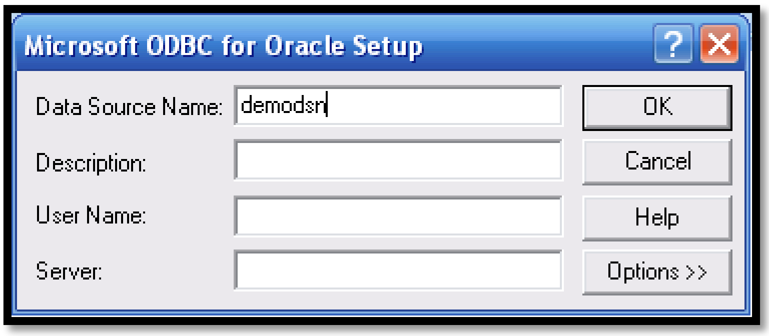
Name	Driver
dBASE Files	Microsoft Access dBASE Driver (*.dbf, *.
demodsn	Microsoft ODBC for Oracle
demodsn11	Microsoft ODBC for Oracle
demodsn12	Microsoft ODBC for Oracle
demodsn2	Microsoft ODBC for Oracle
demodsn3	Microsoft ODBC for Oracle
demodsn4	Microsoft ODBC for Oracle
demodsn7	Microsoft ODBC for Oracle
demodsn7777	Microsoft ODBC for Oracle

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.


Create New Data Source

Select a driver for which you want to set up a data source.

Name
Microsoft dBase VFP Driver (*.dbf)
Microsoft dBase-Treiber (*.dbf)
Microsoft Excel Driver (*.xls)
Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xlsb)
Microsoft Excel-Treiber (*.xls)
Microsoft FoxPro VFP Driver (*.dbf)
Microsoft ODBC for Oracle
Microsoft Paradox Driver (*.db)
Microsoft Paradox-Treiber (*.db)



Data Source Name: demodsn
Description:
User Name:
Server:

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?

```java
import java.sql.*;
public class DbConnectDemo1
{
   public static void main(String[] args) throws Exception
   {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con=DriverManager.getConnection("jdbc:odbc:demodsn7","scott","tiger"
);
      if(con != null)
      {
         System.out.println("Connection established Successfully");
      }
      else
      {
         System.out.println("Connection not established");
      }
   }
}
```

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:

```
boolean b = st.execute("dynamically provided query");

if(b==true)//select query
{
  ResultSet rs=st.getResultSet();
  //use rs to get data
}
else// non-select query
{
  int rowCount=st.getUpdateCount();
  SOP("The number of rows effected:"+rowCount);
}
```

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)");
```

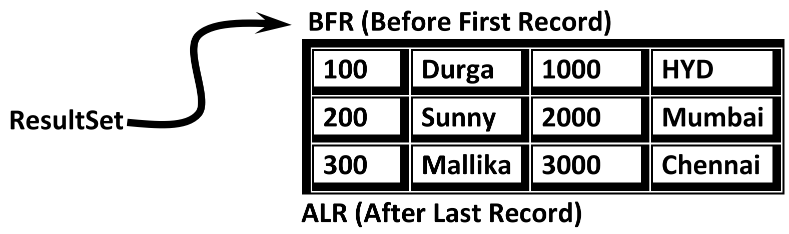
For Type-1 Driver, we will get -1 and For Type-4 Driver, we will get 0

```
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.



100	Durga	1000	HYD
200	Sunny	2000	Mumbai
300	Mallika	3000	Chennai

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.

```
while(rs.next())
{
   read data from that record
}
```

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.

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

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();*

1.7 Version: try With Resources

Usually we will close the Resources inside finally Block.

```
try
{
   Open Database Connection
}
catch(X e)
{
}
finally
{
   Close That Database Connection
}
```

But in Java 1.7 Version try with Resources Concept introduced.

The Advantage of this Concept is, whatever Resources we opened as the Part of *try* Block, will be closed automatically once Control reaches End of *try* Block either Normally OR Abnormally. We are not required to close explicitly.

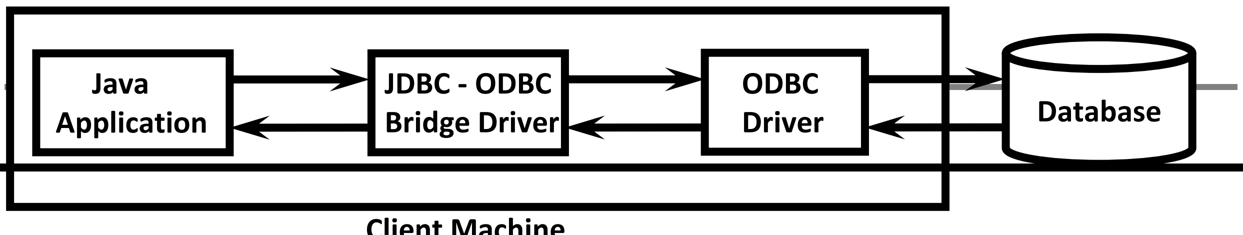
Hence until 1.6 Version *finally* Block is just like Hero but from 1.7 Version onwards *finally* Block became Zero.

```
try (Resource)
{
}
```

<u>Eg:</u>

```
try (Connection con = DM.getConnection(-,-,-))
{
	Use con based on our Requirement
	Once Control reaches End of try Block, automatically con will be closed, we are not
	required to close explicitly
}
```

Working With Type-1 Driver:



Also known as JDBC ODBC Bridge Driver.

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

Driver Class Name: sun.jdbc.odbc.JdbcOdbcDriver
JDBC URL: jdbc:odbc:demodsn
username: scott
pwd: tiger
query: select * from movies;

<u>Note:</u> We should Create Movies Table in the Database and Insert some Sample Data...

