Advanced Java

DURGA SOFTWARE SOLUTIONS

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.

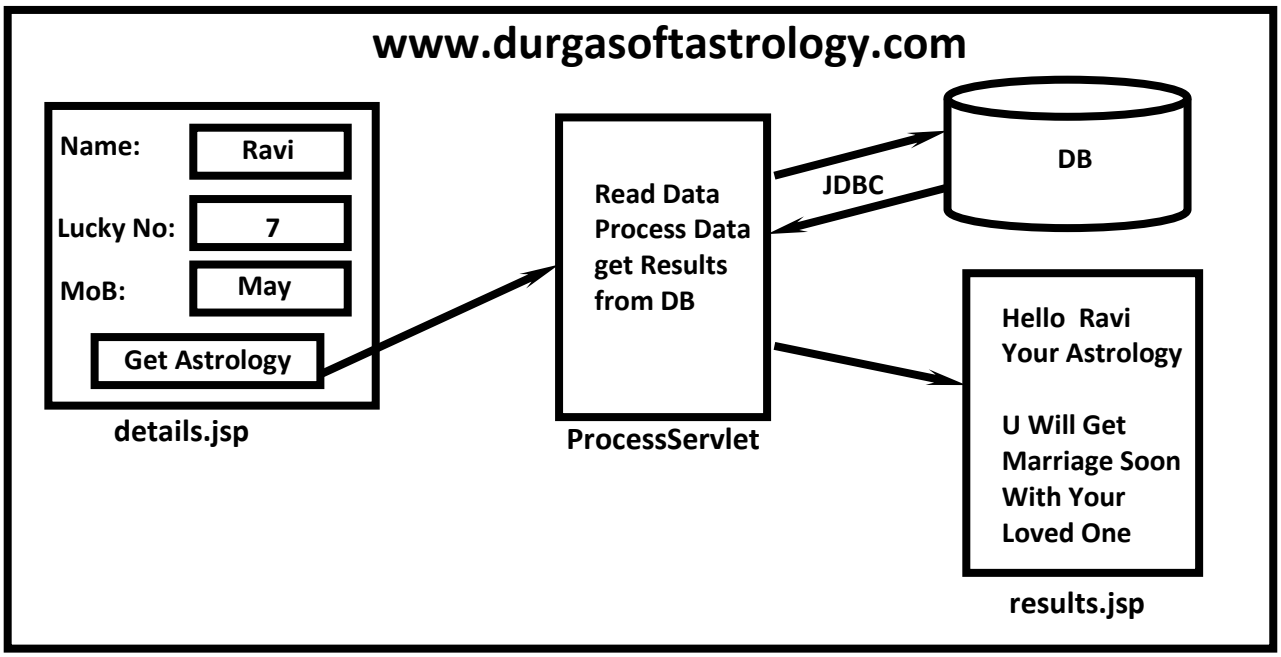
Eg: display login page
display inbox page
display error page
display result page
etc..

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

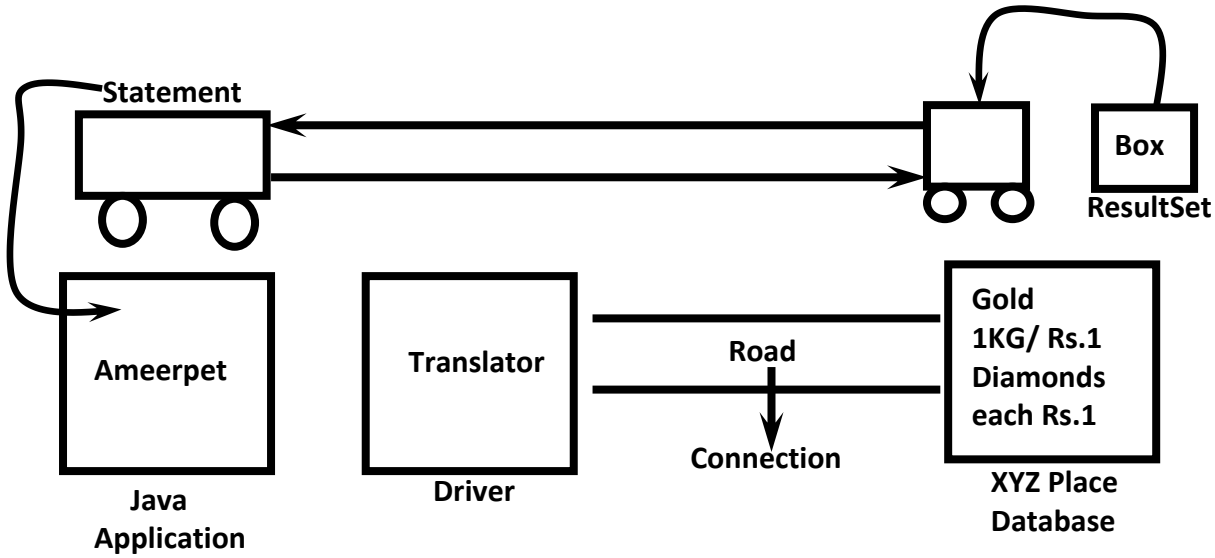
Current Version of JSE is Java 1.8
Current Version of JDBC is: 4.2 V

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:

```
1)  import java.sql.*;
2)  public class JdbcDemo
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:demodsn","scott","tiger");
8)        Statement st = con.createStatement();
9)        ResultSet rs= st.executeQuery("select * from employees");
10)       while(rs.next())
11)       {
12)          System.out.println(rs.getInt(1)+".."+rs.getString(2)+".."+rs.getDouble(3)+"..."+rs.get
     String(4));
13)       }
14)       con.close();
15)    }
16) }
```

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.

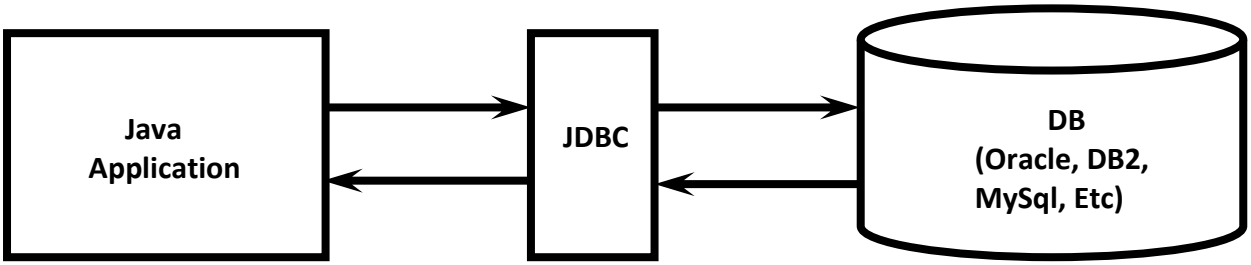
Limitations of Databases:

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

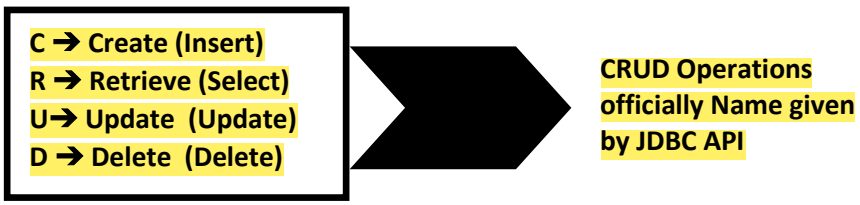
- **JDBC is a Technology, which can be used to communicate with Database from Java Application.**



- **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.**

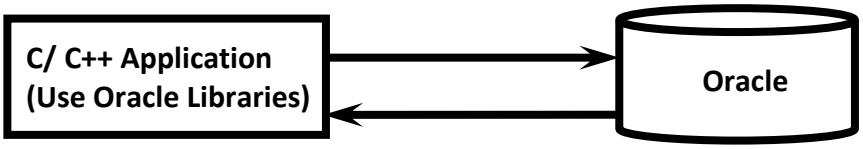
List of supported Vendors we can check in the link
http://www.oracle.com/technetwork/java/index-136695.html

JDBC Versions:

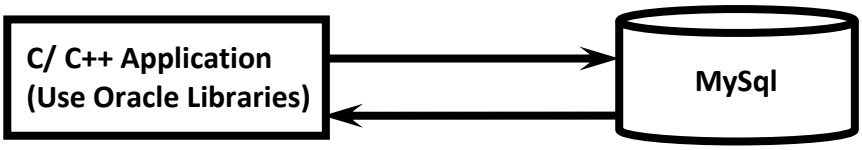
- JDBC 3.0 is Part J2SE 1.4
- No Update in Java SE 5.0
- JDBC 4.0 is Part Java SE 6.0
- JDBC 4.1 is Part Java SE 7.0
- JDBC 4.2 is Part Java SE 8.0

Evolution of JDBC:

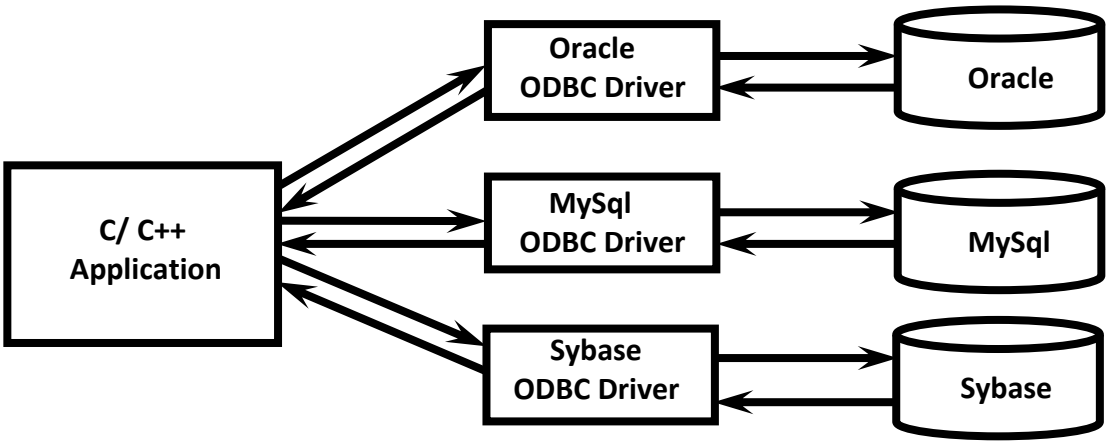
- If we want to communicate with Database by using C OR C++, compulsory we have to use database specific Libraries in our Application directly.



- 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.



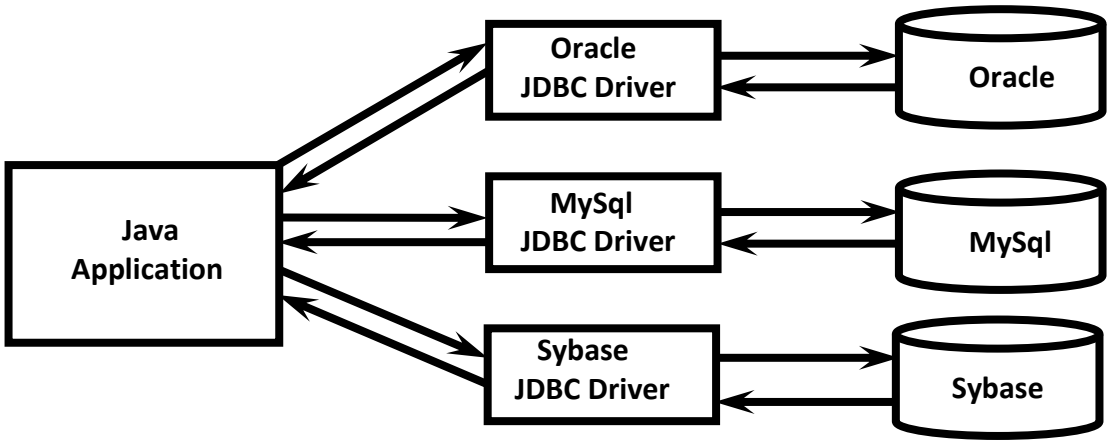
- The Application will become Database Dependent and creates Maintenance Problems.
- To overcome this Problem, Microsoft People introduced "ODBC" Concept in 1992. It is Database Independent API.
- With ODBC API, Application can communicate with any Database just by selecting corresponding ODBC Driver.
- 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.



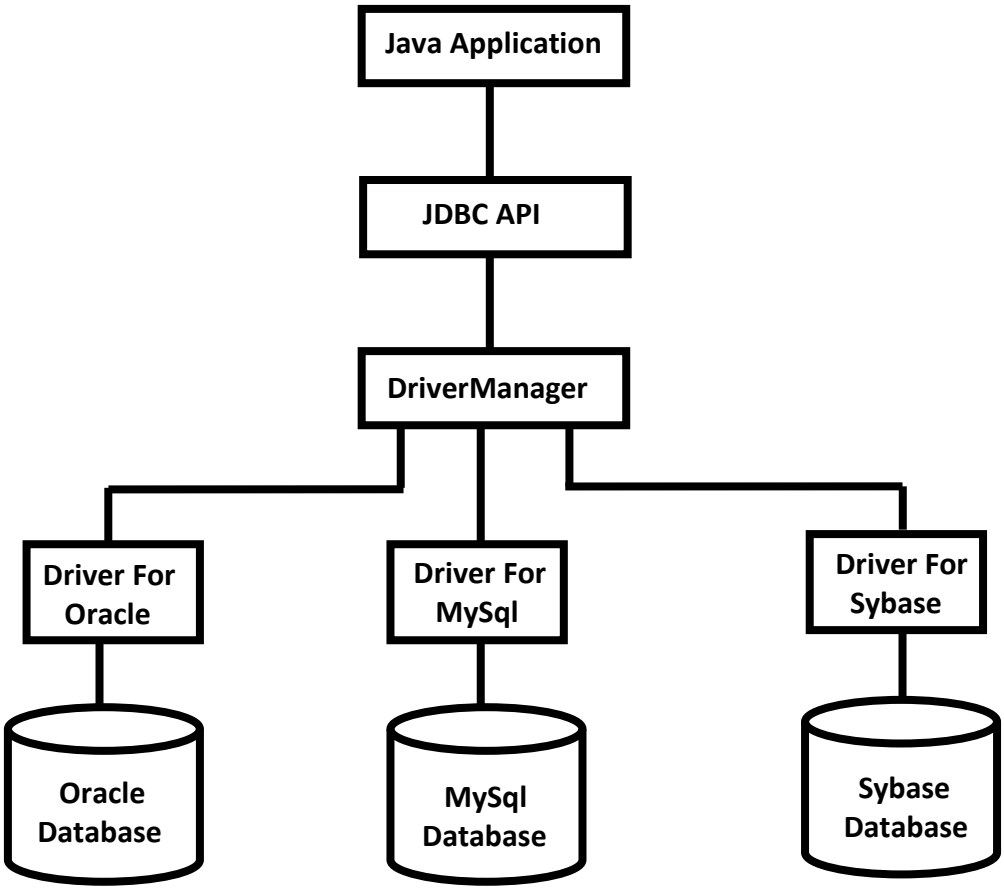
***Note:

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

2) javax.sql Package:

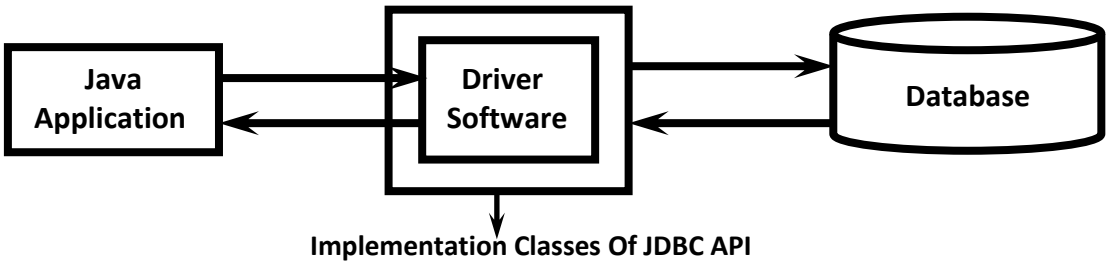
It defines more advanced Classes and Interfaces which can be used for Database Communication.

There are multiple Sub Packages are also available

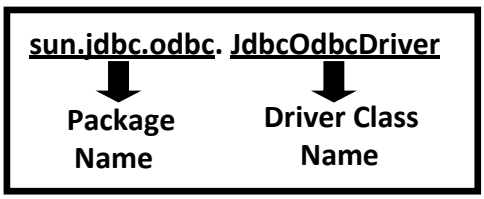
- javax.sql.rowset;
- javax.sql.rowset.serial;
- javax.sql.rowset.spi;

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

- 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)

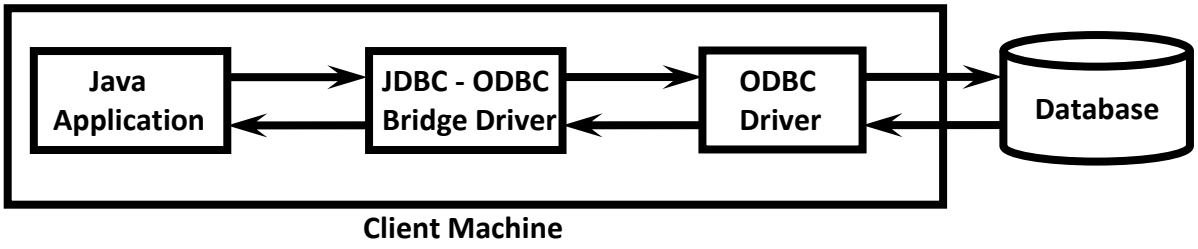
4) Type-4 Driver (All Java Native Protocol Driver OR Pure Java Driver OR Thin Driver)

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.