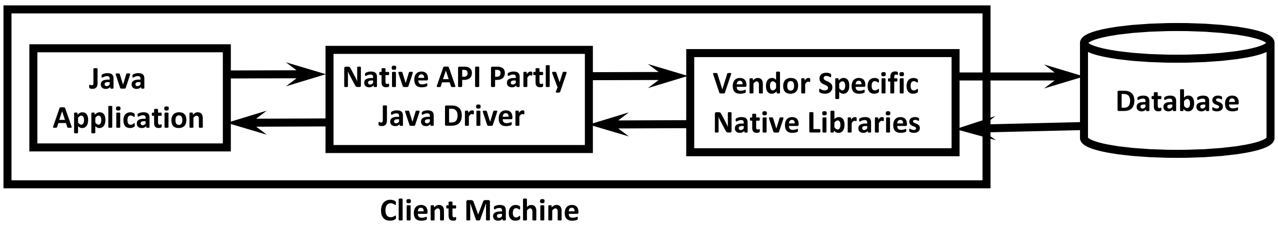
```
create table movies(no number, name varchar2(20),hero varchar2(20),heroine varchar2(20));

insert into movies values(1,'Bahubali','Prabhas','Anushka');
insert into movies values(2,'Raees','Sharukh','Sunny');
insert into movies values(3,'Winner','Sai','Rakul');
```

<u>Eg:</u>

```
import java.sql.*;
public class  Type1DriverDemo
{
   public static void main(String[] args) throws Exception
   {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
     Connection con= DriverManager.getConnection("jdbc:odbc:demodsn","scott","tiger");
      Statement st = con.createStatement();
      ResultSet rs = st.executeQuery("select * from movies");
      while(rs.next())
      {
         System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getString(3)+"\t"+rs.get
String(4));
      }
      con.close();
   }
}
```

<u>Note:</u>
Type-1 Driver is available until 1.7 Version only.
From 1.8 Version onwards, the above Program won't work.

Working with Type-2 Driver:


Java Application
Native API Partly Java Driver
Vendor Specific Native Libraries
Database
Client Machine

Oracle People provided Type-2 Driver is OCI (Oracle Call Interface) Driver.
Internally OCI Driver uses OCI Native Libraries.

OCI Driver and corresponding Native Libraries are available in the following Jar File.

```
ojdbc14.jar ➔  Oracle 10g (Internally Oracle Uses Java1.4V)
ojdbc6.jar  ➔  Oracle 11g (Internally Oracle Uses Java 6V)
ojdbc7.jar  ➔  Oracle 12c (Internally Oracle Uses Java 7V)
```

To make Driver Software available to our Program we have to place *ojdbc6.jar* in Class Path.

We have to collect Jar File from the following Location of Oracle Installation.

```
C:\oraclexe\app\oracle\product\11.2.0\server\jdbc\lib\ojdbc6.jar

D:\Tomcat 7.0\lib\servlet-api.jar;

D:\Tomcat 7.0\lib\jsp-api.jar;

D:\mysql-connector-java-bin.jar;.;
```

Driver Class Name: oracle.jdbc.driver.OracleDriver
oracle.jdbc.OracleDriver

jdbc url: jdbc:oracle:oci8:@XE (until oracle 8V)
jdbc:oracle:oci:@XE (From Oracle 9 onwards)
where XE is SID(System ID)

Every Database has a Unique System ID. We can find SID of our Database in the following 2 ways.

1st way:

We have to execute the following Command from SQL Plus Command Prompt

```
SQL> select * from global_name;
```

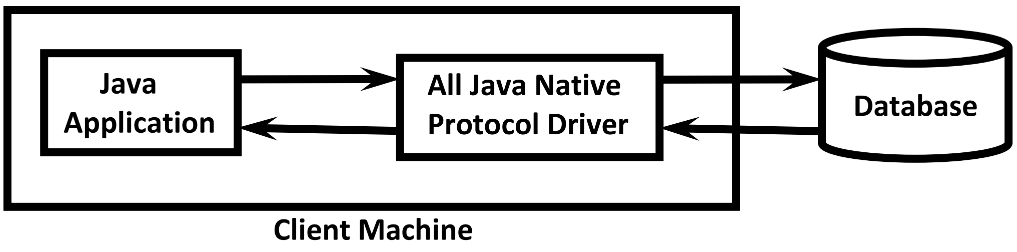
2nd way:

We can find SID from the following File

```
C:\oraclexe\app\oracle\product\11.2.0\server\network\ADMIN\tnsnames.ora
```

```
import java.sql.*;
public class  Type2DriverDemo
{
   public static void main(String[] args) throws Exception
   {
      Class.forName("oracle.jdbc.OracleDriver");
     Connection con= DriverManager.getConnection("jdbc:oracle:oci:@XE","scott","tiger");
      Statement st = con.createStatement();
      ResultSet rs = st.executeQuery("select * from movies");
      while(rs.next())
      {
         System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getString(3)+"\t"+rs.get
String(4));
      }
      con.close();
   }
}
```

Working With Type-4 Driver:



Also known as *Pure Java Driver* OR *Thin Driver.*

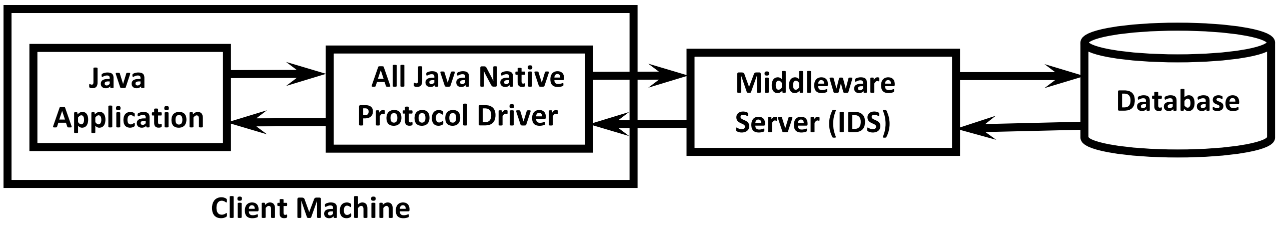
Type-2 and Type-4 Drivers of Oracle having same Jar File, same Driver Class Name, but different JDBC URL's.

Driver Class Name: oracle.jdbc.driver.OracleDriver
oracle.jdbc.OracleDriver

JDBC URL: jdbc:oracle:thin:@localhost:1521:XE

```
import java.sql.*;
public class  Type4DriverDemo
{
   public static void main(String[] args) throws Exception
   {
      Class.forName("oracle.jdbc.OracleDriver");
      Connection con= DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
","scott","tiger");
      Statement st = con.createStatement();
      ResultSet rs = st.executeQuery("select * from movies");
      while(rs.next())
      {
            System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getString(3)+"\t"+rs.getString(4));
        }
        con.close();
    }
}
```

Working with Type-3 Driver:



An extra activity in Type-3 Driver is we have to install Middleware Server.
Eg: IDS Server (Internet Database Access Server)

How to install IDS Server?

idssoftware.com ➔ Download ➔ IDS Server Trial ➔ IDS Server 4.2.2 Lite Evaluation ➔ Windows (2008/2003/XP/2000/NT)

Download and Install IDS Server.

We have to set Driver Software in the Class Path. For this the following Jar File should be placed in the Class Path.

```
C:\IDSServer\classes\jdk13drv.jar
```

Driver Class Name: ids.sql.IDSDriver

jdbc url: jdbc:ids://localhost:12/conn?dsn=mysysdsn

Internally IDS Server will use Type-1 Driver to communicate with Database. For this we have to configure "System DSN" and we have to choose "Oracle In XE".

```
import java.sql.*;
public class  Type3DriverDemo
{
    public static void main(String[] args) throws Exception
    {
        Class.forName("ids.sql.IDSDriver");
        Connection con= DriverManager.getConnection("jdbc:ids://localhost:12/conn?dsn=mysysdsn","scott","tiger");
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery("select * from movies");
        while(rs.next())
        {
           System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getString(3)+"\t"+rs.get
String(4));
        }
        con.close();
    }
}
```

Working With Type-5 Driver:

This Driver introduced by "Progress Data Direct" Software Company.
This Driver is the enhanced Version of Type-4 Driver.
This Driver is not Industry recognized Driver.

We have to Download Driver Software from Progress Data Direct Web Site as follows...

https://www.progress.com/jdbc ➔ Available JDBC Data Sources ➔ Relational and Analytics ➔ Oracle Database ➔ Download JDBC connectors ➔ Windows ➔ Fill Form and Download

We will get Setup File and execute so that Driver Software available in our System.

Type-5 Driver Software available in *oracle.jar* which is available in the following Location.

```
C:\Program Files\Progress\DataDirect\Connect_for_JDBC_51\lib\oracle.jar
```

We have to Place this Jar File in the Class Path

Driver Class Name: `com.ddtek.jdbc.oracle.OracleDriver`

jdbc_url: `jdbc:datadirect:oracle://localhost:1521;ServiceName=XE`

```
import java.sql.*;
public class  Type5DriverDemo
{
    public static void main(String[] args) throws Exception
    {
        Class.forName("com.ddtek.jdbc.oracle.OracleDriver");
        Connection con= DriverManager.getConnection("datadirect:oracle://localhost:1521;S
erviceName=XE","scott","tiger");
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery("select * from movies");
        while(rs.next())
        {
           System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getString(3)+"\t"+rs.get
String(4));
        }
        con.close();
    }
}
```

Summary of All JDBC 5 Drivers

Driver Type	Required Jar File	Driver Class Name	JDBC URL
Type - 1	No jar File required	sun.jdbc.odbc.JdbcOdbcDriver	jdbc:odbc:demodsn
Type - 2	ojdbc14.jar ojdbc6.jar ojdbc7.jar	oracle.jdbc.driver.OracleDriver oracle.jdbc.OracleDriver	jdbc:oracle:oci:@XE
Type - 3	jdk13drv.jar	ids.sql.IDSDriver	jdbc:ids://localhost: 12/conn?dsn=demo systemdsn3
Type - 4	ojdbc14.jar ojdbc6.jar ojdbc7.jar	oracle.jdbc.driver.OracleDriver oracle.jdbc.OracleDriver	jdbc:oracle:thin: @localhost:1521:XE
Type - 5	oracle.jar	com.ddtek.jdbc.oracle.OracleDriver	jdbc:datadirect: oracle://localhost: 1521;ServiceName= XE

How To Read Dynamic Input From Key Board?

Scanner is specially designed Class to read Dynamic Input from the Keyboard.
Scanner Class introduced in Java 1.5V.
Scanner Class present in *java.util* Package.

```
import java.util.*;
class Test
{
  public static void main(String[] args)
  {
    Scanner sc = new Scanner(System.in);
```

```java
        System.out.println("Enter Employee Number:");
        int eno=sc.nextInt();
        System.out.println("Enter Employee Name:");
        String ename=sc.next();
        System.out.println("Enter Employee Salary:");
        double esal=sc.nextDouble();
        System.out.println("Enter Employee Address:");
        String eaddr=sc.next();
        System.out.println(eno+"\t"+ename+"\t"+esal+"\t"+eaddr);
    }
}
```

Application-1: How to Create a Table

```java
import java.sql.*;
public class CreateTableDemo
{
    public static void main(String[] args) throws Exception
    {
        String driver="oracle.jdbc.OracleDriver";
        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
        String user="scott";
        String pwd="tiger";
        String sql_query="create table employees(eno number,ename varchar2(10),esal numb
er,eaddr varchar2(10))";
        Class.forName(driver);
        Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
        Statement st = con.createStatement();
        st.executeUpdate(sql_query);
        System.out.println("Table Created Successfully");
        con.close();
    }
}
```

Application-2: How To Delete A Table

```java
import java.sql.*;
public class DropTableDemo
{
    public static void main(String[] args) throws Exception
    {
        String driver="oracle.jdbc.OracleDriver";
        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
        String user="scott";
        String pwd="tiger";
        String sql_query="drop table students";
        Class.forName(driver);
        Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
        Statement st = con.createStatement();
        st.executeUpdate(sql_query);
        System.out.println("Table Deleted Successfully");
        con.close();
    }
}
```

Formatting SQL Queries With Dynamic Input

```
String sqlQuery="insert into employees values(100,'durga',1000,'Hyd')";
```

If Data is available in the following Variables eno, ename, esal, eaddr

```
String sqlQuery="insert into employees values("+eno+",'"+ename+"',"+esal+",'"+eaddr+"')";
```

It is highly recommended to use String Class *format()* Method while writing SQL Queries with Dynamic Input.

```
String sqlQuery = String.format("insert into employees values (%d,'%s',%f,'%s')",
eno,ename,esal,eaddr);
```

Database Operations: Insert Operation

Use Cases Of Insert Operation:

1. Adding new Train Information in the IRCTC Database.
2. Adding new Movie Information in BookMyShow Database.
3. Adding new Book Information in Amazon Database.
4. Adding a new Customer.