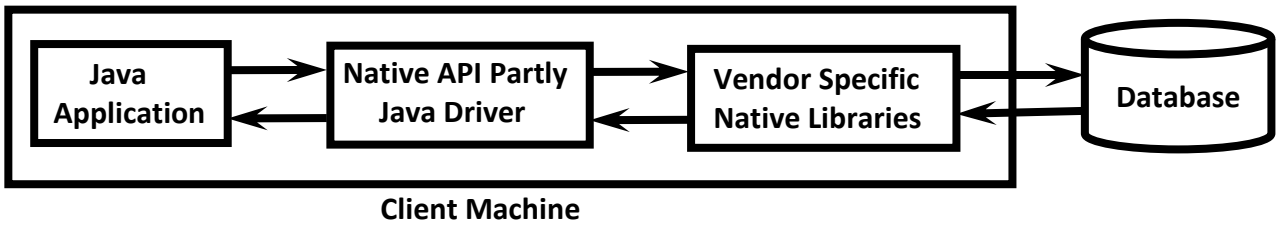
3. No Support from JDK 1.8 Version onwards.

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*.



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"

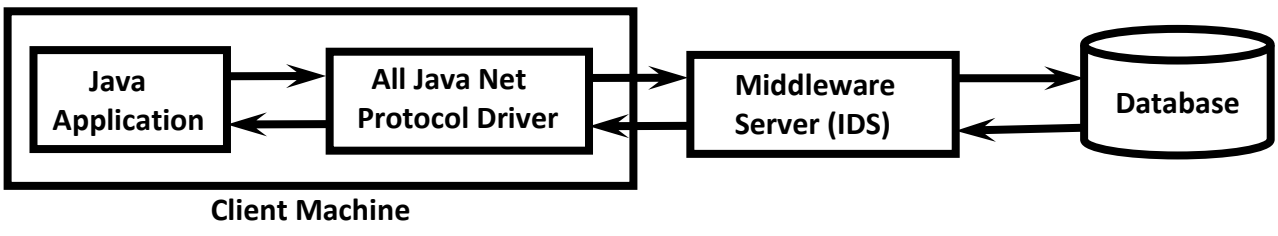
OCI Driver and corresponding OCI Libraries are available in the following Jar File. Hence we have to place this Jar File in the Class Path.

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.

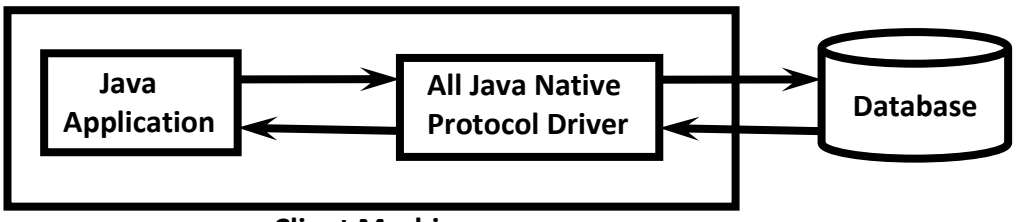
2. We need to purchase Middleware Server and hence the cost of this Driver is more when compared with remaining Drivers.

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.

Eg: 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.

Eg: 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.

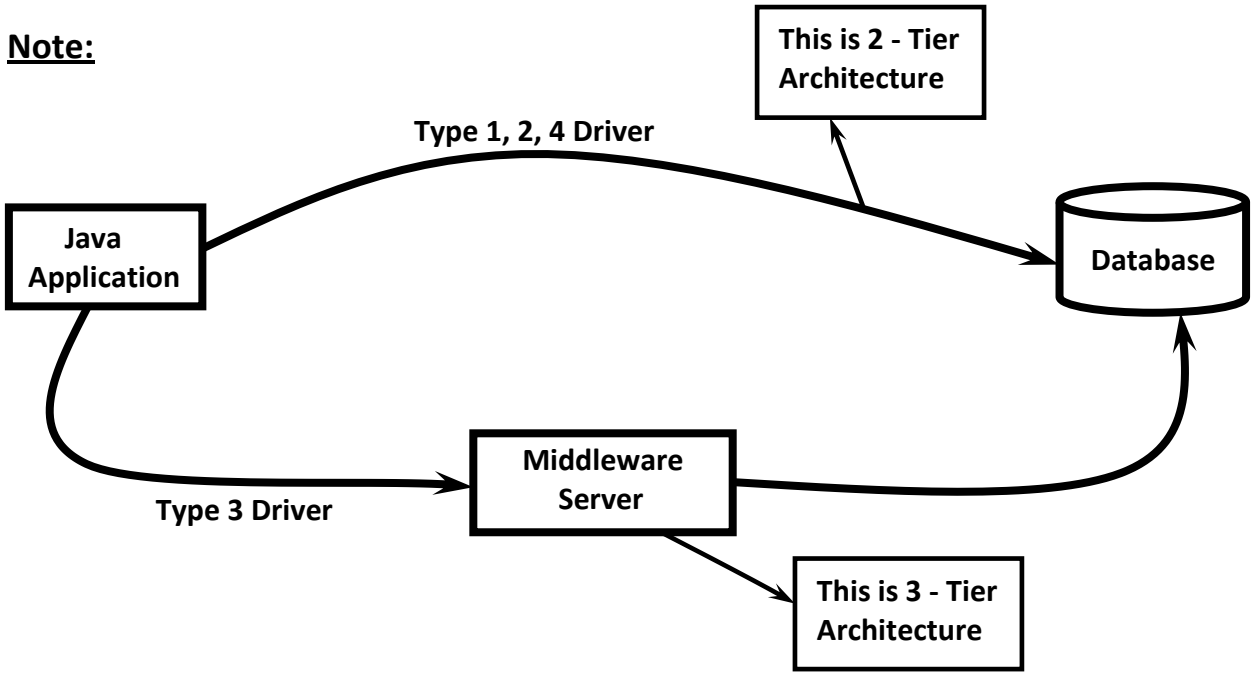
4. If no other Driver is available then only we should go for Type-1 Driver.

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.
Eg: **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.
Eg: **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.

```
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:

```
<mainprotocol> : <subprotocol> : <subname>
      ↓                ↓              ↓
  Always JDBC    Driver Protocol   Driver Connection
                                       Detail
```

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

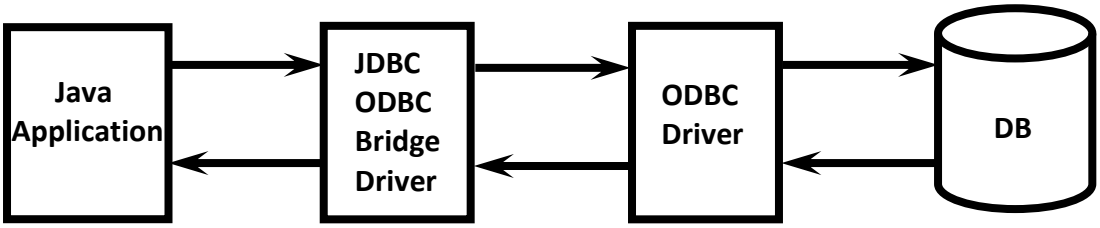
Eg: Connection con= DriverManager.getConnection("jdbc:odbc:demodsn","scott","tiger");

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:

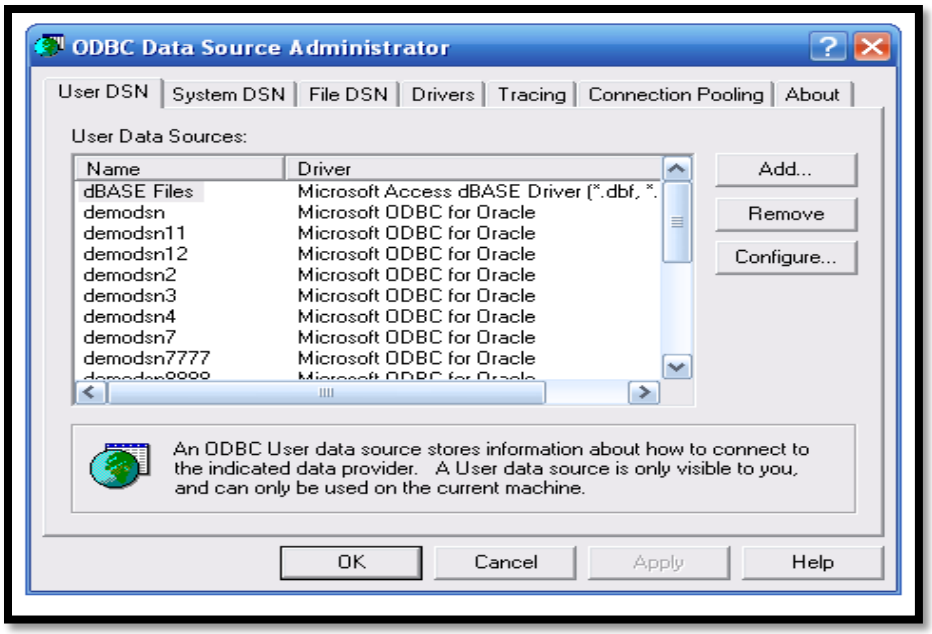
It is the sharable DSN and it is available for all Users who can access that System.
It is also known as Global DSN.

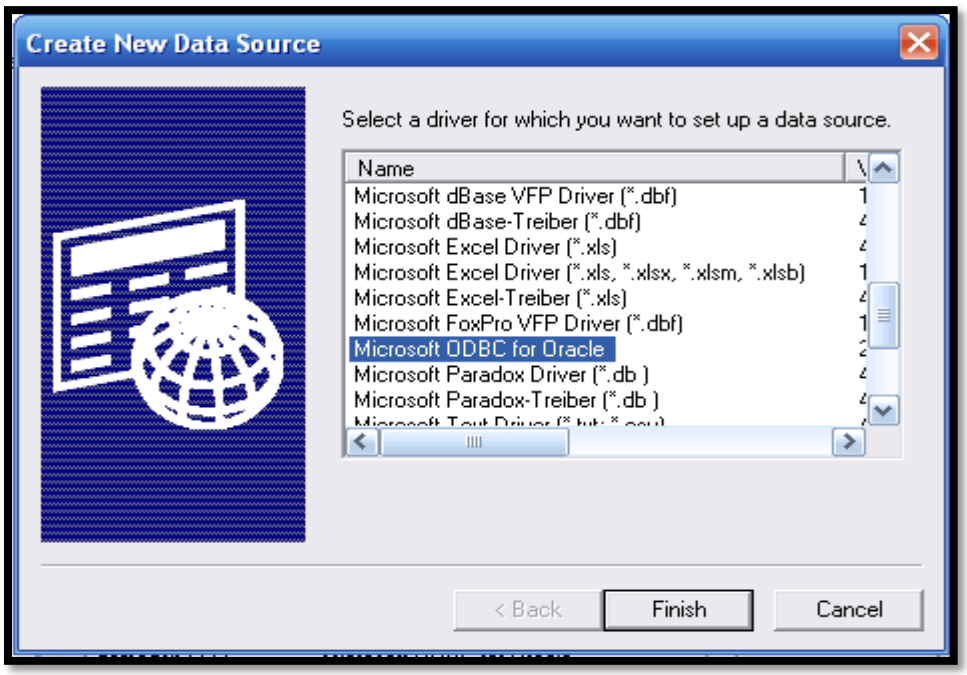
3) File DSN:

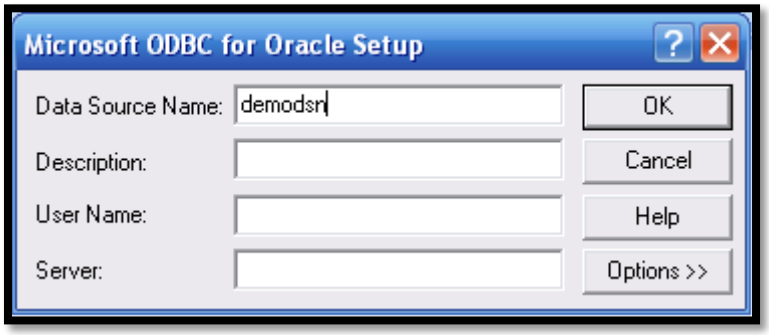
It is exactly same as User DSN but will be stored in a File with .dsn Extension.

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
Add...
Remove
Configure...
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.
OK
Cancel
Apply
Help


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)
< Back
Finish
Cancel


Microsoft ODBC for Oracle Setup
Data Source Name:
demodsn
Description:
User Name:
Server:
OK
Cancel
Help
Options >>

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)");
```

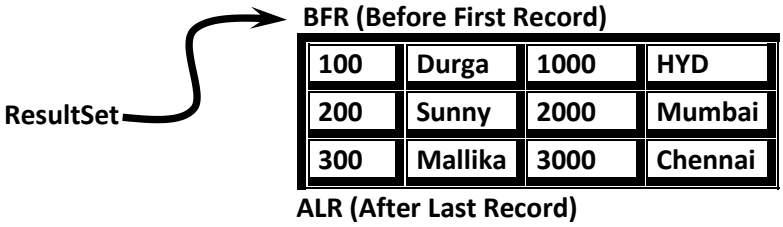
For Type-1 Driver, we will get -1 and For Type-4 Driver, we will get 0

```
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();*

1.7 Version: try With Resources

Usually we will close the Resources inside finally Block.

```
1)  try
2)  {
3)     Open Database Connection
4)  }
5)  catch(X e)
6)  {
7)  }
8)  finally
9)  {
10)    Close That Database Connection
11) }
```

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.

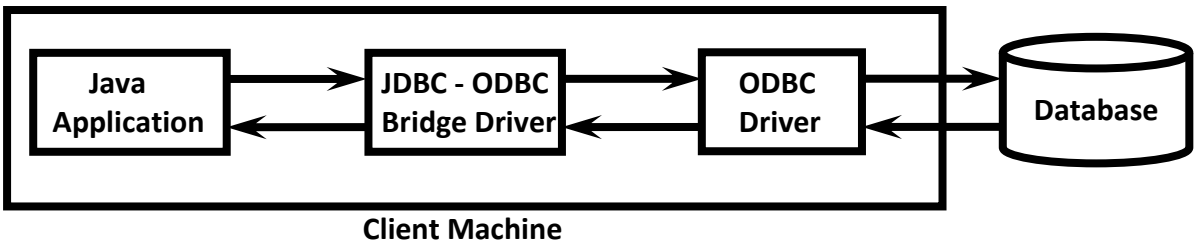
Hence until 1.6 Version *finally* Block is just like Hero but from 1.7 Version onwards *finally* Block became Zero.

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

Eg:

```
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;

Note: 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');
```

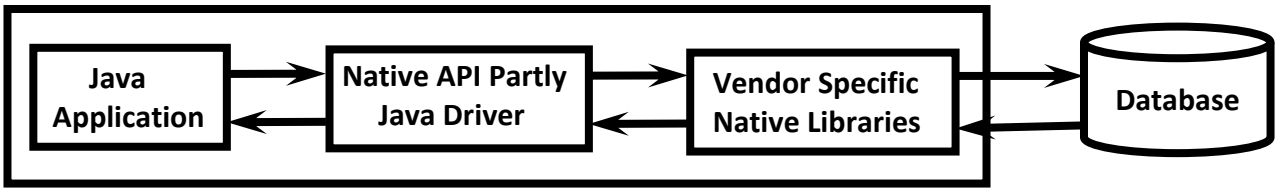
Eg:

```
1)  import java.sql.*;
2)  public class  Type1DriverDemo
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:demodsn","scott","tiger");
8)        Statement st = con.createStatement();
9)        ResultSet rs = st.executeQuery("select * from movies");
10)       while(rs.next())
11)       {
12)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getString(3)+"\t"+rs.get
     String(4));
13)       }
14)       con.close();
15)    }
16) }
```

Note:

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:



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;
```

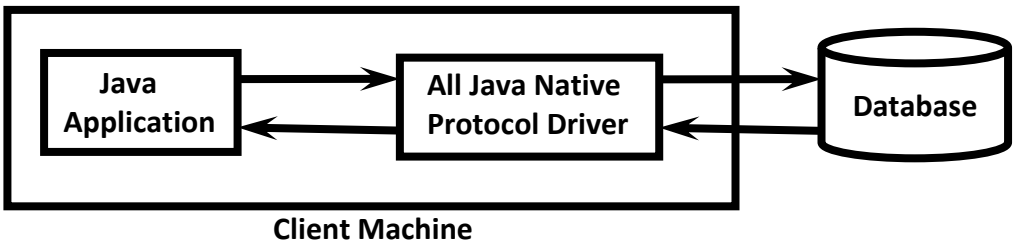
2nd way:

We can find SID from the following File

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

```
1)  import java.sql.*;
2)  public class  Type2DriverDemo
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Class.forName("oracle.jdbc.OracleDriver");
7)       Connection con= DriverManager.getConnection("jdbc:oracle:oci:@XE","scott","tiger");
8)        Statement st = con.createStatement();
9)        ResultSet rs = st.executeQuery("select * from movies");
10)       while(rs.next())
11)       {
12)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getString(3)+"\t"+rs.get
     String(4));
13)       }
14)       con.close();
15)    }
16) }
```

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.