Application-3: How to Insert a Record into Table

```
import java.sql.*;
public class InsertSingleRowDemo
{
   public static void main(String[] args) throws Exception
   {
      String driver="oracle.jdbc.OracleDriver";
      String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
      String user="scott";
      String pwd="tiger";
      String sql_query="insert into employees values(100,'durga',1000,'hyd')";
      Class.forName(driver);
      Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
      Statement st = con.createStatement();
      int updateCount=st.executeUpdate(sql_query);
      System.out.println("The number of rows inserted :"+updateCount);
      con.close();
   }
}
```

Note:

From SQL Plus Command Prompt, if we are performing any Database Operations then compulsory we should perform Commit Operation explicitly because Auto Commit Mode is not enabled.

From JDBC Application if we perform any Database Operations then the Results will be committed automatically and we are not required to Commit explicitly, because in JDBC Auto Commit is enabled by default.

Application-4: How to Insert Multiple Records into Table

```
import java.sql.*;
import java.util.*;
public class InsertMultipleRowsDemo
{
   public static void main(String[] args) throws Exception
   {
      String driver="oracle.jdbc.OracleDriver";
      String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
      String user="scott";
      String pwd="tiger";
      Class.forName(driver);
      Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
      Statement st = con.createStatement();
      Scanner sc = new Scanner(System.in);
      while(true)
      {
         System.out.println("Employee Number:");
         int eno=sc.nextInt();
         System.out.println("Employee Name:");
         String ename=sc.next();
         System.out.println("Employee Sal:");
         double esal=sc.nextDouble();
```

```
            System.out.println("Employee Address:");
            String eaddr=sc.next();
            String sqlQuery=String.format("insert into employees values(%d,'%s',%f,'%s')",eno,e
name,esal,eaddr);
            st.executeUpdate(sqlQuery);
            System.out.println("Record Inserted Successfully");
            System.out.println("Do U want to Insert one more record[Yes/No]:");
            String option = sc.next();
            if(option.equalsIgnoreCase("No"))
            {
               break;
            }
         }
         con.close();
      }
}
```

Database Operations: Update Operation

Use Cases of Update Operation:

1. Update Train Information According To New Schedule
2. Update/Change Price Of Book In Amazon Database.
3. Update Bonus For All Employees Whose Salary Less Than 5000

Application-5: How to Update a Record in the Table

```
import java.sql.*;
public class UpdateSingleRowDemo
{
   public static void main(String[] args) throws Exception
   {
      String driver="oracle.jdbc.OracleDriver";
      String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
      String user="scott";
      String pwd="tiger";
      String sql_query="update employees set esal=10000 where ename='durga'";
      Class.forName(driver);
      Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
      Statement st = con.createStatement();
      int updateCount=st.executeUpdate(sql_query);
      System.out.println("The number of rows updated :"+updateCount);
      con.close();
   }
}
```

Application-6: How to Update Multiple Records in the Table

```
import java.sql.*;
import java.util.*;
public class UpdateMultipleRowsDemo
{
   public static void main(String[] args) throws Exception
   {
      String driver="oracle.jdbc.OracleDriver";
      String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
      String user="scott";
      String pwd="tiger";
      Class.forName(driver);
      Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
      Statement st = con.createStatement();
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter Bonus Amount:");
      double bonus =sc.nextDouble();
      System.out.println("Enter Salary Range:");
      double salRange =sc.nextDouble();
      String sqlQuery=String.format("update employees set esal=esal+%f where esal<%f",b
onus,salRange);
      int updateCount=st.executeUpdate(sqlQuery);
      System.out.println("The number of rows updated :"+updateCount);
      con.close();
   }
}
```

Database Operations: Delete Operation

Use Cases of Delete Operation:

1. Terminate all Employees whose Salary greater than 7 Lakhs.
2. Delete outdated Book Information from Amazon Database.
3. Delete Old Movie Information from BookMyShow Database.

Application-7: How to Delete a Record from the Table

```
import java.sql.*;
public class DeleteSingleRowDemo
{
   public static void main(String[] args) throws Exception
   {
      String driver="oracle.jdbc.OracleDriver";
      String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
```

```
        String user="scott";
        String pwd="tiger";
        String sqlQuery="delete from employees where ename='durga'";
        Class.forName(driver);
        Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
        Statement st = con.createStatement();
        int updateCount=st.executeUpdate(sqlQuery);
        System.out.println("The number of rows deleted :"+updateCount);
        con.close();
    }
}
```

Application-8: How to Delete multiple Records from the Table

```
import java.sql.*;
import java.util.*;
public class DeleteMultipleRowsDemo
{
    public static void main(String[] args) throws Exception
    {
        String driver="oracle.jdbc.OracleDriver";
        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
        String user="scott";
        String pwd="tiger";
        Class.forName(driver);
        Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
        Statement st = con.createStatement();
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter CutOff Salary:");
        double cutOff =sc.nextDouble();
        String sqlQuery=String.format("delete from employees where esal>=%f",cutOff);
        int updateCount=st.executeUpdate(sqlQuery);
        System.out.println("The number of rows deleted :"+updateCount);
        con.close();
    }
}
```

Database Operations: Select Operation

Use Cases Of Select Operation:

1. Display all Trains Information from HYD to Mumbai
2. Display all Book Names written by Greene
3. Display all Movies Names in HYD City

Application-9: How to Select all Rows from the Table

```
import java.sql.*;
public class SelectAllRowsDemo
{
   public static void main(String[] args) throws Exception
   {
      String driver="oracle.jdbc.OracleDriver";
      String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
      String user="scott";
      String pwd="tiger";
      Class.forName(driver);
      Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
      Statement st = con.createStatement();
      String sqlQuery="select * from employees";
      boolean flag= false;
      ResultSet rs =st.executeQuery(sqlQuery);
      System.out.println("ENO\tENAME\tESALARY\tEADDR");
      System.out.println("---------------------------------------");
      while(rs.next())
      {
         flag=true;
         //System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename")+"\t"+rs.getDoubl
e("esal")+"\t"+rs.getString("eaddr"));
         System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getDouble(3)+"\t"+rs.ge
tString(4));
      }
      if(flag==false)
      {
         System.out.println("No Records found");
      }
      con.close();
   }
}
```

<u>Application-10:</u> How to Select all Rows from the Table based on sorting Order of the Salaries

```
import java.sql.*;
public class SelectAllRowsSortingDemo
{
   public static void main(String[] args) throws Exception
   {
      String driver="oracle.jdbc.OracleDriver";
      String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
      String user="scott";
      String pwd="tiger";
      Class.forName(driver);
      Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
      Statement st = con.createStatement();
      String sqlQuery="select * from employees order by esal DESC";
```

```
        boolean flag= false;
        ResultSet rs =st.executeQuery(sqlQuery);
        System.out.println("ENO\tENAME\tESALARY\tEADDR");
        System.out.println("--------------------------------------");
        while(rs.next())
        {
           flag=true;
           //System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename")+"\t"+rs.getDouble("esal")+"\t"+rs.getString("eaddr"));
           System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getDouble(3)+"\t"+rs.getString(4));
        }
        if(flag==false)
        {
           System.out.println("No Records found");
        }
        con.close();
    }
}
```

Note: For Ascending Order Query Is : select * from employees order by esal ASC;

Selecting particular Columns from the Database:

While retrieving Data from Database, we have to consider Order of Columns in the ResultSet but not in the Database Table.

Database Table Columns Order and ResultSet Columns Order need not be same. We have to give Importance only for ResultSet Columns Order.

Eg-1: select * from employees;

In this case Database Table contains 4 Columns and ResultSet also contains 4 Columns and Order is also same.

DB:(eno,ename,esal,eaddr)
RS:(eno,ename,esal,eaddr)

```
while(rs.next())
{
  SOP(rs.getInt(1)+"..."+rs.getString(2)+"..."+rs.getDouble(3)+".."+rs.getString(4));
}
```

Eg-2: select esal, eno, eaddr, ename from employees;

In this case Database Table contains 4 Columns and ResultSet also contains 4 Columns, but Order is not same.

DB:(eno,ename,esal,eaddr)
RS:(esal,eno,eaddr,ename)

We have to write the Code w.r.t ResultSet.

```
while(rs.next())
{
  SOP(rs.getDouble(1)+"..."+rs.getInt(2)+"..."+rs.getString(3)+".."+rs.getString(4));
}
```

Eg-3: select ename,eaddr from employees;

In this case Database Table contains 4 Columns, but ResultSet contains 2 Columns.

DB:(eno,ename,esal,eaddr)
RS:(ename,eaddr)

```
while(rs.next())
{
  SOP(rs.getString(1)+"..."+rs.getString(2));
}
```

Application-11: How to Select particular Columns from the Table

```
import java.sql.*;
public class SelectParticularColumnsDemo
{
   public static void main(String[] args) throws Exception
   {
      String driver="oracle.jdbc.OracleDriver";
      String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
      String user="scott";
      String pwd="tiger";
      Class.forName(driver);
      Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
      Statement st = con.createStatement();
      String sqlQuery="select ename,eaddr from employees";
      boolean flag=false;
      ResultSet rs =st.executeQuery(sqlQuery);
      System.out.println("ENAME\tEADDR");
      System.out.println("------------------");
      while(rs.next())
      {
         flag=true;
         System.out.println(rs.getString("ename")+"\t"+rs.getString("eaddr"));
         //System.out.println(rs.getString(1)+"\t"+rs.getString(2));
```

```
        }
        if(flag==false)
        {
           System.out.println("No Records found");
        }
        con.close();
    }
}
```

Application-12: How to Select Range of Records based on Address

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

public class SelectRangeOfRecordsDemo1
{
    public static void main(String[] args) throws Exception
    {
        String driver="oracle.jdbc.OracleDriver";
        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
        String user="scott";
        String pwd="tiger";
        Class.forName(driver);
        Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
        Statement st = con.createStatement();
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter City Name:");
        String addr=sc.next();
        String sqlQuery=String.format("select * from employees where eaddr='%s'",addr);
        boolean flag=false;
        ResultSet rs =st.executeQuery(sqlQuery);
        System.out.println("ENO\tENAME\tESALARY\tEADDR");
        System.out.println("--------------------------------------");
        while(rs.next())
        {
           flag=true;
           //System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename")+"\t"+rs.getDouble("esal")+"\t"+rs.getString("eaddr"));
           System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getDouble(3)+"\t"+rs.getString(4));
        }
        if(flag==false)
        {
           System.out.println("No Records found");
        }
        con.close();
    }
}
```

Application-13: How to Select Range of Records based on Salaries

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

public class SelectRangeOfRecordsDemo2
{
   public static void main(String[] args) throws Exception
   {
      String driver="oracle.jdbc.OracleDriver";
      String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
      String user="scott";
      String pwd="tiger";
      Class.forName(driver);
      Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
      Statement st = con.createStatement();
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter Begin Salary Range:");
      double beginSal=sc.nextDouble();
      System.out.println("Enter End Salary Range:");
      double endSal=sc.nextDouble();
      String sqlQuery=String.format("select * from employees where esal>%f and esal<%f",
beginSal,endSal);
      boolean flag=false;
      ResultSet rs =st.executeQuery(sqlQuery);
      System.out.println("ENO\tENAME\tESALARY\tEADDR");
      System.out.println("---------------------------------------");
      while(rs.next())
      {
         flag=true;
         //System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename")+"\t"+rs.getDoubl
e("esal")+"\t"+rs.getString("eaddr"));
         System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getDouble(3)+"\t"+rs.ge
tString(4));
      }
      if(flag==false)
      {
         System.out.println("No Records found");
      }
      con.close();
   }
}
```

Application-14: How to Select Range of Records based on Initial Characters of the Employee Name

```
import java.sql.*;
import java.util.*;
public class SelectRangeOfRecordsDemo3
```

```
{
  public static void main(String[] args) throws Exception
  {
    String driver="oracle.jdbc.OracleDriver";
    String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
    String user="scott";
    String pwd="tiger";
    Class.forName(driver);
    Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
    Statement st = con.createStatement();
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter Initial Characters of Employee Name:");
    String initialChar=sc.next()+"%";
    String sqlQuery=String.format("select * from employees where ename like '%s'",initialChar);
    boolean flag=false;
    ResultSet rs =st.executeQuery(sqlQuery);
    System.out.println("ENO\tENAME\tESALARY\tEADDR");
    System.out.println("---------------------------------------");
    while(rs.next())
    {
      flag=true;
      //System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename")+"\t"+rs.getDouble("esal")+"\t"+rs.getString("eaddr"));
      System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getDouble(3)+"\t"+rs.getString(4));
    }
    if(flag==false)
    {
      System.out.println("No Records found");
    }
    con.close();
  }
}
```

Aggregate Functions

Oracle Database defines several Aggregate Functions to get Summary Results like the Number of Records, Maximum Value of a particular Column etc

count(*)	➔ Returns The Number of Records
max(esal)	➔ Returns Maximum Salary
min(esal)	➔ Returns Minimum Salary

Eg:

```
String sqlQuery="select count(*) from employees";
ResultSet rs =st.executeQuery(sqlQuery);
if(rs.next())
{
  System.out.println(rs.getInt(1));
}
```

Note: If Number of Records is more, then we should use *while* Loop.