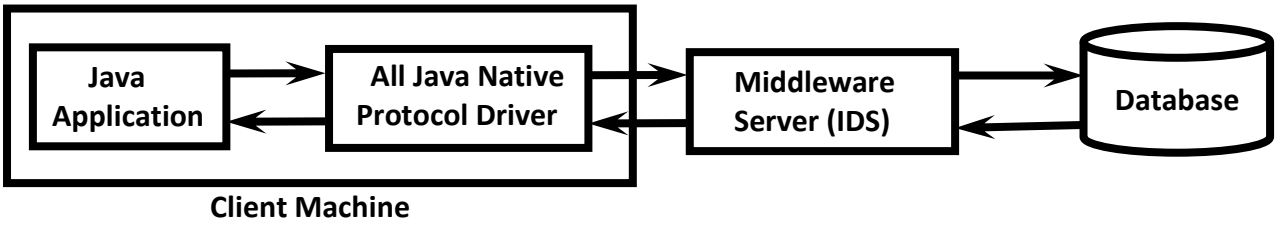
Driver Class Name: oracle.jdbc.driver.OracleDriver
oracle.jdbc.OracleDriver

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

```
1)  import java.sql.*;
2)  public class  Type4DriverDemo
3)  {
4)     public static void main(String[] args) throws Exception
```

```
5)    {
6)       Class.forName("oracle.jdbc.OracleDriver");
7)       Connection con= DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","scott","tiger");
8)       Statement st = con.createStatement();
9)       ResultSet rs = st.executeQuery("select * from movies");
10)      while(rs.next())
11)      {
12)         System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getString(3)+"\t"+rs.get
    String(4));
13)      }
14)      con.close();
15)   }
16) }
```

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".

```
1)  import java.sql.*;
2)  public class  Type3DriverDemo
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Class.forName("ids.sql.IDSDriver");
7)        Connection con= DriverManager.getConnection("jdbc:ids://localhost:12/conn?dsn=m
    ysysdsn","scott","tiger");
8)        Statement st = con.createStatement();
9)        ResultSet rs = st.executeQuery("select * from movies");
10)       while(rs.next())
11)       {
12)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getString(3)+"\t"+rs.get
    String(4));
13)       }
14)       con.close();
15)    }
16) }
```

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

```
1)  import java.sql.*;
2)  public class  Type5DriverDemo
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Class.forName("com.ddtek.jdbc.oracle.OracleDriver");
```

```
7)      Connection con= DriverManager.getConnection("datadirect:oracle://localhost:1521;S
    erviceName=XE","scott","tiger");
8)      Statement st = con.createStatement();
9)      ResultSet rs = st.executeQuery("select * from movies");
10)     while(rs.next())
11)     {
12)        System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getString(3)+"\t"+rs.get
    String(4));
13)     }
14)     con.close();
15)   }
16) }
```

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=demosystemdsn3
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.

```
1)  import java.util.*;
2)  class Test
3)  {
4)     public static void main(String[] args)
5)     {
6)        Scanner sc = new Scanner(System.in);
7)        System.out.println("Enter Employee Number:");
8)        int eno=sc.nextInt();
9)        System.out.println("Enter Employee Name:");
10)       String ename=sc.next();
11)       System.out.println("Enter Employee Salary:");
12)       double esal=sc.nextDouble();
13)       System.out.println("Enter Employee Address:");
14)       String eaddr=sc.next();
15)       System.out.println(eno+"\t"+ename+"\t"+esal+"\t"+eaddr);
16)    }
17) }
```

Application-1: How to Create a Table

```
1)  import java.sql.*;
2)  public class CreateTableDemo
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        String driver="oracle.jdbc.OracleDriver";
7)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
8)        String user="scott";
9)        String pwd="tiger";
10)       String sql_query="create table employees(eno number,ename varchar2(10),esal numb
      er,eaddr varchar2(10))";
11)       Class.forName(driver);
12)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
13)       Statement st = con.createStatement();
14)       st.executeUpdate(sql_query);
15)       System.out.println("Table Created Successfully");
16)       con.close();
17)    }
18) }
```

Application-2: How To Delete A Table

```
1)  import java.sql.*;
2)  public class DropTableDemo
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        String driver="oracle.jdbc.OracleDriver";
7)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
8)        String user="scott";
9)        String pwd="tiger";
10)       String sql_query="drop table students";
11)       Class.forName(driver);
12)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
13)       Statement st = con.createStatement();
14)       st.executeUpdate(sql_query);
15)       System.out.println("Table Deleted Successfully");
16)       con.close();
17)    }
18) }
```

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

```
1)  import java.sql.*;
2)  public class InsertSingleRowDemo
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        String driver="oracle.jdbc.OracleDriver";
7)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
8)        String user="scott";
9)        String pwd="tiger";
10)       String sql_query="insert into employees values(100,'durga',1000,'hyd')";
11)       Class.forName(driver);
12)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
13)       Statement st = con.createStatement();
14)       int updateCount=st.executeUpdate(sql_query);
15)       System.out.println("The number of rows inserted :"+updateCount);
16)       con.close();
17)    }
18) }
```

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

```
1)  import java.sql.*;
2)  import java.util.*;
3)  public class InsertMultipleRowsDemo
4)  {
5)     public static void main(String[] args) throws Exception
```

```
6)    {
7)       String driver="oracle.jdbc.OracleDriver";
8)       String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
9)       String user="scott";
10)      String pwd="tiger";
11)      Class.forName(driver);
12)      Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
13)      Statement st = con.createStatement();
14)      Scanner sc = new Scanner(System.in);
15)      while(true)
16)      {
17)         System.out.println("Employee Number:");
18)         int eno=sc.nextInt();
19)         System.out.println("Employee Name:");
20)         String ename=sc.next();
21)         System.out.println("Employee Sal:");
22)         double esal=sc.nextDouble();
23)         System.out.println("Employee Address:");
24)         String eaddr=sc.next();
25)         String sqlQuery=String.format("insert into employees values(%d,'%s',%f,'%s')",eno,e
    name,esal,eaddr);
26)         st.executeUpdate(sqlQuery);
27)         System.out.println("Record Inserted Successfully");
28)         System.out.println("Do U want to Insert one more record[Yes/No]:");
29)         String option = sc.next();
30)         if(option.equalsIgnoreCase("No"))
31)         {
32)            break;
33)         }
34)      }
35)      con.close();
36)   }
37) }
```

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

```
1)  import java.sql.*;
2)  public class UpdateSingleRowDemo
3)  {
```

```
4)    public static void main(String[] args) throws Exception
5)    {
6)       String driver="oracle.jdbc.OracleDriver";
7)       String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
8)       String user="scott";
9)       String pwd="tiger";
10)      String sql_query="update employees set esal=10000 where ename='durga'";
11)      Class.forName(driver);
12)      Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
13)      Statement st = con.createStatement();
14)      int updateCount=st.executeUpdate(sql_query);
15)      System.out.println("The number of rows updated :"+updateCount);
16)      con.close();
17)   }
18) }
```

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

```
1)  import java.sql.*;
2)  import java.util.*;
3)  public class UpdateMultipleRowsDemo
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        String driver="oracle.jdbc.OracleDriver";
8)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
9)        String user="scott";
10)       String pwd="tiger";
11)       Class.forName(driver);
12)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
13)       Statement st = con.createStatement();
14)       Scanner sc = new Scanner(System.in);
15)       System.out.println("Enter Bonus Amount:");
16)       double bonus =sc.nextDouble();
17)       System.out.println("Enter Salary Range:");
18)       double salRange =sc.nextDouble();
19)       String sqlQuery=String.format("update employees set esal=esal+%f where esal<%f",b
    onus,salRange);
20)       int updateCount=st.executeUpdate(sqlQuery);
21)       System.out.println("The number of rows updated :"+updateCount);
22)       con.close();
23)    }
24) }
```

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

```
1)  import java.sql.*;
2)  public class DeleteSingleRowDemo
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        String driver="oracle.jdbc.OracleDriver";
7)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
8)        String user="scott";
9)        String pwd="tiger";
10)       String sqlQuery="delete from employees where ename='durga'";
11)       Class.forName(driver);
12)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
13)       Statement st = con.createStatement();
14)       int updateCount=st.executeUpdate(sqlQuery);
15)       System.out.println("The number of rows deleted :"+updateCount);
16)       con.close();
17)    }
18) }
```

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

```
1)  import java.sql.*;
2)  import java.util.*;
3)  public class DeleteMultipleRowsDemo
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        String driver="oracle.jdbc.OracleDriver";
8)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
9)        String user="scott";
10)       String pwd="tiger";
11)       Class.forName(driver);
12)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
13)       Statement st = con.createStatement();
14)       Scanner sc = new Scanner(System.in);
15)       System.out.println("Enter CutOff Salary:");
```

```
16)       double cutOff =sc.nextDouble();
17)       String sqlQuery=String.format("delete from employees where esal>=%f",cutOff);
18)       int updateCount=st.executeUpdate(sqlQuery);
19)       System.out.println("The number of rows deleted :"+updateCount);
20)       con.close();
21)    }
22) }
```

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

```
1)  import java.sql.*;
2)  public class SelectAllRowsDemo
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        String driver="oracle.jdbc.OracleDriver";
7)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
8)        String user="scott";
9)        String pwd="tiger";
10)       Class.forName(driver);
11)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
12)       Statement st = con.createStatement();
13)       String sqlQuery="select * from employees";
14)       boolean flag= false;
15)       ResultSet rs =st.executeQuery(sqlQuery);
16)       System.out.println("ENO\tENAME\tESALARY\tEADDR");
17)       System.out.println("-------------------------------------");
18)       while(rs.next())
19)       {
20)          flag=true;
21)          //System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename")+"\t"+rs.getDoubl
     e("esal")+"\t"+rs.getString("eaddr"));
22)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getDouble(3)+"\t"+rs.ge
     tString(4));
23)       }
24)       if(flag==false)
25)       {
26)          System.out.println("No Records found");
27)       }
```

```
28)       con.close();
29)    }
30) }
```

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

```
1)  import java.sql.*;
2)  public class SelectAllRowsSortingDemo
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        String driver="oracle.jdbc.OracleDriver";
7)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
8)        String user="scott";
9)        String pwd="tiger";
10)       Class.forName(driver);
11)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
12)       Statement st = con.createStatement();
13)       String sqlQuery="select * from employees order by esal DESC";
14)       boolean flag= false;
15)       ResultSet rs =st.executeQuery(sqlQuery);
16)       System.out.println("ENO\tENAME\tESALARY\tEADDR");
17)       System.out.println("-------------------------------------");
18)       while(rs.next())
19)       {
20)          flag=true;
21)          //System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename")+"\t"+rs.getDoubl
    e("esal")+"\t"+rs.getString("eaddr"));
22)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getDouble(3)+"\t"+rs.ge
    tString(4));
23)       }
24)       if(flag==false)
25)       {
26)          System.out.println("No Records found");
27)       }
28)       con.close();
29)    }
30) }
```

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

```
1)  import java.sql.*;
2)  public class SelectParticularColumnsDemo
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        String driver="oracle.jdbc.OracleDriver";
7)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
8)        String user="scott";
9)        String pwd="tiger";
10)       Class.forName(driver);
11)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
12)       Statement st = con.createStatement();
13)       String sqlQuery="select ename,eaddr from employees";
14)       boolean flag=false;
15)       ResultSet rs =st.executeQuery(sqlQuery);
16)       System.out.println("ENAME\tEADDR");
17)       System.out.println("-----------------");
18)       while(rs.next())
19)       {
20)          flag=true;
21)          System.out.println(rs.getString("ename")+"\t"+rs.getString("eaddr"));
22)          //System.out.println(rs.getString(1)+"\t"+rs.getString(2));
23)       }
24)       if(flag==false)
25)       {
26)          System.out.println("No Records found");
27)       }
28)       con.close();
29)    }
30) }
```

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

```
1)  import java.sql.*;
2)  import java.util.*;
3)
4)  public class SelectRangeOfRecordsDemo1
5)  {
6)     public static void main(String[] args) throws Exception
7)     {
8)        String driver="oracle.jdbc.OracleDriver";
9)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
10)       String user="scott";
11)       String pwd="tiger";
12)       Class.forName(driver);
13)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
```

```
14)       Statement st = con.createStatement();
15)       Scanner sc = new Scanner(System.in);
16)       System.out.println("Enter City Name:");
17)       String addr=sc.next();
18)       String sqlQuery=String.format("select * from employees where eaddr='%s'",addr);
19)       boolean flag=false;
20)       ResultSet rs =st.executeQuery(sqlQuery);
21)       System.out.println("ENO\tENAME\tESALARY\tEADDR");
22)       System.out.println("-----------------------------------");
23)       while(rs.next())
24)       {
25)          flag=true;
26)          //System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename")+"\t"+rs.getDoubl
    e("esal")+"\t"+rs.getString("eaddr"));
27)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getDouble(3)+"\t"+rs.ge
    tString(4));
28)       }
29)       if(flag==false)
30)       {
31)          System.out.println("No Records found");
32)       }
33)       con.close();
34)    }
35) }
```

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

```
1)  import java.sql.*;
2)  import java.util.*;
3)
4)  public class SelectRangeOfRecordsDemo2
5)  {
6)     public static void main(String[] args) throws Exception
7)     {
8)        String driver="oracle.jdbc.OracleDriver";
9)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
10)       String user="scott";
11)       String pwd="tiger";
12)       Class.forName(driver);
13)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
14)       Statement st = con.createStatement();
15)       Scanner sc = new Scanner(System.in);
16)       System.out.println("Enter Begin Salary Range:");
17)       double beginSal=sc.nextDouble();
18)       System.out.println("Enter End Salary Range:");
19)       double endSal=sc.nextDouble();
20)       String sqlQuery=String.format("select * from employees where esal>%f and esal<%f",
    beginSal,endSal);
```

```
21)       boolean flag=false;
22)       ResultSet rs =st.executeQuery(sqlQuery);
23)       System.out.println("ENO\tENAME\tESALARY\tEADDR");
24)       System.out.println("-----------------------------------");
25)       while(rs.next())
26)       {
27)          flag=true;
28)          //System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename")+"\t"+rs.getDoubl
    e("esal")+"\t"+rs.getString("eaddr"));
29)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getDouble(3)+"\t"+rs.ge
    tString(4));
30)       }
31)       if(flag==false)
32)       {
33)          System.out.println("No Records found");
34)       }
35)       con.close();
36)    }
37) }
```

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

```
1)  import java.sql.*;
2)  import java.util.*;
3)  public class SelectRangeOfRecordsDemo3
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        String driver="oracle.jdbc.OracleDriver";
8)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
9)        String user="scott";
10)       String pwd="tiger";
11)       Class.forName(driver);
12)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
13)       Statement st = con.createStatement();
14)       Scanner sc = new Scanner(System.in);
15)       System.out.println("Enter Initial Characters of Employee Name:");
16)       String initialChar=sc.next()+"%";
17)       String sqlQuery=String.format("select * from employees where ename like '%s'",initial
    Char);
18)       boolean flag=false;
19)       ResultSet rs =st.executeQuery(sqlQuery);
20)       System.out.println("ENO\tENAME\tESALARY\tEADDR");
21)       System.out.println("-----------------------------------");
22)       while(rs.next())
23)       {
24)          flag=true;
```

```
25)         //System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename")+"\t"+rs.getDoubl
    e("esal")+"\t"+rs.getString("eaddr"));
26)         System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getDouble(3)+"\t"+rs.ge
    tString(4));
27)       }
28)       if(flag==false)
29)       {
30)         System.out.println("No Records found");
31)       }
32)       con.close();
33)    }
34) }
```

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

```
1)  import java.sql.*;
2)  public class RowCountDemo
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        String driver="oracle.jdbc.OracleDriver";
7)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
8)        String user="scott";
9)        String pwd="tiger";
10)       Class.forName(driver);
11)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
12)       Statement st = con.createStatement();
13)       String sqlQuery="select count(*) from employees";
14)       ResultSet rs =st.executeQuery(sqlQuery);
15)       if(rs.next())
16)       {
17)          System.out.println(rs.getInt(1));
18)       }
19)       con.close();
20)    }
21) }
```

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

```
1)  import java.sql.*;
2)  public class HighestSalaryEmpDemo
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        String driver="oracle.jdbc.OracleDriver";
7)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
8)        String user="scott";
9)        String pwd="tiger";
10)       Class.forName(driver);
11)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
12)       Statement st = con.createStatement();
13)       String sqlQuery="select  * from employees where esal in (select max(esal) from emplo
     yees)";
14)       ResultSet rs =st.executeQuery(sqlQuery);
15)       if(rs.next())
16)       {
17)          System.out.println("Highest sal employee information");
18)          System.out.println("----------------------------------------");
19)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getDouble(3)+"\t"+rs.ge
     tString(4));
20)       }
21)       con.close();
22)    }
23) }
```

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

```
1)  import java.sql.*;
2)  import java.util.*;
3)  public class NthHighestSalaryEmpDemo
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        String driver="oracle.jdbc.OracleDriver";
8)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
9)        String user="scott";
10)       String pwd="tiger";
11)       Class.forName(driver);
12)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
13)       Statement st = con.createStatement();
14)       Scanner sc = new Scanner(System.in);
```

```
15)       System.out.println("Enter Number:");
16)       int n = sc.nextInt();
17)       String sqlQuery="select  * from ( select eno,ename,esal,eaddr, rank() over (order by es
     al DESC) ranking from employees)  where ranking="+n;
18)       ResultSet rs =st.executeQuery(sqlQuery);
19)       while(rs.next())
20)       {
21)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getDouble(3)+"\t"+rs.ge
     tString(4));
22)       }
23)       con.close();
24)    }
25) }
```

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

```
1)  import java.sql.*;
2)  import java.io.*;
3)  public class SelectAllRowsToHtmlDemo
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        String driver="oracle.jdbc.OracleDriver";
8)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
9)        String user="scott";
10)       String pwd="tiger";
11)       String sqlQuery="select * from employees";
12)       Class.forName(driver);
13)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
14)       Statement st = con.createStatement();
15)       ResultSet rs =st.executeQuery(sqlQuery);
16)       String data="";
17)       data = data+"<html><body><center><table border='1' bgcolor='green'>";
18)      data=data+"<tr><td>ENO</td><td>ENAME</td><td>ESAL</td><td>EADDR</td></tr>";
19)       while(rs.next())
20)       {
```

```
21)         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>";
22)       }
23)       data=data+"</table><center></body></html>";
24)       FileOutputStream fos = new FileOutputStream("emp.html");
25)       byte[] b = data.getBytes();
26)       fos.write(b);
27)       fos.flush();
28)       System.out.println("Open emp.html to get Employees data");
29)       fos.close();
30)       con.close();
31)    }
32) }
```

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

```
1)  import java.sql.*;
2)  import java.util.*;
3)  public class SelectNonSelectDemo {
4)     public static void main(String[] args) throws Exception
5)     {
6)        String driver="oracle.jdbc.OracleDriver";
7)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
8)        String user="scott";
9)        String pwd="tiger";
10)       Class.forName(driver);// This step is optional
11)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
12)       Statement st = con.createStatement();
13)       Scanner sc = new Scanner(System.in);
14)       System.out.println("Enter the Query: ");
15)       String sqlQuery=sc.nextLine();
16)       boolean b = st.execute(sqlQuery);
17)       if(b== true)//select query
18)       {
19)          ResultSet rs =st.getResultSet();
20)          while(rs.next())
21)         {
22)             System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getDouble(3)+"\t"+rs.
    getString(4));
23)          }
24)       }
25)       else //non-select query
26)       {
27)          int rowCount=st.getUpdateCount();
28)          System.out.println("The number of records effected is:"+rowCount);
29)       }
30)       con.close();
31)    }
32) }
```