If Number of Records is only one then we should use *if* Statement.

Application-15: To Display Number of Rows by SQL Aggregate Function count(*)

Note: SQL Aggregate Function: count(*) Returns Number of Rows Present in the Table

```
import java.sql.*;
public class RowCountDemo
{
    public static void main(String[] args) throws Exception
    {
       String driver="oracle.jdbc.OracleDriver";
       String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
       String user="scott";
       String pwd="tiger";
       Class.forName(driver);
       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
       Statement st = con.createStatement();
       String sqlQuery="select count(*) from employees";
       ResultSet rs =st.executeQuery(sqlQuery);
       if(rs.next())
       {
          System.out.println(rs.getInt(1));
       }
       con.close();
    }
}
```

Application-16: How to Select highest salaried Employee Information by using SQL Aggregate Function Max

```
import java.sql.*;
public class HighestSalaryEmpDemo
{
    public static void main(String[] args) throws Exception
    {
       String driver="oracle.jdbc.OracleDriver";
       String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
       String user="scott";
       String pwd="tiger";
       Class.forName(driver);
       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
       Statement st = con.createStatement();
       String sqlQuery="select  * from employees where esal in (select max(esal) from emplo
yees)";
       ResultSet rs =st.executeQuery(sqlQuery);
       if(rs.next())
       {
          System.out.println("Highest sal employee information");
          System.out.println("---------------------------------------");
          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getDouble(3)+"\t"+rs.ge
tString(4));
       }
       con.close();
    }
}
```

Note: To find Minimum salaried Employee Information

```
String sqlQuery="select  * from employees where esal in (select min(esal) from employees)";
```

Application-17: How to Select Nth Highest Salaried Employee Information

```java
import java.sql.*;
import java.util.*;
public class NthHighestSalaryEmpDemo
{
   public static void main(String[] args) throws Exception
   {
      String driver="oracle.jdbc.OracleDriver";
      String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
      String user="scott";
      String pwd="tiger";
      Class.forName(driver);
      Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
      Statement st = con.createStatement();
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter Number:");
      int n = sc.nextInt();
      String sqlQuery="select  * from ( select eno,ename,esal,eaddr, rank() over (order by es
al DESC) ranking from employees)  where ranking="+n;
      ResultSet rs =st.executeQuery(sqlQuery);
      while(rs.next())
      {
         System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getDouble(3)+"\t"+rs.ge
tString(4));
      }
      con.close();
   }
}
```

Note: The Rank Function will assign a ranking to each Row starting from 1.

```
select eno,ename,esal,eaddr, rank() over (order by esal DESC)
```

```
700 Sree Mukhi   7000 Hyd ➔ 1
600 Anasooya     6000 Hyd ➔ 2
500 Reshmi       5000 Hyd ➔ 3
400 Veena        4000 Chennai ➔ 4
300 Mallika      3500 Chennai ➔ 5
200 Sunny        2000 Mumbai ➔ 6
100 Durga        1000 Hyd ➔ 7
```

Application-18: How to Display retrieved Data from the Database through HTML

```java
import java.sql.*;
import java.io.*;
public class SelectAllRowsToHtmlDemo
{
    public static void main(String[] args) throws Exception
    {
      String driver="oracle.jdbc.OracleDriver";
      String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
      String user="scott";
      String pwd="tiger";
      String sqlQuery="select * from employees";
      Class.forName(driver);
      Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
      Statement st = con.createStatement();
      ResultSet rs =st.executeQuery(sqlQuery);
      String data="";
      data = data+"<html><body><center><table border='1' bgcolor='green'>";
     data=data+"<tr><td>ENO</td><td>ENAME</td><td>ESAL</td><td>EADDR</td></tr>";
      while(rs.next())
      {
        data=data+"<tr><td>"+rs.getInt(1)+"</td><td>"+rs.getString(2)+"</td><td>"+rs.get
Double(3)+"</td><td>"+rs.getString(4)+"</td></tr>";
      }
      data=data+"</table><center></body></html>";
      FileOutputStream fos = new FileOutputStream("emp.html");
      byte[] b = data.getBytes();
      fos.write(b);
      fos.flush();
      System.out.println("Open emp.html to get Employees data");
      fos.close();
      con.close();
    }
}
```

Application-19: How to execute Select and Non-Select Queries by using execute() Method

```
import java.sql.*;
import java.util.*;
public class SelectNonSelectDemo {
    public static void main(String[] args) throws Exception
    {
      String driver="oracle.jdbc.OracleDriver";
      String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
      String user="scott";
      String pwd="tiger";
      Class.forName(driver);// This step is optional
      Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
      Statement st = con.createStatement();
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter the Query: ");
      String sqlQuery=sc.nextLine();
      boolean b = st.execute(sqlQuery);
      if(b== true)//select query
      {
            ResultSet rs =st.getResultSet();
            while(rs.next())
           {
              System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getDouble(3)+"\t"+rs.
getString(4));
            }
        }
        else //non-select query
        {
            int rowCount=st.getUpdateCount();
            System.out.println("The number of records effected is:"+rowCount);
        }
        con.close();
    }
}
```

Application-20: Execute Methods LoopHoles-1: executeQuery() Vs Non-select

If we pass Non-Select Query as Argument to *executeQuery()* Method then Result is varied from Driver to Driver. In the case of Type-1 Driver we will get *SQLException : No ResultSet* was produced.
In the case of Type-4 Driver provided by Oracle then we won't get any Exception and Empty ResultSet Object will be created. If we are trying to access that ResultSet then we will get *SQLException*

For Type-1:

```
import java.sql.*;
public class ExecuteMethodLoopHoles2T1
{
    public static void main(String[] args) throws Exception
    {
        String driver="sun.jdbc.odbc.JdbcOdbcDriver";
        String jdbc_url="jdbc:odbc:demodsn";
        String user="scott";
        String pwd="tiger";
        Class.forName(driver);
        Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
        Statement st = con.createStatement();
    ResultSet rs=st.executeQuery("update employees set esal=7777 where ename='durga'");
        con.close();
    }
}
```

For Type-4 Driver:

```
import java.sql.*;
public class ExecuteMethodLoopHoles2T4
{
    public static void main(String[] args) throws Exception
    {
        String driver="oracle.jdbc.OracleDriver";
        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
        String user="scott";
        String pwd="tiger";
        Class.forName(driver);
        Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
        Statement st = con.createStatement();
    ResultSet rs=st.executeQuery("update employees set esal=7777 where ename='durga'");
        con.close();
    }
}
```

Application-21: Execute Methods LoopHoles-2: executeUpdate() Vs Select

If we send Select Query as Argument to *executeUpdate()* Method then Result is varied from Driver to Driver. In the case of Type-1 Driver we will get *SQLException: No Row Count was produced.*

In the case of Type-4 Driver provided by Oracle then we won't get any Exception and Returns the Number of Records retrieved from the Database.

For Type-1:

```
import java.sql.*;
public class ExecuteMethodLoopHoles3T1
{
    public static void main(String[] args) throws Exception
    {
        String driver="sun.jdbc.odbc.JdbcOdbcDriver";
        String jdbc_url="jdbc:odbc:demodsn";
        String user="scott";
        String pwd="tiger";
        Class.forName(driver);
        Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
        Statement st = con.createStatement();
        int rowCount=st.executeUpdate("select * from employees");
        System.out.println(rowCount);
        con.close();
    }
}
```

For Type-4 Driver:

```
import java.sql.*;
public class ExecuteMethodLoopHoles3T4
{
    public static void main(String[] args) throws Exception
    {
        String driver="oracle.jdbc.OracleDriver";
        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
        String user="scott";
        String pwd="tiger";
        Class.forName(driver);
        Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
        Statement st = con.createStatement();
        int rowCount=st.executeUpdate("select * from employees");
        System.out.println(rowCount);
        con.close();
    }
}
```

Application-20: Execute Methods LoopHoles-3:executeUpdate() Vs DDL

If we use executeUpdate() Method for DDL Queries like Create Table, Drop Table Etc. Then Record Manipulation is not available on Database. In this Case Return Value we cannot expect and it is varied from Driver to Driver. For Type-1 Driver we will get 1 and for Oracle Type-4 Driver we will get 0.

For Type-1 Driver:

```
import java.sql.*;
public class ExecuteMethodLoopHoles1T1
{
    public static void main(String[] args) throws Exception
    {
        String driver="sun.jdbc.odbc.JdbcOdbcDriver";
        String jdbc_url="jdbc:odbc:demodsn";
        String user="scott";
        String pwd="tiger";
        Class.forName(driver);
        Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
        Statement st = con.createStatement();
        int updateCount=st.executeUpdate("create table emp1(eno number)");
        System.out.println(updateCount);//-1
        con.close();
    }
}
```

For Type-4 Driver:

```
import java.sql.*;
public class ExecuteMethodLoopHoles1T4
{
    public static void main(String[] args) throws Exception
    {
        String driver="oracle.jdbc.OracleDriver";
        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
        String user="scott";
        String pwd="tiger";
        Class.forName(driver);
        Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
        Statement st = con.createStatement();
        int updateCount=st.executeUpdate("create table emp2(eno number)");
        System.out.println(updateCount);//0
        con.close();
    }
}
```

Real Time Coding Standards for JDBC Application

1. Every Java Class should be Part of some Package. Hence it is recommended to take Package Statement.

2. It is recommended to use explicit Class Imports than implicit Class Imports because these Imports improve Readability of the Code.

Eg:

```
import java.sql.*; ➔ Implicit Class Import
import java.sql.Connection; ➔ Explicit Class Import
```

3. It is recommended to use *try-catch* over *throws* Statement, because there is a Guarantee for the Normal Termination of the Program.

Even we are using *throws* Statement, somewhere compulsory we should handle that Exception by using *try-catch.*

Eg:

```
m1()
{
    try
    {
        m2();
    }
    catch(Exception e)
    {
    }
}

m2() throws Exception
{
    ....
}
```

4. Avoid Duplicate Code as much as possible, otherwise Maintenance Problems may rise.

5. **We have to use meaningful Names for Classes, Methods, Variables etc. It improves Readability of the Code.**

If any Code repeatedly required, we have to separate that Code inside some other Class and we can call its Functionality where ever it is required.

In JDBC Applications, getting Connection and closing the Resources are common Requirement. Hence we can separate this Code into some *util* Class, and we can reuse that Code where ever it is required.

Program-1: To Demonstrate JDBC Coding Standards

```java
package com.durgasoft.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
/**
* @ Author: Durga
* @ Company: DURGASOFT
* @ see: www.durgasoft.com
*/
public class JDBCCodingStandardsDemo1
{
   public static void main(String[] args)
   {
      try
      {
         Class.forName("oracle.jdbc.OracleDriver");
      }
      catch(ClassNotFoundException e)
      {
         e.printStackTrace();
      }
      Connection con=null;
      Statement st = null;
      ResultSet rs=null;
      try
      {
         con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","scott","
tiger");
         st=con.createStatement();
         rs=st.executeQuery("select * from employees");
         while(rs.next())
         {
            System.out.println(rs.getInt(1)+".."+rs.getString(2)+"..."+rs.getDouble(3)+".."+rs.g
etString(4));
         }
      }
```

```
        catch(SQLException e)
        {
            e.printStackTrace();
        }
        finally
        {
            try
            {
                if(rs!= null)
                    rs.close();
                if(st!= null)
                    st.close();
                if(con!= null)
                    con.close();
            }
            catch(SQLException e)
            {
                e.printStackTrace();
            }
        }
    }
}
```

javac -d . JDBCCodingStandardsDemo1.java
java com.durgasoft.jdbc.JDBCCodingStandardsDemo1

Note:
If any code repeatedly required then it is not recommended to write that code every time separately. We have to define that code inside a separate component and we can call that code where ever it is required without rewriting. It promotes reusability of the code.