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:

```
1)  import java.sql.*;
2)  public class ExecuteMethodLoopHoles2T1
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        String driver="sun.jdbc.odbc.JdbcOdbcDriver";
7)        String jdbc_url="jdbc:odbc:demodsn";
8)        String user="scott";
9)        String pwd="tiger";
10)       Class.forName(driver);
11)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
12)       Statement st = con.createStatement();
13)    ResultSet rs=st.executeQuery("update employees set esal=7777 where ename='durga'");
14)       con.close();
15)    }
16) }
```

For Type-4 Driver:

```
1)  import java.sql.*;
2)  public class ExecuteMethodLoopHoles2T4
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        String driver="oracle.jdbc.OracleDriver";
7)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
8)        String user="scott";
9)        String pwd="tiger";
10)       Class.forName(driver);
11)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
12)       Statement st = con.createStatement();
13)    ResultSet rs=st.executeQuery("update employees set esal=7777 where ename='durga'");
14)       con.close();
15)    }
16) }
```

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:

```
1)  import java.sql.*;
2)  public class ExecuteMethodLoopHoles3T1
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        String driver="sun.jdbc.odbc.JdbcOdbcDriver";
7)        String jdbc_url="jdbc:odbc:demodsn";
8)        String user="scott";
9)        String pwd="tiger";
10)       Class.forName(driver);
11)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
12)       Statement st = con.createStatement();
13)       int rowCount=st.executeUpdate("select * from employees");
14)       System.out.println(rowCount);
15)       con.close();
16)    }
17) }
```

For Type-4 Driver:

```
1)  import java.sql.*;
2)  public class ExecuteMethodLoopHoles3T4
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        String driver="oracle.jdbc.OracleDriver";
7)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
8)        String user="scott";
9)        String pwd="tiger";
10)       Class.forName(driver);
11)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
12)       Statement st = con.createStatement();
13)       int rowCount=st.executeUpdate("select * from employees");
14)       System.out.println(rowCount);
15)       con.close();
16)    }
17) }
```

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:

```
1)  import java.sql.*;
2)  public class ExecuteMethodLoopHoles1T1
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        String driver="sun.jdbc.odbc.JdbcOdbcDriver";
7)        String jdbc_url="jdbc:odbc:demodsn";
8)        String user="scott";
9)        String pwd="tiger";
10)       Class.forName(driver);
11)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
12)       Statement st = con.createStatement();
13)       int updateCount=st.executeUpdate("create table emp1(eno number)");
14)       System.out.println(updateCount);//-1
15)       con.close();
16)    }
17) }
```

For Type-4 Driver:

```
1)  import java.sql.*;
2)  public class ExecuteMethodLoopHoles1T4
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        String driver="oracle.jdbc.OracleDriver";
7)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
8)        String user="scott";
9)        String pwd="tiger";
10)       Class.forName(driver);
11)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
12)       Statement st = con.createStatement();
13)       int updateCount=st.executeUpdate("create table emp2(eno number)");
14)       System.out.println(updateCount);//0
15)       con.close();
16)    }
17) }
```

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:

```
1)  m1()
2)  {
3)     try
4)     {
5)        m2();
6)     }
7)     catch(Exception e)
8)     {
9)     }
10) }
11)
12) m2() throws Exception
13) {
14)    ....
15)  }
```

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

```
1)  package com.durgasoft.jdbc;
2)  import java.sql.Connection;
3)  import java.sql.DriverManager;
4)  import java.sql.Statement;
5)  import java.sql.ResultSet;
6)  import java.sql.SQLException;
7)  /**
8)  * @ Author: Durga
9)  * @ Company: DURGASOFT
10) * @ see: www.durgasoft.com
11) */
12) public class JDBCCodingStandardsDemo1
13) {
14)    public static void main(String[] args)
15)    {
16)       try
17)       {
18)          Class.forName("oracle.jdbc.OracleDriver");
19)       }
20)       catch(ClassNotFoundException e)
21)       {
22)          e.printStackTrace();
23)       }
24)       Connection con=null;
25)       Statement st = null;
26)       ResultSet rs=null;
27)       try
28)       {
29)          con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","scott","
    tiger");
30)          st=con.createStatement();
31)          rs=st.executeQuery("select * from employees");
32)          while(rs.next())
33)          {
34)             System.out.println(rs.getInt(1)+".."+rs.getString(2)+"..."+rs.getDouble(3)+".."+rs.g
    etString(4));
35)          }
36)       }
37)       catch(SQLException e)
38)       {
39)          e.printStackTrace();
40)       }
41)       finally
42)       {
43)          try
44)          {
45)             if(rs!= null)
```

```
46)                 rs.close();
47)               if(st!= null)
48)                 st.close();
49)               if(con!= null)
50)                 con.close();
51)           }
52)           catch(SQLException e)
53)           {
54)              e.printStackTrace();
55)           }
56)       }
57)    }
58) }
```

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:

```
1)  package com.durgasoft.jdbc;
2)  import java.sql.Connection;
3)  import java.sql.DriverManager;
4)  import java.sql.Statement;
5)  import java.sql.ResultSet;
6)  import java.sql.SQLException;
7)  /**
8)  * @ Author: Durga
9)  * @ Company: DURGASOFT
10) * @ see: www.durgasoft.com
11) */
12) public class JDBCCodingStandardsDemo2
13) {
14)    public static void main(String[] args)
15)    {
16)       Connection con=null;
17)       Statement st = null;
18)       ResultSet rs=null;
19)       try
20)       {
21)          con=JdbcUtil.getOracleConnection();
22)          st=con.createStatement();
```

```
23)         rs=st.executeQuery("select * from employees");
24)         while(rs.next())
25)         {
26)            System.out.println(rs.getInt(1)+".."+rs.getString(2)+"..."+rs.getDouble(3)+".."+rs.g
    etString(4));
27)         }
28)      }
29)      catch(SQLException e)
30)      {
31)         e.printStackTrace();
32)      }
33)      finally
34)      {
35)         JdbcUtil.cleanup(con,st,rs);
36)      }
37)   }
38) }
```

JdbcUtil.java:

```
1)  package com.durgasoft.jdbc;
2)  import java.sql.Connection;
3)  import java.sql.DriverManager;
4)  import java.sql.Statement;
5)  import java.sql.ResultSet;
6)  import java.sql.SQLException;
7)  public class JdbcUtil
8)  {
9)     static
10)    {
11)       try
12)       {
13)          Class.forName("oracle.jdbc.OracleDriver");
14)       }
15)       catch(ClassNotFoundException e)
16)       {
17)          e.printStackTrace();
18)       }
19)    }
20)    public static Connection getOracleConnection()throws SQLException
21)    {
22)       Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE"
    ,"scott","tiger");
23)       return con;
24)    }
25)    public static void cleanup(Connection con,Statement st,ResultSet rs)
26)    {
27)       try
28)       {
```

```
29)         if(rs!= null)
30)             rs.close();
31)         if(st!= null)
32)             st.close();
33)         if(con!= null)
34)             con.close();
35)       }
36)       catch(SQLException e)
37)       {
38)          e.printStackTrace();
39)       }
40)    }
41) }
```

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

Working with MySQL Database

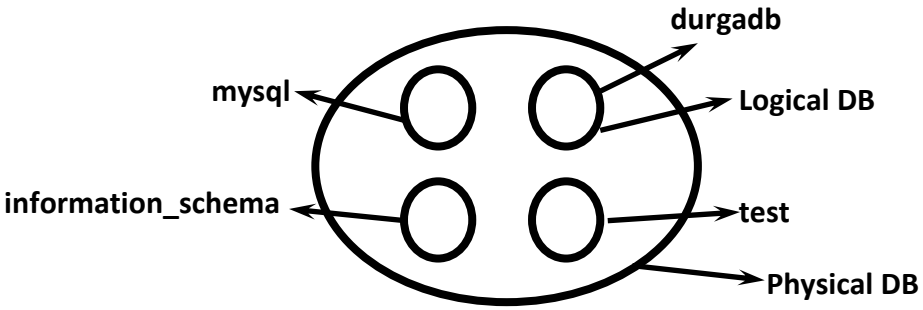
Current Version :5.7.14
Vendor: Sun Microsystems/Oracle Corporation
Open Source And Freeware
Default Port: 3306
Default User: root

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

```
1)  import java.sql.*;
2)  public class JdbcMySQLDemo
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Class.forName("com.mysql.jdbc.Driver");
7)        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/durgadb","root","root");
8)        Statement st = con.createStatement();
9)        ResultSet rs =st.executeQuery("select * from employees");
10)       while(rs.next())
11)       {
```

```
12)         System.out.println(rs.getInt(1)+".."+rs.getString(2)+".."+rs.getDouble(3)+".."+rs.get
    String(4));
13)       }
14)       con.close();
15)    }
16) }
```

Program to demonstrate copy data from Oracle to MySql database

```
1)  import java.sql.*;
2)  class OracleToMySQL
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        int count=0;
7)        Connection con1= DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:X
    E","scott","tiger");
8)        Connection con2 = DriverManager.getConnection("jdbc:mysql://localhost:3306/durga
    db","root","root");
9)        Statement st1= con1.createStatement();
10)       Statement st2= con2.createStatement();
11)       ResultSet rs=st1.executeQuery("select * from employees");
12)       while(rs.next())
13)       {
14)          count++;
15)          int eno=rs.getInt(1);
16)          String ename=rs.getString(2);
17)          double esal=rs.getDouble(3);
18)          String eaddr=rs.getString(4);
19)          String sqlQuery=String.format("insert into employees values(%d,'%s',%f,'%s')",eno,e
    name,esal,eaddr);
20)          st2.executeUpdate(sqlQuery);
21)       }
22)       System.out.println("Total Data copied from Oracle to MySQL and number of records:"
    +count);
23)       con1.close();
24)       con2.close();
25)    }
26) }
```

Life Cycle of SQL Query Execution

From Java application if we submit SQL Query by using Statement object execute method,

```
Statement st = con.createStatement();
ResultSet rs = st.executeQuery(sqlQuery);
```

Then database engine will perform the following sequence of activities

1. Compilation
2. Execution
3. Fetch Result

1. Compilation:

As the part of compilation, 0database engine will perform the following activities

A.Query Tokenization:

In this step total SQL Query will be divided into number of tokens and generate a Stream of tokens as output.

B.Query Parsing:

In this step, database engine will create parse tree (query tree) with stream of tokens. If the Query Tree is proper then there are no syntactical mistakes in that query.

If the query tree construction fails then it indicates that there are some syntactical errors present in SQL Query and SQLException will be raised.

C.Query Optimization:

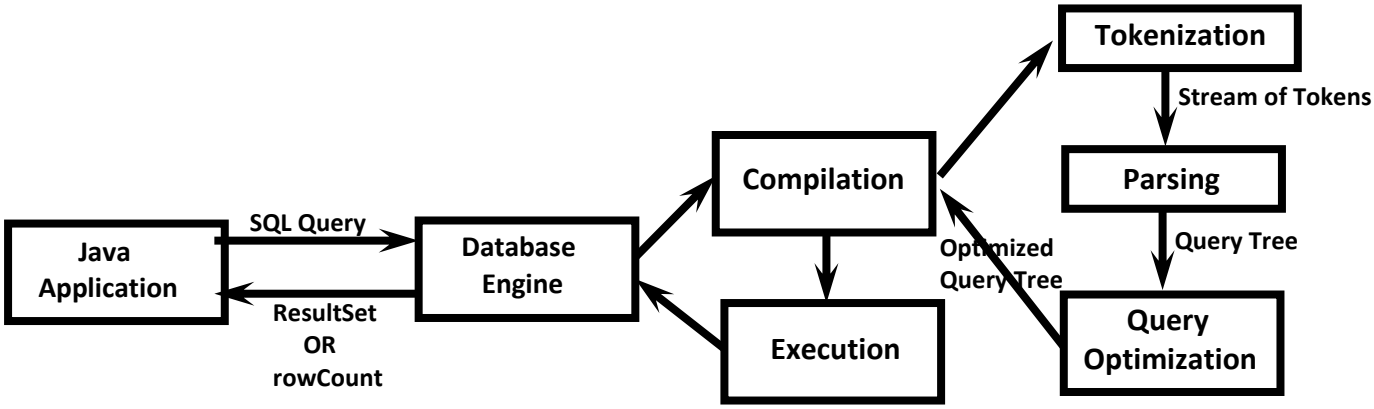
The main purpose of query optimization is to improve performance. In this step optimized query tree will be constructed.

2. Execution of sql Query:

Once compilation success then database engine will take that query tree as input and execute that query by using interpreter.

3. Fetch the Result:

Database engine will provide result of SQL Query either in the form of ResultSet (for select query) OR in the form of rowCount (for non-select query) to the Java application.

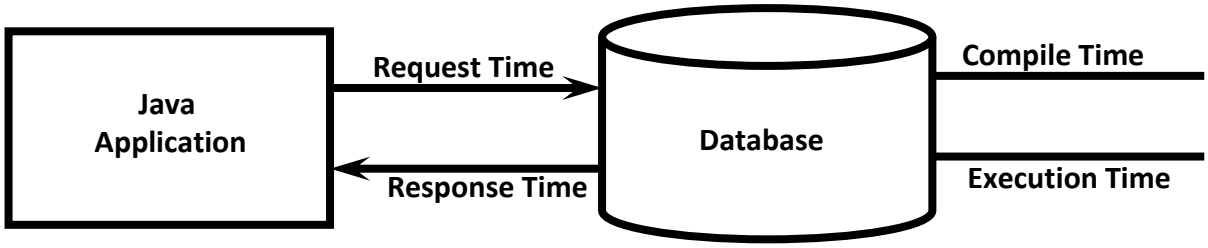

Java Application
SQL Query
ResultSet OR rowCount
Database Engine
Compilation
Execution
Tokenization
Stream of Tokens
Parsing
Query Tree
Query Optimization
Optimized Query Tree

PreparedStatement (I)

Need of PreparedStatement:

In the case of normal Statement, whenever we are executing SQL Query, every time compilation and execution will be happened at database side.

```
Statement st = con.createStatement();
ResultSet rs = st.executeQuery ("select * from employees");
```



Total Time per Query = Req.T+C.T+E.T+Resp.T
= 1 ms + 1 ms + 1 ms + 1 ms = 4ms

per 1000 Queries = 4 * 1000ms = 4000ms

Sometimes in our application,we required to execute same query multiple times with same or different input values.

Eg1:

In IRCTC application,it is common requirement to list out all possible trains between 2 places

```
select * from trains where source='XXX' and destination='YYY';
```

Query is same but source and destination places may be different. This query is required to execute lakhs of times per day.

Eg2:

In BookMyShow application, it is very common requirement to display theatre names where a particular movie running/playing in a particular city

```
select * from theatres where city='XXX' and movie='YYY';
```

In this case this query is required to execute lakhs of times per day. May be with different movie names and different locations.

For the above requirements if we use Statement object, then the query is required to compile and execute every time, which creates performance problems.

To overcome this problem, we should go for PreparedStatement.

The main advantage of PreparedStatement is the query will be compiled only once even though we are executing multiple times, so that overall performance of the application will be improved.

We can create PreparedStatement by using prepareStatement() method of Connection interface.

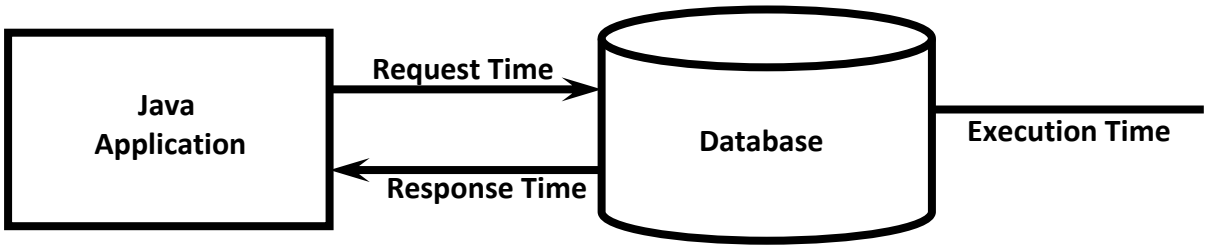
public PreparedStatement prepareStatement(String sqlQuery) throws SQLException

Eg: PreparedStatment pst=con.prepareStatement(sqlQuery);

At this line,sqlQuery will send to the database. Database engine will compile that query and stores in the database.
That pre compiled query will be returned to the java application in the form of PreparedStatement object.

Hence PreparedStatement represents "pre compiled sql query".

Whenever we call execute methods,database engine won't compile query once again and it will directly execute that query,so that overall performance will be improved.



Total Time per Query = Req.T+Resp.T+E.T
= 1 ms + 1 ms + 1 ms = 3ms

1000 Queries = 3000ms

Steps to develop JDBC Application by using PreparedStatement

1. Prepare SQLQuery either with parameters or without parameters.

Eg: `insert into employees values(100,'durga',1000,'hyd');`

```
insert into employees values(?, ?, ?, ?);
```

Positional Parameter OR Place Holder OR IN Parameter

2. Create PreparedStatement object with our sql query.

```
PreparedStatement pst = con.prepareStatement(sqlQuery);
```

At this line only query will be compiled.

3.If the query is parameterized query then we have to set input values to these parameters by using corresponding setter methods.

We have to consider these positional parameters from left to right and these are 1 index based. i.e index of first positional parameter is 1 but not zero.

```
pst.setInt(1,100);
pst.setString(2,"durga");
pst.setDouble(3,1000);
pst.setString(4,"Hyd");
```

Note:
Before executing the query, for every positional parameter we have to provide input values otherwise we will get SQLException

4. Execute SQL Query:

PreparedStatement is the child interface of Statement and hence all methods of Statement interface are bydefault available to the PreparedStatement.Hence we can use same methods to execute sql query.

```
executeQuery()
executeUpdate()
execute()
```

Note:

We can execute same parameterized query multiple times with different sets of input values. In this case query will be compiled only once and we can execute multiple times.