Program-2: To Demonstrate JDBC Coding Standards with Code Reusability

JDBCCodingStandardsDemo2.java:

```
package com.durgasoft.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
/**
* @ Author: Durga
* @ Company: DURGASOFT
* @ see: www.durgasoft.com
*/
public class JDBCCodingStandardsDemo2
{
    public static void main(String[] args)
    {
        Connection con=null;
        Statement st = null;
        ResultSet rs=null;
        try
        {
            con=JdbcUtil.getOracleConnection();
            st=con.createStatement();
            rs=st.executeQuery("select * from employees");
            while(rs.next())
            {
                System.out.println(rs.getInt(1)+".."+rs.getString(2)+"..."+rs.getDouble(3)+".."+rs.g
etString(4));
            }
        }
        catch(SQLException e)
        {
            e.printStackTrace();
        }
        finally
        {
            JdbcUtil.cleanup(con,st,rs);
        }
    }
}
```

JdbcUtil.java:

```
package com.durgasoft.jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
public class JdbcUtil
{
    static
    {
        try
        {
            Class.forName("oracle.jdbc.OracleDriver");
        }
        catch(ClassNotFoundException e)
        {
            e.printStackTrace();
        }
    }
    public static Connection getOracleConnection()throws SQLException
    {
        Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE"
,"scott","tiger");
```

```
        return con;
    }
    public static void cleanup(Connection con,Statement st,ResultSet rs)
    {
        try
        {
            if(rs!= null)
                rs.close();
            if(st!= null)
                st.close();
            if(con!= null)
                con.close();
        }
        catch(SQLException e)
        {
            e.printStackTrace();
        }
    }
}
```

```
javac -d . JDBCCodingStandardsDemo2.java
java com.durgasoft.jdbc.JDBCCodingStandardsDemo2
```

Working with MySQL Database

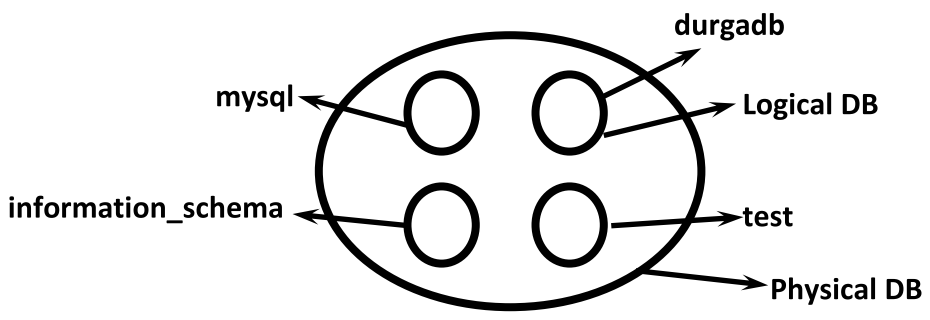
Current Version :5.7.14
Vendor: Sun Microsystems/Oracle Corporation
Open Source And Freeware
Default Port: 3306
Default User: root

Note:

In MySql, everything we have to work with our own Databases, which are also known as *Logical Databases.*

The Default Databases are:

```
information_schema
mysql
test
```



Here only one Physical Database but 4 Logical Databases are available.

Commonly used Commands:

1. To know available Databases

```
mysql> show databases;
```

2. To Create our own Logical Database

```
mysql> create database durgadb;
```

3. To Drop our own Database

```
mysql> drop database durgadb;
```

4. To use a particular Logical Database

```
mysql> use durgadb;        OR     mysql> connect durgadb;
```

5. To Create a Table:

create table employees(eno int(5) primary key,ename varchar(20),esal double(10,2),eaddr varchar(20));

6. To Insert Data

insert into employees values(100,'durga',1000,'Hyd');

Instead of Single Quotes, we can use Double Quotes also.

JDBC Information:

In general, we can use Type-4 Driver to communicates with MySQL Database which is provided by MySQL Vendor, and its Name is connector/J

Jar File: Driver Software is available in the following Jar File.

mysql-connector-java-5.1.41-bin.jar

We have to download separately from MySql Web Site.

jdbc url: jdbc:mysql://localhost:3306/durgadb
jdbc:mysql:///durgadb

If MySQL is available in Local System then we can specify JDBC URL as above.

Driver Class Name: com.mysql.jdbc.Driver
User Name: root
pwd: root

We required to Set Class Path of MySql Driver Jar File

Variable Name: CLASSPATH
Variable Value: D:\mysql-connector-java-bin.jar;.;

Program to Demonstrate JDBC with MySql Database

```
import java.sql.*;
public class JdbcMySQLDemo
{
  public static void main(String[] args) throws Exception
  {
    Class.forName("com.mysql.jdbc.Driver");
    Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/durgadb","root","root");
    Statement st = con.createStatement();
    ResultSet rs =st.executeQuery("select * from employees");
```

```
        while(rs.next())
        {
           System.out.println(rs.getInt(1)+".."+rs.getString(2)+".."+rs.getDouble(3)+".."+rs.get
String(4));
        }
        con.close();
    }
}
```

Program to demonstrate copy data from Oracle to MySql database

```
import java.sql.*;
class OracleToMySQL
{
    public static void main(String[] args) throws Exception
    {
        int count=0;
        Connection con1= DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:X
E","scott","tiger");
        Connection con2 = DriverManager.getConnection("jdbc:mysql://localhost:3306/durga
db","root","root");
        Statement st1= con1.createStatement();
        Statement st2= con2.createStatement();
        ResultSet rs=st1.executeQuery("select * from employees");
        while(rs.next())
        {
           count++;
           int eno=rs.getInt(1);
           String ename=rs.getString(2);
           double esal=rs.getDouble(3);
           String eaddr=rs.getString(4);
           String sqlQuery=String.format("insert into employees values(%d,'%s',%f,'%s')",eno,e
name,esal,eaddr);
           st2.executeUpdate(sqlQuery);
        }
        System.out.println("Total Data copied from Oracle to MySQL and number of records:"
+count);
        con1.close();
        con2.close();
    }
}
```