Q. Which of the following are valid sql statements?

1. delete from employees where ename=?
2. delete from employees ? ename=?
3. delete from ? where ename=?
4. delete ? employees where ename=?

Note:

We can use ? only in the place of input values and we cannot use in the place of sql keywords,table names and column names.

Static Query vs Dynamic Query:

The sql query without positional parameter(?) is called static query.

Eg: delete from employees where ename='durga'

The sql query with positional parameter(?) is called dynamic query.

Eg: select * from employees where esal>?

Program-1 to Demonstrate PreparedStatement:

```
1)  import java.sql.*;
2)  public class PreparedStatementDemo1
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        String driver="oracle.jdbc.OracleDriver";
7)        Class.forName(driver);
8)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
9)        String user="scott";
10)       String pwd="tiger";
11)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
12)       String sqlQuery ="delete from employees where ename=?";
13)
14)       PreparedStatement pst = con.prepareStatement(sqlQuery);
15)       pst.setString(1,"Mallika");
16)       int updateCount=pst.executeUpdate();
17)       System.out.println("The number of rows deleted :"+updateCount);
18)
19)       System.out.println("Reusing PreparedStatement to delete one more record...");
20)       pst.setString(1,"Durga");
21)       int updateCount1=pst.executeUpdate();
22)       System.out.println("The number of rows deleted :"+updateCount1);
23)       con.close();
```

```
24)    }
25)  }
```

Program-2 to Demonstrate PreparedStatement:

```
1)  import java.sql.*;
2)  import java.util.*;
3)  public class PreparedStatementDemo2
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        String driver="oracle.jdbc.OracleDriver";
8)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
9)        String user="scott";
10)       String pwd="tiger";
11)       Class.forName(driver);
12)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
13)       String sqlQuery="insert into employees values(?,?,?,?)";
14)       PreparedStatement pst = con.prepareStatement(sqlQuery);
15)
16)       Scanner sc = new Scanner(System.in);
17)       while(true)
18)       {
19)          System.out.println("Employee Number:");
20)          int eno=sc.nextInt();
21)          System.out.println("Employee Name:");
22)          String ename=sc.next();
23)          System.out.println("Employee Sal:");
24)          double esal=sc.nextDouble();
25)          System.out.println("Employee Address:");
26)          String eaddr=sc.next();
27)          pst.setInt(1,eno);
28)          pst.setString(2,ename);
29)          pst.setDouble(3,esal);
30)          pst.setString(4,eaddr);
31)          pst.executeUpdate();
32)          System.out.println("Record Inserted Successfully");
33)          System.out.println("Do U want to Insert one more record[Yes/No]:");
34)          String option = sc.next();
35)          if(option.equalsIgnoreCase("No"))
36)          {
37)             break;
38)          }
39)       }
40)       con.close();
41)    }
42) }
```

Advantages of PreparedStatement:

1. Performance will be improved when compared with simple Statement b'z query will be compiled only once.

2. Network traffic will be reduced between java application and database b'z we are not required to send query every time to the database.

3. We are not required to provide input values at the beginning and we can provide dynamically so that we can execute same query multiple times with different sets of values.

4. It allows to provide input values in java style and we are not required to convert into database specific format.

5.Best suitable to insert Date values

6. Best Sutitable to insert Large OBjects(CLOB,BLOB)

7. It prevents SQL Injection Attack.

Limitation of PreparedStatement:

We can use PreparedStatement for only one sql query (Like CDMA Phone), but we can use simple Statement to work with any number of queries (Like GSM Phone).

Eg:

```
Statement st = con.createStatement();
st.executeUpdate("insert into ...");
st.executeUpdate("update employees...");
st.executeUpdate("delete...");
```

Here We Are Using One Statement Object To Execute 3 Queries

```
PreparedStatement pst = con.prepareStatement("insert into employees..");
```

Here PreparedStatement object is associated with only insert query.

Note:

Simple Statement can be used only for static queries where as PreparedStatement can used for both static and dynamic queries.

Differences Between Statement And PreparedStatement

Statement	PreparedStatement
1) At the time of creating Statement Object, we are not required to provide any Query. Statement st = con.createStatement(); Hence Statement Object is not associated with any Query and we can use for multiple Queries.	1) At the time of creating PreparedStatement, we have to provide SQL Query compulsory and will send to the Database and will be compiled. PS pst = con.prepareStatement(query); Hence PS is associated with only one Query.
2) Whenever we are using execute Method, every time Query will be compiled and executed.	2) Whenever we are using execute Method, Query won't be compiled just will be executed.
3) Statement Object can work only for Static Queries.	3) PS Object can work for both Static and Dynamic Queries.
4) Relatively Performance is Low.	4) Relatively Performance is High.
5) Best choice if we want to work with multiple Queries.	5) Best choice if we want to work with only one Query but required to execute multiple times.
6) There may be a chance of SQL Injection Attack.	6) There is no chance of SQL Injection Attack.
7) Inserting Date and Large Objects (CLOB and BLOB) is difficult.	7) Inserting Date and Large Objects (CLOB and BLOB) is easy.

SQL Injection Attack

In the case of Simple Statement every time the query will send to the database with user provided input values.

Every time the query will be compiled and executed. Some times end user may provide special characters as the part user input,which may change behaviour of sql query.This is nothing but SQL Injection Attack,which causes security problems.

But in the case of PreparedStatement query will be compiled at the beginning only without considering end user's input. User provided data will be considered at the time of execution only. Hence as the part of user input,if he provides any special characters as the part of input,query behaviour wont be changed. Hence there is no chance of SQL Injection Attack in PreparedStatement.

Eg: `select count(*) from users where uname='"+uname+"' and upwd='"+upwd+"'"`

If the end user provides username as durga and pwd as java then the query will become

```
select count(*) from users where uname='durga' and upwd='java'
```

The query is meaningful and it is validating both username and pwd.
If the end user provides username as durga'-- and pwd as anushka then the query will become

```
select count(*) from users where uname='durga'--' and upwd='anushka'
```

It is not meaningful query b'z it is validating only username but not pwd. i.e with end user's provided input the query behaviour is changing,which is nothing but sql injection attack.

Note:

```
-- Single Line SQL Comment
/*
  Multi Line SQL
  Comment
*/
```

Eg 2:

```
select * from  users where uid=enduserprovidedinput
```

```
select * from users where uid=101;
```

returns record information where uid=101

select * from users where uid=101 OR 1=1;

1=1 is always true and hence it returns complete table information like username,pwe,uid etc...which may create security problems.

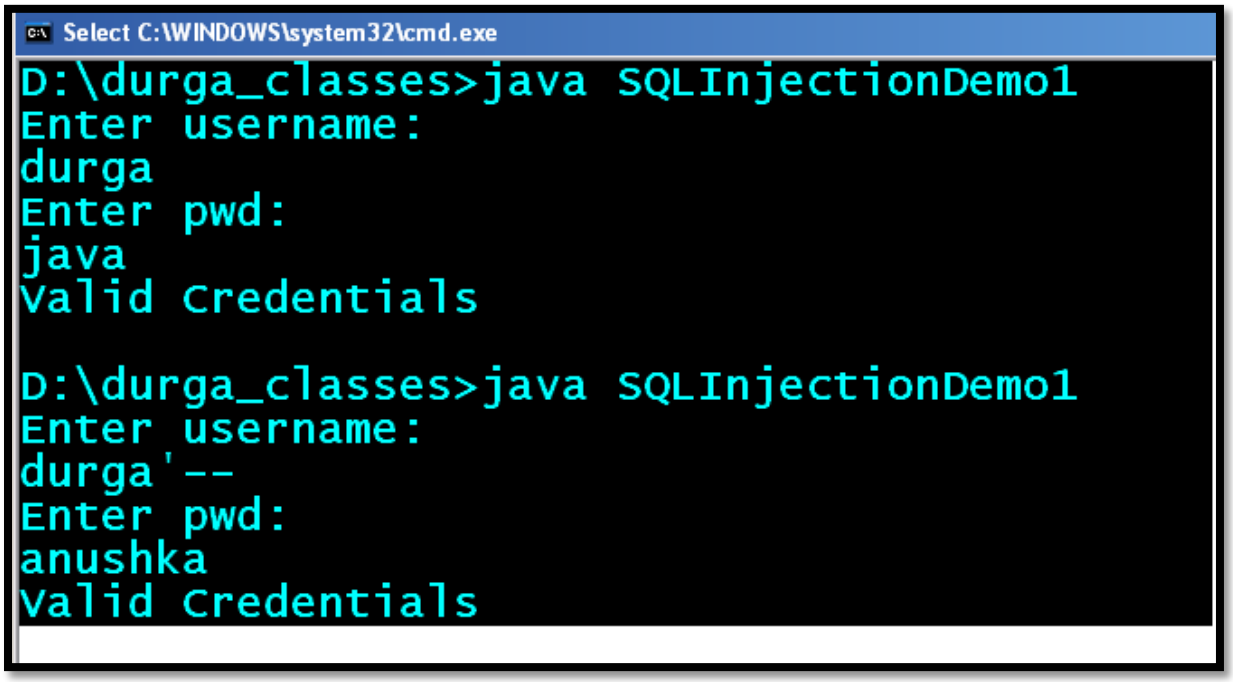
Program to Demonstrate SQL Injection Attack with Statement object:

SQL Script

```
1)  create table users(uname varchar2(20),upwd varchar2(20));
2)
3)  insert into users values('durga','java');
4)  insert into users values('ravi','testing');
```

SQLInjectionDemo1.java

```
1)  import java.sql.*;
2)  import java.util.*;
3)  public class SQLInjectionDemo1
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        String driver="oracle.jdbc.OracleDriver";
8)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
9)        String user="scott";
10)       String pwd="tiger";
11)       Class.forName(driver);
12)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
13)       Statement st = con.createStatement();
14)       Scanner sc = new Scanner(System.in);
15)       System.out.println("Enter username:");
16)       String uname=sc.next();
17)       System.out.println("Enter pwd:");
18)       String upwd=sc.next();
19)       String sqlQuery="select count(*) from users where uname='"+uname+"' and upwd='"+
    upwd+"'";
20)       ResultSet rs =st.executeQuery(sqlQuery);
21)       int c=0;
22)       if(rs.next())
23)       {
24)          c=rs.getInt(1);
25)       }
26)       if(c==0)
27)          System.out.println("Invalid Credentials");
28)       else
29)          System.out.println("Valid Credentials");
30)       con.close();
31)    }
32) }
```



Program to Demonstrate that there is no chance of SQL Injection Attack with PreparedStatement object:

```
1)  import java.sql.*;
2)  import java.util.*;
3)  public class SQLInjectionDemo2
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        String driver="oracle.jdbc.OracleDriver";
8)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
9)        String user="scott";
10)       String pwd="tiger";
11)       Class.forName(driver);
12)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
13)       Scanner sc = new Scanner(System.in);
14)       System.out.println("Enter username:");
15)       String uname=sc.next();
16)       System.out.println("Enter pwd:");
17)       String upwd=sc.next();
18)       String sqlQuery="select count(*) from users where uname=? and upwd=?";
19)       PreparedStatement ps = con.prepareStatement(sqlQuery);
20)       ps.setString(1,uname);
21)       ps.setString(2,upwd);
22)       ResultSet rs =ps.executeQuery();
23)       int c=0;
24)       if(rs.next())
```

```
25)      {
26)         c=rs.getInt(1);
27)      }
28)      if(c==0)
29)         System.out.println("Invalid Credentials");
30)      else
31)         System.out.println("Valid Credentials");
32)
33)      con.close();
34)   }
35) }
```

```
Select C:\WINDOWS\system32\cmd.exe
D:\durga_classes>java SQLInjectionDemo2
Enter username:
durga
Enter pwd:
java
Valid Credentials

D:\durga_classes>java SQLInjectionDemo2
Enter username:
durga'--
Enter pwd:
anushka
Invalid Credentials
```

Stored Procedures and CallableStatement

In our programming if any code repeatedly required,then we can define that code inside a method and we can call that method multiple times based on our requirement.

Hence method is the best reusable component in our programming.

Similarly in the database programming,if any group of sql statements is repeatedly required then we can define those sql statements in a single group and we can call that group repeatedly based on our requirement.

This group of sql statements that perform a particular task is nothing but Stored Procedure.Hence stored procedure is the best reusable component at database level.

Hence Stored Procedure is a group of sql statements that performs a particular task.

These procedures stored in database permanently for future purpose and hence the name stored procedure.

Usually stored procedures are created by Database Admin (DBA).

Every database has its own language to create Stored Procedures.

Oracle has ➔ PL/SQL
MySQL has ➔ Stored Procedure Language
Microsoft SQL Server has ➔ Transact SQL(TSQL

Similar to methods stored procedure has its own parameters. Stored Procedure has 3 Types of parameters.

1. IN parameters(to provide input values)
2. OUT parameters(to collect output values)
3. INOUT parameters(to provide input and to collect output)

Eg 1 :

Z:=X+Y;

X,Y are IN parameters and Z is OUT parameter

Eg 2:

X:=X+X;

X is INOUT parameter

Syntax for creating Stored Procedure (Oracle):

```
1)  create or replace procedure procedure1(X IN number, Y IN number,Z OUT number) as
2)  BEGIN
3)   z:=x+y;
4)  END;
```

Note:

SQL and PL/SQL are not case-sensitive languages. We can use lower case and upper case also.

After writing Stored Procedure, we have to compile for this we required to use "/" (forward slash)

/ ➔ For compilation

while compiling if any errors occurs,then we can check these errors by using the following command

```
SQL> show errors;
```

Once we created Stored Procedure and compiled successfully,we have to register OUT parameter to hold result of stored procedure.

```
SQL> variable sum number;  (declaring a variable)
```

We can execute with execute command as follows

```
SQL> execute procedure1(10,20,:sum);

SQL> print sum;
```

Eg 2:

```
1)  create or replace procedure procedure1(X IN number,Y OUT number) as
2)  BEGIN
3)  Y:= x*x;
4)  END;
5)  /
```

```
SQL> variable square number;
SQL> execute procedure1(10,:square);
SQL> print square;

  SQUARE
----------
   100
```

Eg3: Procedure To Print Employee Salary Based On Given Employee Number.

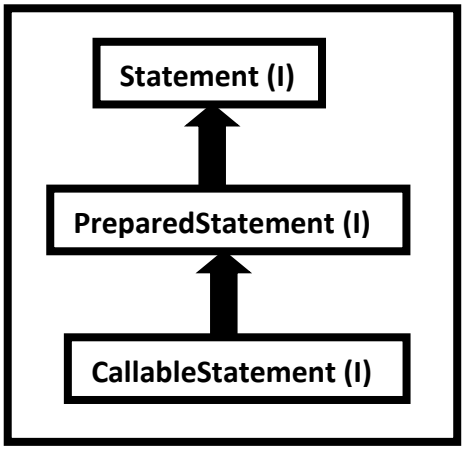
```
1)  create or replace procedure procedure2(eno1 IN number,esal1 OUT number) as
2)  BEGIN
3)   select esal into esal1 from employees where eno=eno1;
4)  END;
5)  /
```

```
SQL>variable salary number;
SQL>execute procedure2(100,:salary);
SQL>print salary;
```

Java Code for calling Stored Procedures:

If we want to call stored procedure from java application, then we should go for CallableStatement.

CallableStatement is an interface present in java.sql package and it is the child interface of PreparedStatement.

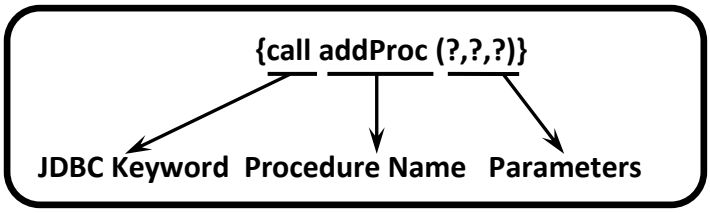


Driver software vendor is responsible to provide implementation for CallableStatement interface.

We can create CallableStatement object by using following method of Connection interface.

```
public CallableStatement prepareCall(String procedure_call) throws SQLException
```

Eg: CallableStatement cst=con.prepareCall("{call addProc(?,?,?)}");



Whenever JVM encounters this line,JVM will send call to database.Database engine will check whether the specified procedure is already available or not. If it is available then it returns CallableStatement object representing that procedure.

Mapping Java Types to database Types by using JDBC Types:

Java related data types and database related data types are not same.Some mechanism must be required to convert java types to database types and database types to java types. This mechanism is nothing but "JDBC Types",which are also known as "Bridge Types".

Java Data Type	JDBC Data Type	Oracle Data Type
int	Types.INTEGER	number
float	Types.FLOAT	number
String	Types.VARCHAR	varchar.varchar2
java.sql.Date	Types.DATE	date
:	:	:
:	:	:
:	:	:

Note: JDBC data types are defined as constants in "java.sql.Types" class.

Process to call Stored Procedure from java application by using CallableStatement:

1. Make sure Stored procedure available in the database

```
1)  create or replace procedure addProc(num1 IN number,num2 IN number,num3 OUT numbe
    r) as
2)  BEGIN
3)    num3 :=num1+num2;
4)  END;
5)  /
```

2. Create a CallableStatement with the procedure call.

```
CallableStatement cst = con.prepareCall("{call addProc(?,?,?)}");
```

3. Provide values for every IN parameter by using corresponding setter methods.

```
cst.setInt(1, 100);
cst.setInt(2, 200);
     ↙     ↘
  index    value
```

4. Register every OUT parameter with JDBC Types.

If stored procedure has OUT parameter then to hold that output value we should register every OUT parameter by using the following method.

```
public  void registerOutParameter (int index, int jdbcType)
```

Eg: cst.registerOutParameter(3,Types.INTEGER);

Note:

Before executing procedure call, all input parameters should set with values and every OUT parameter we have to register with jdbc type.

5. execute procedure call

```
cst.execute();
```

6. Get the result from OUT parameter by using the corresponding getXxx() method.

Eg: int result=cst.getInt(3);

Stored Procedures App1: JDBC Program to call StoredProcedure which can take two input numbers and produces the result.

Stored Procedure:

```
1)  create or replace procedure addProc(num1 IN number,num2 IN number,num3 OUT numbe
    r) as
2)  BEGIN
3)    num3 :=num1+num2;
4)  END;
5)  /
```

StoredProceduresDemo1.java

```
1)  import java.sql.*;
2)  class StoredProceduresDemo1
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","system","durga");
7)        CallableStatement cst=con.prepareCall("{call addProc(?,?,?)}");
8)        cst.setInt(1,100);
9)        cst.setInt(2,200);
10)       cst.registerOutParameter(3,Types.INTEGER);
11)       cst.execute();
12)       System.out.println("Result.."+cst.getInt(3));
13)       con.close();
14)    }
15) }
```

Stored Procedures App2: JDBC Program to call StoredProcedure which can take employee number as input and provides corresponding salary.

Stored Procedure:

```
1)  create or replace procedure getSal(id IN number,sal OUT number) as
2)  BEGIN
3)    select esal into sal from employees where eno=id;
4)  END;
5)  /
```

StoredProceduresDemo2.java

```
1)  import java.sql.*;
2)  class StoredProceduresDemo2
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","system","durga");
7)        CallableStatement cst=con.prepareCall("{call getSal(?,?)}");
8)        cst.setInt(1,100);
9)        cst.registerOutParameter(2,Types.FLOAT);
10)       cst.execute();
11)       System.out.println("Salary ..."+cst.getFloat(2));
12)       con.close();
13)    }
14) }
```

Stored Procedures App3: JDBC Program to call StoredProcedure which can take employee number as input and provides corresponding name and salary.

Stored Procedure:

```
1)  create or replace procedure getEmpInfo(id IN number,name OUT varchar2,sal OUT numbe
    r) as
2)  BEGIN
3)    select ename,esal into name,sal from employees where eno=id;
4)  END;
5)  /
```

StoredProceduresDemo3.java

```
1)  import java.sql.*;
2)  class StoredProceduresDemo3
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
```

```
6)      Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","system","durga");
7)      CallableStatement cst=con.prepareCall("{call getEmpInfo(?,?,?)}");
8)      cst.setInt(1,100);
9)      cst.registerOutParameter(2,Types.VARCHAR);
10)     cst.registerOutParameter(3,Types.FLOAT);
11)     cst.execute();
12)     System.out.println("Employee Name is :"+cst.getString(2));
13)     System.out.println("Employee Salary is :"+cst.getFloat(3));
14)     con.close();
15)   }
16) }
```

CURSORS

The results of SQL Queries will be stored in special memory area inside database software. This memory area is called Context Area.

To access Results of this context area,Some pointers are required and these pointers are nothing but cursors.

Hence the main objective of cursor is to access results of SQL Queries.

There are 2 types of cursors

1. Implicit cursors
2. Explicit cursors

1. Implicit cursors:

These cursors will be created automatically by database software to hold results whenever a particular type of sql query got executed.

2. Explicit Cursors:

These cursors will be created explicitly by the developer to hold results of particular sql queries.

Eg 1: SYS_REFCURSOR can be used to access result of select query i.e to access ResultSet.

Eg 2: %ROWCOUNT is an implicit cursor provided by Oracle to represent the number of rows effected b'z of insert,delete and update queries.

Eg 3: %FOUND is an implicit cursor provided by Oracle to represent whether any rows effected or not b'z of insert,delete and update operations(non-select query)

SYS_REFCURSOR VS OracleTypes.CURSOR:

To register SYS_REFCURSOR type OUT parameter JDBC does not contain any type.

To handle this situation,Oracle people provided

OracleTypes.CURSOR

class name **variable**

OracleTypes is a java class present in oracle.jdbc package and it is available as the part of ojdbc6.jar

If OUT parameter is SYS_REFCURSOR type,then we can get ResultSet by using getObject() method. But return type of getObject() method is Object and hence we should perform typecasting.

```
ResultSet rs = (ResultSet)cst.getObject(1);
```

Eg:

```
1)  create or replace prodecure getAllEmpInfo(emps OUT SYS_REFCURSOR) as
2)  BEGIN
3)   OPEN emps for
4)   select * from employees;
5)  end;
6)  /
```

```
1)  CallableStatement cst=con.prepareCall("{ call getAllEmpInfo(?)}");
2)  cst.registerOutParameter(1,OracleTypes.CURSOR);
3)  cst.execute();
4)  RS rs = (RS)cst.getObject(1);
5)  while(rs.next())
6)  {
7)  SOP(rs.getInt(1)+".."+rs...);
8)  }
```

Stored Procedures App4: JDBC Program to call StoredProcedure which returns all Employees info by using SYS_REFCURSOR

Stored Procedure:

```
1)  create or replace procedure getAllEmpInfo1(sal IN number,emps OUT SYS_REFCURSOR) as

2)  BEGIN
3)    open emps for
4)    select * from employees where esal<sal;
5)  END;
6)  /
```

StoredProceduresDemo4.java

```
1)  import java.sql.*;
2)  import oracle.jdbc.*;// for OracleTyes.CURSOR and it is present in ojdbc6.jar
3)  class StoredProceduresDemo4
4)  {
5)     public static void main(String[] args) throws Exception
```

```
6)    {
7)       Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
   ","system","durga");
8)       CallableStatement cst=con.prepareCall("{call getAllEmpInfo1(?,?)}");
9)       cst.setFloat(1,6000);
10)      cst.registerOutParameter(2,OracleTypes.CURSOR);
11)      cst.execute();
12)      ResultSet rs = (ResultSet)cst.getObject(2);
13)      boolean flag=false;
14)      System.out.println("ENO\tENAME\tESAL\tEADDR");
15)      System.out.println("----------------------");
16)      while(rs.next())
17)      {
18)         flag=true;
19)         System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
   tring(4));
20)      }
21)      if(flag== false)
22)      {
23)         System.out.println("No Recors Available");
24)      }
25)      con.close();
26)   }
27) }
```

Stored Procedures App5: JDBC Program to call StoredProcedure which returns all Employees info by using SYS_REFCURSOR based initial characters of the name

Stored Procedure:

```
1)  create or replace procedure getAllEmpInfo2(initchars IN varchar,emps OUT SYS_REFCURSO
    R) as
2)  BEGIN
3)    open emps for
4)    select * from employees where ename like initchars;
5)  END;
6)  /
```

StoredProceduresDemo5.java

```
1)  import java.sql.*;
2)  import java.util.*;
3)  import oracle.jdbc.*;// for OracleTyes.CURSOR and it is present in ojdbc6.jar
4)  class StoredProceduresDemo5
5)  {
6)     public static void main(String[] args) throws Exception
7)     {
8)        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
   ","system","durga");
```

```
9)      CallableStatement cst=con.prepareCall("{call getAllEmpInfo2(?,?)}");
10)     Scanner sc = new Scanner(System.in);
11)     System.out.println("Enter initial characters of the name");
12)     String initialchars=sc.next()+"%";
13)     cst.setString(1,initialchars);
14)     cst.registerOutParameter(2,OracleTypes.CURSOR);
15)     cst.execute();
16)     ResultSet rs = (ResultSet)cst.getObject(2);
17)     boolean flag= false;
18)     System.out.println("ENO\tENAME\tESAL\tEADDR");
19)     System.out.println("----------------------");
20)     while(rs.next())
21)     {
22)        flag=true;
23)        System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
    tring(4));
24)     }
25)     if(flag== false)
26)     {
27)        System.out.println("No Recors Available");
28)     }
29)     con.close();
30)   }
31) }
```

Functions

Functions are exactly same as procedures except that function has return statement directly.

Procedure can also returns values indirectly in the form of OUT parameters.

Usually we can use procedure to define business logic and we can use functions to perform some calculations like getAverage() , getMax() etc..

Syntax for functions:

```
1)  create or replace function getAvg(id1 IN number,id2 IN number)return number
2)  as
3)  sal1 number;
4)  sal2 number;
5)  BEGIN
6)    select esal into sal1 from employees where eno=id1;
7)    select esal into sal2 from employees where eno=id2;
8)    return (sal1+sal2)/2;
9)  END;
10) /
```

Function call can return some value.Hence the syntax of function call is

```
CS cst = con.prepareCall("{? = call getAvg(?,?)}");
```

return value of function call should be register as OUT parameter.

Stored Procedures App6: JDBC Program to call Function which returns average salary of given two employees

Stored Procedure

```
1)  create or replace function getAvg(id1 IN number,id2 IN number)return number
2)  as
3)  sal1 number;
4)  sal2 number;
5)  BEGIN
6)    select esal into sal1 from employees where eno=id1;
7)    select esal into sal2 from employees where eno=id2;
8)    return (sal1+sal2)/2;
9)  END;
10) /
```

StoredProceduresDemo6.java

```
1)  import java.sql.*;
2)  class StoredProceduresDemo6
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","system","durga");
7)        CallableStatement cst=con.prepareCall("{?=call getAvg(?,?)}");
8)        cst.setInt(2,100);
9)        cst.setInt(3,200);
10)       cst.registerOutParameter(1,Types.FLOAT);
11)       cst.execute();
12)       System.out.println("Salary ..."+cst.getFloat(1));
13)       con.close();
14)    }
15) }
```

Stored Procedures App7: JDBC Program to call function returns all employees information based on employee numbers

Stored Procedure

```
1)  create or replace function getAllEmpInfo4(no1 IN number,no2 IN number) return SYS_REF
    CURSOR as
2)  emps SYS_REFCURSOR;
3)  BEGIN
4)    open emps for
5)    select * from employees where eno>=no1 and eno<=no2;
6)    return emps;
7)  END;
8)  /
```

StoredProceduresDemo7.java

```
1)  import java.sql.*;
2)  import oracle.jdbc.*;// for OracleTyes.CURSOR and it is present in ojdbc6.jar
3)  class StoredProceduresDemo7
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","system","durga");
8)        CallableStatement cst=con.prepareCall("{?=call getAllEmpInfo4(?,?)}");
9)        cst.setInt(2,1000);
10)       cst.setInt(3,2000);
11)       cst.registerOutParameter(1,OracleTypes.CURSOR);
12)       cst.execute();
```

```
13)       ResultSet rs = (ResultSet)cst.getObject(1);
14)       boolean flag=false;
15)       System.out.println("ENO\tENAME\tESAL\tEADDR");
16)       System.out.println("----------------------");
17)       while(rs.next())
18)       {
19)          flag=true;
20)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
     tring(4));
21)       }
22)       if(flag== false)
23)       {
24)          System.out.println("No Recors Available");
25)       }
26)       con.close();
27)    }
28) }
```

Stored Procedures App8: JDBC Program to call function to Demonstrate SQL%ROWCOUNT implicit cursor

Stored Procedure

```
1)  create or replace function getDeletedEMPInfo(no1 IN number,count OUT number) return S
    YS_REFCURSOR as
2)  emps SYS_REFCURSOR;
3)  BEGIN
4)   open emps for
5)      select * from employees where eno=no1;
6)      delete from employees where eno=no1;
7)      count :=SQL%ROWCOUNT;
8)    return emps;
9)  END;
10) /
```

StoredProceduresDemo8.java

```
1)  import java.sql.*;
2)  import oracle.jdbc.*;// for OracleTyes.CURSOR and it is present in ojdbc6.jar
3)  class StoredProceduresDemo8
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","system","durga");
8)        CallableStatement cst=con.prepareCall("{?=call getDeletedEMPInfo(?,?)}");
9)        cst.setInt(2,100);
10)       cst.registerOutParameter(1,OracleTypes.CURSOR);
11)       cst.registerOutParameter(3,Types.INTEGER);
```

```
12)       cst.execute();
13)       ResultSet rs = (ResultSet)cst.getObject(1);
14)       System.out.println("ENO\tENAME\tESAL\tEADDR");
15)       System.out.println("----------------------");
16)       while(rs.next())
17)       {
18)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
     tring(4));
19)       }
20)       int count=cst.getInt(3);
21)       System.out.println("The number of rows deleted: "+count);
22)       con.close();
23)    }
24) }
```

Statement vs PreparedStatement vs CallableStatement:

1. We can use normal Statement to execute multiple queries.

```
st.executeQuery(query1)
st.executeQuery(query2)
st.executeUpdate(query2)
```

i.e if we want to work with multiple queries then we should go for Statement object.

2. If we want to work with only one query,but should be executed multiple times then we should go for PreparedStatement.

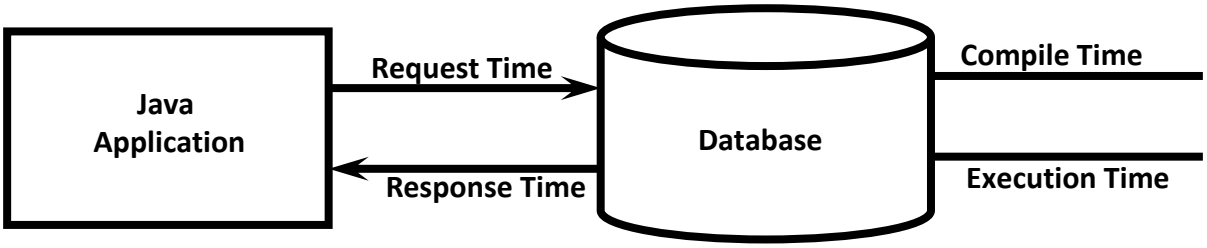
3. If we want to work with stored procedures and functions then we should go for CallableStatement.

Batch Updates

Need of Batch Updates:

When we submit multiple SQL Queries to the database one by one then lot of time will be wasted in request and response.

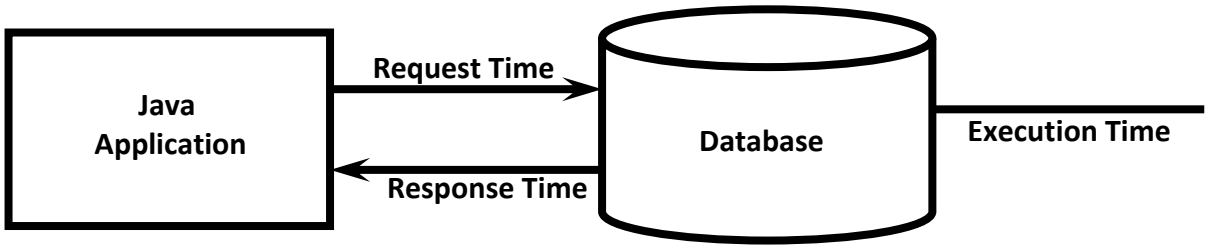
In the case of simple Statement:



Total Time per Query = Req.T+C.T+E.T+Resp.T
= 1 ms + 1 ms + 1 ms + 1 ms = 4ms

per 1000 Queries = 4 * 1000ms = 4000ms

In the case of PreparedStatement:

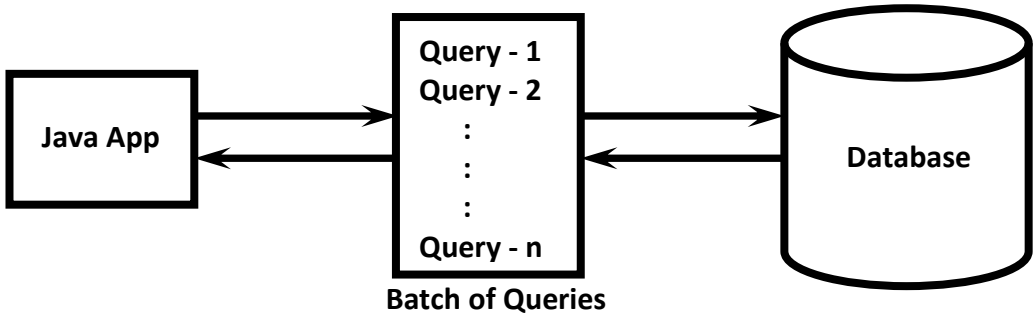


Total Time per Query = Req.T+Resp.T+E.T
= 1 ms + 1 ms + 1 ms = 3ms

1000 Queries = 3000ms

In the above 2 cases , we are trying to submit 1000 queries to the database one by one. For submitting 1000 queries we need to communicate with the database 1000 times. It increases network traffic Between java application and database and even creates performance problems also.

To overcome these problems, we should go for Batch updates. We can group all related SQL Queries into a single batch and we can send that batch at a time to the database.



With Simple Statement Batch Updates:

Per 1000 Queries = Req.Time+1000*C.T+1000*E.T+Resp.Time
= 1ms+1000*1ms+1000*1ms+1ms
= 2002ms

With PreparedStatement Batch Updates:

Per 1000 Queries = Req.Time+1000*E.T+Resp.Time
= 1ms+1000*1ms+1ms
= 1002ms

Hence the main advantages of Batch updates are
1. We can reduce network traffic
2. We can improve performance.

We can implement batch updates by using the following two methods

1. public void addBatch(String sqlQuery)
   To add query to batch

2. int[] executeBatch()
   to execute a batch of sql queries

We can implement batch updates either by simple Statement or by PreparedStatement

Program to Demonstrate Batch Updates with Simple Statement

```
1)  import java.sql.*;
2)  public class BatchUpdatesDemo1
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
     ","scott","tiger");
7)        Statement st = con.createStatement();
```

```
8)      //st.addBatch("select * from employees");
9)      st.addBatch("insert into employees values(600,'Mallika',6000,'Chennai')");
10)     st.addBatch("update employees set esal=esal+1000 where esal<4000");
11)     st.addBatch("delete from employees where esal>5000");
12)     int[] count=st.executeBatch();
13)     int updateCount=0;
14)     for(int x: count)
15)     {
16)        updateCount=updateCount+x;
17)     }
18)     System.out.println("The number of rows updated :"+updateCount);
19)     con.close();
20)   }
21) }
```

Program to Demonstrate Batch Updates with PreparedStatement

```
1)  import java.sql.*;
2)  import java.util.*;
3)  public class BatchUpdatesDemo2
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","scott","tiger");
8)        PreparedStatement pst = con.prepareStatement("insert  into employees values(?,?,?,?
    )");
9)        Scanner sc = new Scanner(System.in);
10)       while(true)
11)       {
12)          System.out.println("Employee Number:");
13)          int eno=sc.nextInt();
14)          System.out.println("Employee Name:");
15)          String ename=sc.next();
16)          System.out.println("Employee Sal:");
17)          double esal=sc.nextDouble();
18)          System.out.println("Employee Address:");
19)          String eaddr=sc.next();
20)          pst.setInt(1,eno);
21)          pst.setString(2,ename);
22)          pst.setDouble(3,esal);
23)          pst.setString(4,eaddr);
24)          pst.addBatch();
25)          System.out.println("Do U want to Insert one more record[Yes/No]:");
26)          String option = sc.next();
27)          if(option.equalsIgnoreCase("No"))
28)          {
29)             break;
30)          }
```

```
31)       }
32)       pst.executeBatch();
33)       System.out.println("Records inserted Successfully");
34)       con.close();
35)    }
36) }
```

Advantages of Batch Updates:

1. Network traffic will be reduced
2. Performance will be improved

Limitations of Batch updates:

1. We can use Batch Updates concept only for non-select queries. If we are trying to use for select queries then we will get RE saying BatchUpdateException.

2. In batch if one sql query execution fails then remaining sql queries wont be executed.

*****Q: In JDBC How Many Execute Methods Are Avaialble?

In total there are 4 methods are available

1. executeQuery() ➔ For select queries

2. executeUpdate() ➔ For non-select queries(insert|delete|update)

3. execute()
   ➔ For both select and non-select queries
   ➔ For calling Stored Procedures

4. executeBatch()➔ For Batch Updates

Handling Date Values For Database Operations

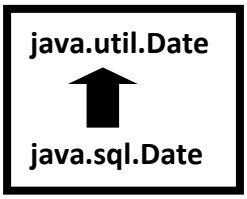
Sometimes as the part of programing requirement,we have to insert and retrieve Date like DOB,DOJ,DOM,DOP...wrt database.

It is not recommended to maintain date values in the form of String,b'z comparisons will become difficult.

In Java we have two Date classes

1. java.util.Date
2. java.sql.Date

java.sql.Date is the child class of java.util.Date.



java.sql.Date is specially designed class for handling Date values wrt database.

Otherthan database operations,if we want to represent Date in our java program then we should go for java.util.Date.

java.util.Date can represent both Date and Time where as java.sql.Date represents only Date but not time.

```
1)  class Test
2)  {
3)     public static void main(String[] args)
4)     {
5)        java.util.Date udate=new java.util.Date();
6)        System.out.println("util Date:"+udate);
7)              long l =udate.getTime();
8)        java.sql.Date sdate= new java.sql.Date(l);
9)        System.out.println("sql Date:"+sdate);
10)    }
11) }
```

util Date:Mon Mar 20 19:07:29 IST 2017
sql Date:2017-03-20

Differences between *java.util.Date* and *java.sql.Date*

java.util.Date	java.sql.Date
1) It is general Utility Class to handle Dates in our Java Program.	1) It is specially designed Class to handle Dates w.r.t DB Operations.
2) It represents both Data and Tieme.	2) It represents only Date but not Time.

Note: In sql package Time class is availble to represent Time values and TimeStamp class is available to represent both Date and Time.

Inserting Date Values into Database:

Various databases follow various styles to represent Date.

Eg:

Oracle: dd-MMM-yy 28-May-90
MySQL: yyyy-mm-dd 1990-05-28

If we use simple Statement object to insert Date values then we should provide Date value in the database supported format,which is difficult to the programmer.

If we use PreparedStatement,then we are not required to worry about database supported form, just we have to call
pst.setDate (2, java.sql.Date);

This method internally converts date value into the database supported format.

Hence it is highly recommendedto use PreparedStatement to insert Date values into database.

Steps to insert Date value into Database:

DB: create table users(name varchar2(10),dop date);

1. Read Date from the end user(in String form)

```
System.out.println("Enter DOP(dd-mm-yyyy):");
String dop=sc.next();
```

2. Convert date from String form to java.util.Date form by using SimpleDateFormat object.

```
SDF sdf= new SDF("dd-MM-yyyy");
java.util.Date udate=sdf.parse(dop);
```

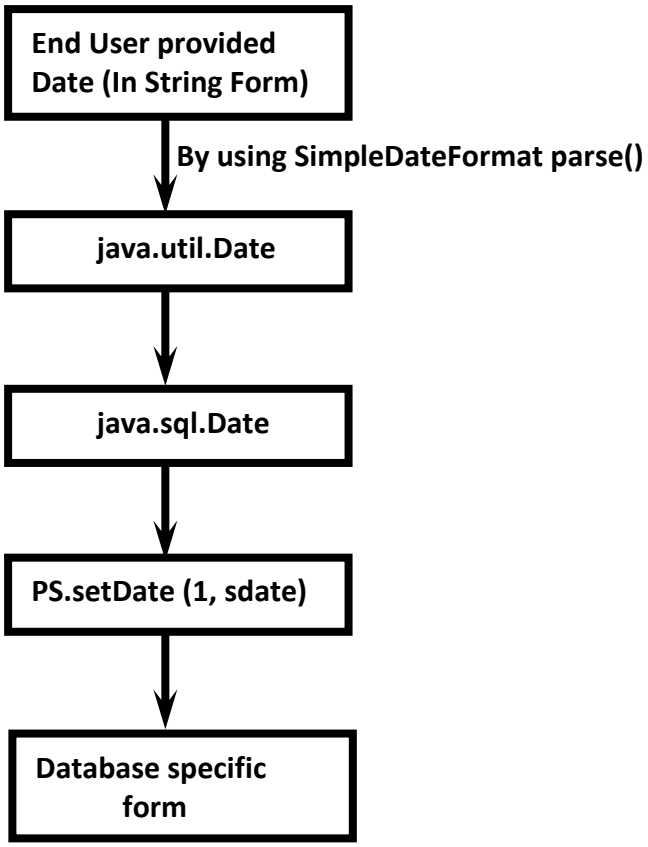
3. convert date from java.util.Date to java.sql.Date

```
long l = udate.getTime();
```

java.sql.Date sdate=new java.sql.Date(l);

4. set sdate to query

pst.setDate(2,sdate);



Program To Demonstrate Inserting Date Values Into Database:

DB: create table users(name varchar2(10),dop date);

DateInsertDemo.java

```
1)  import java.sql.*;
2)  import java.util.*;
3)  import java.text.*;
4)  public class DateInsertDemo
5)  {
6)     public static void main(String[] args) throws Exception
7)     {
8)        String driver="oracle.jdbc.OracleDriver";
9)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
10)       String user="scott";
11)       String pwd="tiger";
12)       Class.forName(driver);
```

```
13)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
14)       Scanner sc = new Scanner(System.in);
15)       System.out.println("Enter Person Name:");
16)       String uname=sc.next();
17)       System.out.println("Enter DOP(dd-mm-yyyy):");
18)       String dop=sc.next();
19)
20)       SimpleDateFormat sdf= new SimpleDateFormat("dd-MM-yyyy");
21)       java.util.Date udate=sdf.parse(dop);
22)       long l = udate.getTime();
23)       java.sql.Date sdate= new java.sql.Date(l);
24)       String sqlQuery="insert into users values(?,?)";
25)       PreparedStatement ps = con.prepareStatement(sqlQuery);
26)       ps.setString(1,uname);
27)       ps.setDate(2,sdate);
28)       int  rc =ps.executeUpdate();
29)       if(rc==0)
30)          System.out.println("Record Not inserted");
31)       else
32)          System.out.println("Record  inserted");
33)
34)       con.close();
35)    }
36) }
```

***Note: If end user provides Date in the form of "yyyy-MM-dd" then we can convert directly that String into java.sql.Date form as follows...

```
String s = "1980-05-27";
java.sql.Date sdate=java.sql.Date.valueOf(s);
```

Program To Demonstrate Inserting Date Values Into Database:

DB: `create table users(name varchar2(10),dop date);`

DateInsertDemo1.java

```
1)  import java.sql.*;
2)  import java.util.*;
3)  import java.text.*;
4)  public class DateInsertDemo1
5)  {
6)     public static void main(String[] args) throws Exception
7)     {
8)        String driver="oracle.jdbc.OracleDriver";
9)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
10)       String user="scott";
11)       String pwd="tiger";
```

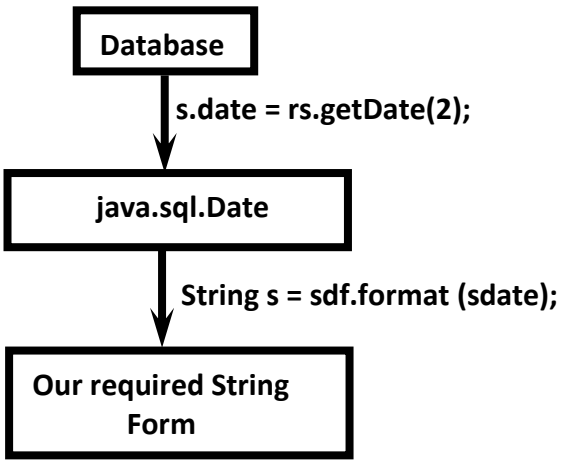
```
12)       Class.forName(driver);
13)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
14)       Scanner sc = new Scanner(System.in);
15)       System.out.println("Enter Person Name:");
16)       String uname=sc.next();
17)       System.out.println("Enter DOP(yyyy-MM-dd):");
18)       String dop=sc.next();
19)
20)       java.sql.Date sdate=java.sql.Date.valueOf(dop);
21)       String sqlQuery="insert into users values(?,?)";
22)       PreparedStatement ps = con.prepareStatement(sqlQuery);
23)       ps.setString(1,uname);
24)       ps.setDate(2,sdate);
25)       int  rc =ps.executeUpdate();
26)       if(rc==0)
27)          System.out.println("Record Not inserted");
28)       else
29)          System.out.println("Record  inserted");
30)
31)       con.close();
32)    }
33) }
```

Retrieving Date values from the database:

For this we can use either simple Statement or PreparedStatement.

The retrieved Date values are Stored in ResultSet in the form of "java.sql.Date" and we can get this value by using getDate() method.

Once we got java.sql.Date object,we can format into our required form by using SimpleDateFormat object.



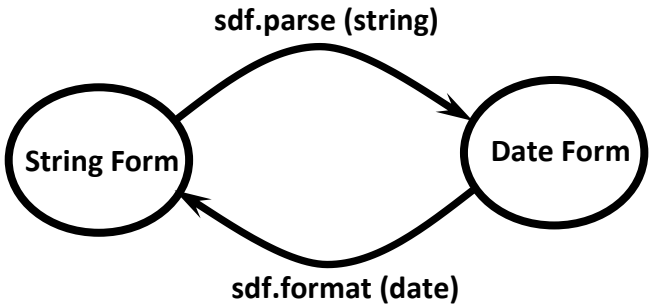
Program To Retrieve Date Values From The Database:

```
1)  import java.sql.*;
2)  import java.text.*;
3)  public class DateRetriveDemo
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        String driver="oracle.jdbc.OracleDriver";
8)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
9)        String user="scott";
10)       String pwd="tiger";
11)       Class.forName(driver);
12)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
13)       PreparedStatement ps = con.prepareStatement("select * from users6");
14)       ResultSet rs =ps.executeQuery();
15)       SimpleDateFormat sdf=new SimpleDateFormat("dd-MMM-yyyy");
16)       while(rs.next())
17)       {
18)          String name=rs.getString(1);
19)          java.sql.Date sdate=rs.getDate(2);
20)          String s = sdf.format(sdate);
21)          System.out.println(name+"..."+s);
22)       }
23)       con.close();
24)    }
25) }
```

FAQs:

1. In Java how many Date classes are available?
2. What is the difference Between java.util.Date and java.sql.Date?
3. What is the relation Between java.util.Date and java.sql.Date?
4. How to perform the following conversions?
   1. java.util.Date to java.sql.Date
   2. String to Date
   3. Date to String

Note: SimpleDateFormat class present in java.text package.



DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Working with Large Objects (BLOB And CLOB)

Sometimes as the part of programming requirement,we have to insert and retrieve large files like images,video files,audio files,resume etc wrt database.

Eg:
upload image in matrinomial web sites
upload resume in job related web sites

To store and retrieve large information we should go for Large Objects(LOBs).

There are 2 types of Large Objects.

1. Binary Large Object (BLOB)
2. Character Large Object (CLOB)

1) Binary Large Object (BLOB)

A BLOB is a collection of binary data stored as a single entity in the database.

BLOB type objects can be images,video files,audio files etc..

BLOB datatype can store maximum of "4GB" binary data.

2) CLOB (Character Large Objects):

A CLOB is a collection of Character data stored as a single entity in the database.

CLOB can be used to store large text documents(may plain text or xml documents)

CLOB Type can store maximum of 4GB data.

Eg: hydhistory.txt

Steps to insert BLOB type into database:

1. create a table in the database which can accept BLOB type data.

```
create table persons(name varchar2(10),image BLOB);
```

2. Represent image file in the form of Java File object.

```
File f = new File("katrina.jpg");
```

3. Create FileInputStream to read binary data represented by image file

```
FileInputStream fis = new FileInputStream(f);
```

4. Create PreparedStatement with insert query.

```
PreparedStatement pst = con.prepareStatement("insert into persons values(?,?)");
```

5. Set values to positional parameters.

```
pst.setString(1,"katrina");
```

To set values to BLOB datatype, we can use the following method: setBinaryStream()

```
public void setBinaryStream(int index,InputStream is)
public void setBinaryStream(int index,InputStream is,int length)
public void setBinaryStream(int index,InputStream is,long length)
```

Eg:

pst.setBinaryStream(2,fis); ➔ Oracle 11g
pst.setBinaryStream(2,fis,(int)f.length()); ➔ Oracle 10g

6. execute sql query

```
pst.executeUpdate();
```

Program to Demonstrate insert BLOB type into database:

DB: `create table persons(name varchar2(10),image BLOB);`

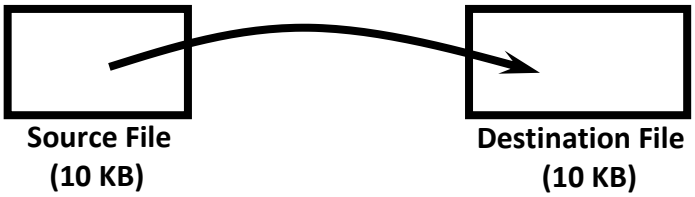
BLOBDemo1.java

```
1)  import java.sql.*;
2)  import java.io.*;
3)  public class BLOBDemo1
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        String driver="oracle.jdbc.OracleDriver";
8)        Class.forName(driver);
9)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
10)       String user="scott";
11)       String pwd="tiger";
12)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
13)       String sqlQuery="insert into persons values(?,?)";
14)       PreparedStatement ps = con.prepareStatement(sqlQuery);
15)       ps.setString(1,"Katrina");
16)       File f = new File("katrina.jpg");
17)       FileInputStream fis = new FileInputStream(f);
18)       ps.setBinaryStream(2,fis);
19)       System.out.println("inserting image from :"+f.getAbsolutePath());
20)       int updateCount=ps.executeUpdate();
21)       if(updateCount==1)
```

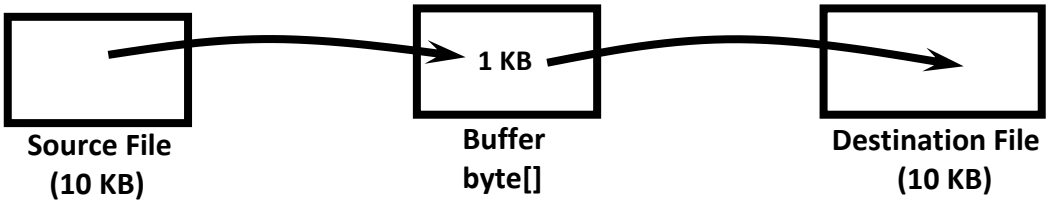
```
22)      {
23)         System.out.println("Record Inserted");
24)      }
25)      else
26)      {
27)         System.out.println("Record Not Inserted");
28)      }
29)
30)   }
31) }
```

Retrieving BLOB Type from Database:

We can use either simple Statement or PreparedStatement.

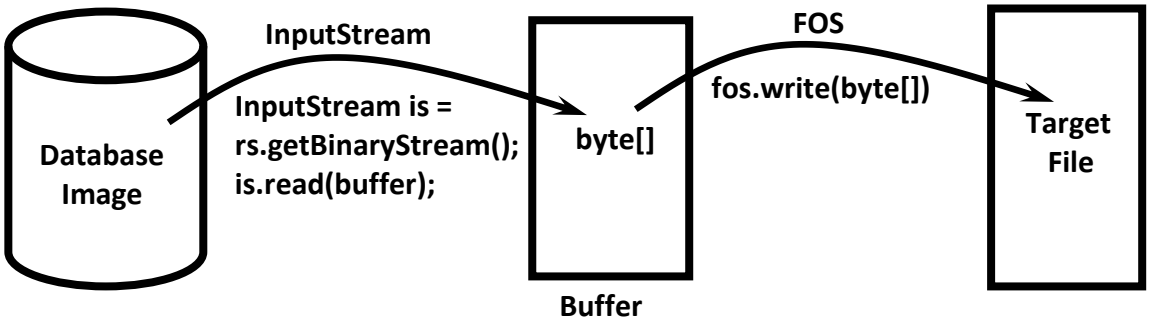


Without buffering 10 * 1024 read & write Operations are required



Because of Buffer we have to perform only 10 Read Operations & 10 Write Operations

Steps to Retrieve BLOB type from Database



1. Prepare ResultSet object with BLOB type

```
RS rs = st.executeQuery("select * from persons");
```

2. Read Normal data from ResultSet

```
String name=rs.getString(1);
```

3. Get InputStream to read binary data from ResultSet

```
InputStream is = rs.getBinaryStream(2);
```

4. Prepare target resource to hold BLOB data by using FileOutputStream

```
FOS fos = new FOS("katrina_new.jpg");
```

5. Read Binary Data from InputStream and write that Binary data to output Stream.

Without Buffer

```
int i = is. read();
while (i != -1)
{
      fos.write(i);
      i = is.read();
}
```

With Buffer

```
byte[] buffer = new byte[1024];
while (is.read(buffer)>0)
{
      fos.write(buffer);
}
```

Program to to Retrieve BLOB type from Database:

```
1)  import java.sql.*;
2)  import java.io.*;
3)  public class BLOBDemo2
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        String driver="oracle.jdbc.OracleDriver";
```

```
8)      String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
9)      String user="scott";
10)     String pwd="tiger";
11)     Class.forName(driver);
12)     Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
13)     PreparedStatement ps = con.prepareStatement("select * from persons");
14)     ResultSet rs =ps.executeQuery();
15)     FileOutputStream os = new FileOutputStream("katrina_sat.jpeg");
16)     if(rs.next())
17)     {
18)        String name=rs.getString(1);
19)        InputStream is = rs.getBinaryStream(2);
20)        byte[] buffer = new byte[2048];
21)        while(is.read(buffer)>0)
22)        {
23)           os.write(buffer);
24)        }
25)        os.flush();
26)        System.out.println("image is available in :katrina_sat.jpeg");
27)     }
28)     con.close();
29)  }
30) }
```

CLOB (Character Large Objects):

A CLOB is a collection of Character data stored as a single entity in the database.

CLOB can be used to store large text documents(may plain text or xml documents)

CLOB Type can store maximum of 4GB data.

Eg: hydhistory.txt

Steps to insert CLOB type file in the database:

All steps are exactly same as BLOB, except the following differences

1. Instead of FileInputStream, we have to take FileReader.

2. Instead of setBinaryStream() method we have to use setCharacterStream() method.

```
public void setCharacterStream(int index,Reader r) throws SQLException
public void setCharacterStream(int index,Reader r,int length) throws SQLException
public void setCharacterStream(int index,Reader r,long length) throws SQLException
```

Program to insert CLOB type file in the database:

DB: create table cities(name varchar2(10),history CLOB);

CLOBDemo1.java

```
1)  import java.sql.*;
2)  import java.io.*;
3)  public class CLOBDemo1
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        String driver="oracle.jdbc.OracleDriver";
8)        Class.forName(driver);
9)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
10)       String user="scott";
11)       String pwd="tiger";
12)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
13)       String sqlQuery="insert into cities values(?,?)";
14)       PreparedStatement ps = con.prepareStatement(sqlQuery);
15)            ps.setString(1,"Hyderabad");
16)       File f = new File("hyd_history.txt");
17)       FileReader fr = new FileReader(f);
18)       ps.setCharacterStream(2,fr);
19)       System.out.println("file is inserting from :"+f.getAbsolutePath());
20)       int updateCount=ps.executeUpdate();
21)       if(updateCount==1)
22)       {
23)          System.out.println("Record Inserted");
24)       }
25)       else
26)       {
27)          System.out.println("Record Not Inserted");
28)       }
29)
30)    }
31) }
```

Retrieving CLOB Type from Database:

All steps are exactly same as BLOB, except the following differences..

1. Instead of using FileOutputStream,we have to use FileWriter

2.Instead of using getBinaryStream() method we have to use getCharacterStream() method

Program For Retrieving CLOB Type from Database:

```
1)  import java.sql.*;
2)  import java.io.*;
3)  public class CLOBDemo2
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        String driver="oracle.jdbc.OracleDriver";
8)        String jdbc_url="jdbc:oracle:thin:@localhost:1521:XE";
9)        String user="scott";
10)       String pwd="tiger";
11)       Class.forName(driver);
12)       Connection con = DriverManager.getConnection(jdbc_url,user,pwd);
13)       PreparedStatement ps = con.prepareStatement("select * from cities");
14)       ResultSet rs =ps.executeQuery();
15)       FileWriter fw = new FileWriter("output_sat.txt");
16)       if(rs.next())
17)       {
18)          String name=rs.getString(1);
19)          Reader r = rs.getCharacterStream(2);
20)          /*char[] buffer = new char[1024];
21)          while(r.read(buffer)>0)
22)          {
23)             fw.write(buffer);
24)          }*/
25)          int i=r.read();
26)          while(i != -1)
27)          {
28)             fw.write(i);
29)             i = r.read();
30)          }
31)          fw.flush();
32)          System.out.println("Retrieved Successfully file :output_sat.txt");
33)       }
34)       con.close();
35)    }
36) }
```

Q. What is the difference between BLOB and CLOB?

We can use BLOB Type to represent binary information like images, video files, audio files etc

Where as we can use CLOB Type to represent Character data like text file, xml file etc...

Assignment for Inserting and Retrieving Date, BLOB and CLOB type data:

Create a table named with jobseeker and insert data and retrieve data

```
jobseeker
(name varchar2(20),dob Date,image BLOB,resume CLOB);
name="durga";
dob="28-05-1968";
image="durga.jpg";
resume="resume.txt";

String ➔ udate ➔ sdate
SDF sdf= new SDF("dd-MM-yyyy");
java.util.Date udate= sdf.parse(dob);
long l = udate.getTime();
java.sql.Date sdate=new java.sql.Date(l);
FIS fis = new FIS("durga.jpg");
FR fr= new FR("resume.txt");

PS pst=con.pS("insert into jobseeker values(?,?,?,?)");
pst.setString(1,name);
pst.setDate(2,sdate);
pst.setBinaryStream(3,fis);
pst.setCharacterStream(4,fr);
pst.executeUpdate();
================================
FOS fos= new FOS("updatedimage.jpg");
PW pw= new PW("updatedresume.txt");
SDF sdf= new SDF("dd-MM-yyyy");
PS pst=con.PS("select * from jobseeker");
RS rs = pst.executeQuery();
```

```
1)  if(rs.next())
2)  {
3)     //reading name
4)     String name=rs.getString(1);
5)     //reading dob
6)     java.sql.Date sdate=rs.getDate(2);
7)     String dob=sdf.format(sdate);
8)     //reading BLOB(image)
9)     InputStream is = rs.getBinaryStream(3);
10)    byte[] b = new byte[1024];
11)    while(is.read(b)>0)
12)    {
13)       fos.write(b);
14)    }
15)    fos.flush();
16)    //reading CLOB(txt file)
```

```
17)    Reader r = rs.getCharacterStream(4);
18)    char[] ch = new char[1024];
19)    while(r.read(ch)>0)
20)    {
21)       pw.write(ch);
22)    }
23)    pw.flush();
24) }
```

Connection Pooling

If we required to communicate with database multiple times then it is not recommended to create separate Connection object every time, b'z creating and destroying Connection object every time creates performance problems.

To overcome this problem, we should go for Connection Pool.

Connection Pool is a pool of already created Connection objects which are ready to use.

If we want to communicate with database then we request Connection pool to provide Connection. Once we got the Connection, by using that we can communicates with database. After completing our work, we can return Connection to the pool instead of destroying.

Hence the main advantage of Connection Pool is we can reuse same Connection object multiple times, so that overall performance of application will be improved.

Process to implement Connection Pooling:

1. Creation of DataSource object

DataSource is responsible to manage connections in Connection Pool.

DataSource is an interface present in javax.sql package.

Driver Software vendor is responsible to provide implementation.

Oracle people provided implementation class name is :
OracleConnectionPoolDataSource.

This class present inside oracle.jdbc.pool package and it is the part of ojdbc6.jar.

```
OracleConnectionPoolDataSource  ds= new OracleConnectionPoolDataSource();
```

2. Set required JDBC Properties to the DataSource object:

```
ds.setURL("jdbc:oracle:thin:@localhost:1521:XE");
ds.setUser("scott");
ds.setPassword("tiger");
```

3. Get Connection from DataSource object:

```
Connection con = ds.getConnection();
```
Once we got Connection object then remaining process is as usual.

Program to Demonstrate Connection Pooling for Oracle Database:

```
1)  import java.sql.*;
2)  import javax.sql.*;
3)  import oracle.jdbc.pool.*;// ojdbc6.jar
4)  class ConnectionPoolDemoOracle
5)  {
6)     public static void main(String[] args) throws Exception
7)     {
8)        OracleConnectionPoolDataSource ds = new OracleConnectionPoolDataSource();
9)        ds.setURL("jdbc:oracle:thin:@localhost:1521:XE");
10)       ds.setUser("scott");
11)       ds.setPassword("tiger");
12)       Connection con=ds.getConnection();
13)       Statement st =con.createStatement();
14)       ResultSet rs=st.executeQuery("select * from employees");
15)       System.out.println("ENO\tENAME\tESAL\tEADDR");
16)       System.out.println("--------------------------");
17)       while(rs.next())
18)       {
19)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
    tring(4));
20)       }
21)       con.close();
22)    }
23) }
```

Program to Demonstrate Connection Pooling for MySQL Database:

```
1)  import java.sql.*;
2)  import javax.sql.*;
3)  import com.mysql.jdbc.jdbc2.optional.*;
4)  class ConnectionPoolDemoMySql
5)  {
6)     public static void main(String[] args) throws Exception
7)     {
8)        MysqlConnectionPoolDataSource ds = new MysqlConnectionPoolDataSource();
9)        ds.setURL("jdbc:mysql://localhost:3306/durgadb");
10)       ds.setUser("root");
11)       ds.setPassword("root");
12)       Connection con=ds.getConnection();
13)       Statement st =con.createStatement();
14)       ResultSet rs=st.executeQuery("select * from employees");
15)       System.out.println("ENO\tENAME\tESAL\tEADDR");
16)       System.out.println("--------------------------");
17)       while(rs.next())
18)       {
```

```
19)         System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
    tring(4));
20)       }
21)       con.close();
22)    }
23) }
```

Note: This way of implementing Connection Pool is useful for Standalone applications. In the case of web and enterprise applications, we have to use server level connection pooling. Every web and application server can provide support for Connection Pooling.

Q. What is the difference Between getting Connection object by using DriverManager and DataSource object?

In the case of DriverManager.getConnection(), always a new Connection object will be created and returned.

But in the case of DataSourceObject.getConnection(), a new Connection object won't be created and existing Connection object will be returned from Connection Pool.

Properties

In Java Program if anything which changes frequently(like jdbc url, username, pwd etc)is not recommended to hard code in our program.

The problem in this approach is if there is any change in java program,to reflect that change we have to recompile,rebuild and redeploy total application and even some times server restart also required,which creates a big business impact to the client.

To overcome this problem, we should go for Properties file. The variable things we have to configure in Properties file and we have to read these properties from java program.

The main advantage of this approach is if there is any change in Properties file and to reflect that change just redeployment is enough, which won't create any business impact to the client.

Program to Demonstrate use of Properties file:

db.properties:

```
url= jdbc:oracle:thin:@localhost:1521:XE
user= scott
pwd= tiger
```

JdbcPropertiesDemo.java:

```
1)  import java.sql.*;
2)  import java.util.*;
3)  import java.io.*;
4)  class JdbcPropertiesDemo
5)  {
6)     public static void main(String[] args) throws Exception
7)     {
8)        Properties p = new Properties();
9)        FileInputStream fis = new FileInputStream("db.properties");
10)       p.load(fis);// to load all properties from properties file into java Properties object
11)       String url=p.getProperty("url");
12)       String user=p.getProperty("user");
13)       String pwd=p.getProperty("pwd");
14)       Connection con=DriverManager.getConnection(url,user,pwd);
15)       Statement st =con.createStatement();
16)       ResultSet rs=st.executeQuery("select * from employees");
17)       System.out.println("ENO\tENAME\tESAL\tEADDR");
18)       System.out.println("--------------------------");
```

```
19)       while(rs.next())
20)       {
21)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
     tring(4));
22)       }
23)       con.close();
24)    }
25) }
```

If we change the properties in properties file for mysql database then the program will fetch data from mysql database.

db.properties:

```
url= jdbc:mysql://localhost:3306/durgadb
user= root
pwd= root
```

Program to Demonstrate use of Properties file:

db1.properties:

```
user=scott
password=tiger
```

JdbcPropertiesDemo1.java:

```
1)  import java.sql.*;
2)  import java.util.*;
3)  import java.io.*;
4)  class JdbcPropertiesDemo1
5)  {
6)     public static void main(String[] args) throws Exception
7)     {
8)        Properties p = new Properties();
9)        FileInputStream fis = new FileInputStream("db1.properties");
10)       p.load(fis);// to load all properties from properties file into java Properties object
11)       Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE"
     ,p);
12)       Statement st =con.createStatement();
13)       ResultSet rs=st.executeQuery("select * from employees");
14)       System.out.println("ENO\tENAME\tESAL\tEADDR");
15)       System.out.println("--------------------------");
16)       while(rs.next())
17)       {
18)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
     tring(4));
19)       }
```

```
20)       con.close();
21)    }
22) }
```

Q. How many getConnection() methods are available in DriverManager class.

1. Connection con=DM.getConnection(url,user,pwd);
2. Connection con=DM.getConnection(url,Properties);
3. Connection con=DM.getConnection(url);

Eg:

```
Connection
con=DM.getConnection("jdbc:mysql://localhost:3306/durgadb?user=root&password=root");
```

Eg:

```
Connection con=DriverManager.getConnection("jdbc:oracle:thin:scott/tiger@localhost:1521:XE");
```

Transaction Management in JDBC

Process of combining all related operations into a single unit and executing on the rule "either all or none", is called transaction management.

Hence transaction is a single unit of work and it will work on the rule "either all or none".

Case-1: Funds Transfer

1. debit funds from sender's account
2. credit funds into receiver's account

All operations should be performed as a single unit only. If debit from sender's account completed and credit into receiver's account fails then there may be a chance of data inconsistency problems.

Case-2: Movie Ticket Reservation

1. Verify the status
2. Reserve the tickets
3. Payment
4. issue tickets.

All operations should be performed as a single unit only. If some operations success and some operations fails then there may be data inconsistency problems.

Transaction Properties:

Every Transaction should follow the following four ACID properties.

1. A ➔ Atomiticity
Either all operations should be done or None.

2. C ➔ Consistency(Reliabile Data)
It ensures bringing database from one consistent state to another consistent state.

3. I ➔ isolation (Sepatation)
Ensures that transaction is isolated from other transactions

4. D ➔ Durability
It means once transaction committed, then the results are permanent even in the case of system restarts, errors etc..

Types of Transactions:

There are two types of Transactions

1. Local Transactions
2. Global Transactions

1. Local Transactions:

All operations in a transaction are executed over same database.

Eg: Funds transfer from one accoun to another account where both accounts in the same bank.

2. Global Transactions:

All operations is a transaction are expected over different databases.

Eg: Funds Transfer from one account to another account and accounts are related to different banks.

Note:
JDBC can provide support only for local transactions.
If we want global transactions then we have to go for EJB or Spring framework.

Process of Transaction Management in JDBC:

1. Disable auto commit mode of JDBC

By default auto commit mode is enabled. i.e after executing every sql query, the changes will be committed automatically in the database.
We can disable auto commit mode as follows

```
con.setAutoCommit(false);
```

2. If all operations completed then we can commit the transaction by using the following method.

```
con.commit();
```

3. If any sql query fails then we have to rollback operations which are already completed by using rollback() method.

```
con.rollback();
```

Program: To demonstrate Transactions

```
1)  create table accounts(name varchar2(10),balance number);
2)
3)  insert into accounts values('durga',100000);
4)  insert into accounts values('sunny',10000);
```

TransactionDemo1.java

```
1)  import java.sql.*;
2)  import java.util.*;
3)  public class TransactionDemo1
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","scott","tiger");
8)        Statement st = con.createStatement();
9)        System.out.println("Data before Transaction");
10)       System.out.println("-------------------------");
11)       ResultSet rs =st.executeQuery("select * from accounts");
12)       while(rs.next())
13)       {
14)          System.out.println(rs.getString(1)+"..."+rs.getInt(2));
15)       }
16)       System.out.println("Transaction begins...");
17)       con.setAutoCommit(false);
18)       st.executeUpdate("update accounts set balance=balance-
    10000 where name='durga'");
19)       st.executeUpdate("update accounts set balance=balance+10000 where name='sunny'
    ");
20)       System.out.println("Can you please confirm this transaction of 10000....[Yes|No]");
21)       Scanner sc = new Scanner(System.in);
22)       String option = sc.next();
23)       if(option.equalsIgnoreCase("yes"))
24)       {
25)          con.commit();
26)          System.out.println("Transaction Commited");
27)       }
28)       else
29)       {
30)          con.rollback();
31)          System.out.println("Transaction Rolled Back");
32)       }
33)       System.out.println("Data After Transaction");
34)       System.out.println("-------------------------");
35)       ResultSet rs1 =st.executeQuery("select * from accounts");
36)       while(rs1.next())
```

```
37)       {
38)          System.out.println(rs1.getString(1)+"..."+rs1.getInt(2));
39)       }
40)       con.close();
41)    }
42) }
```

Savepoint(I):

Savepoint is an interface present in java.sql package.
Introduced in JDBc 3.0 Version.
Driver Software Vendor is responsible to provide implementation.
Savepoint concept is applicable only in Transactions.

Within a transaction if we want to rollback a particular group of operations based on some condition then we should go for Savepoint.

We can set Savepoint by using *setSavepoint()* method of Connection interface.

```
Savepoint sp = con.setSavepoint();
```

To perform rollback operation for a particular group of operations wrt Savepoint, we can use rollback() method as follows.

```
con.rollback(sp);
```

We can release or delete Savepoint by using release Savepoint() method of Connection interface.

```
con.releaseSavepoint(sp);
```

Case Study:

```
con.setAutoCommit(false);
Operation-1;
Operation-2;
Operation-3;
Savepoint sp = con.setSavepoint();
Operation-4;
Operation-5;
if(balance<10000)
{
  con.rollback(sp);
}
else
{
  con.releaseSavepoint(sp);
}
operation-6;
```

con.commit();

At line-1 if balance <10000 then operations 4 and 5 will be Rollback, otherwise all operations will be performed normally.

Program to Demonstrate Savepoint:

```
create table politicians(name varchar2(10),party varchar2(10));
```

```
1)  import java.sql.*;
2)  public class SavePointDemo1
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","scott","tiger");
7)        Statement st = con.createStatement();
8)        con.setAutoCommit(false);
9)        st.executeUpdate("insert into politicians values ('kcr','trs')");
10)       st.executeUpdate("insert into politicians values ('babu','tdp')");
11)       Savepoint sp = con.setSavepoint();
12)       st.executeUpdate("insert into politicians values ('siddu','bjp')");
13)       System.out.println("oops ..wrong entry just rollback");
14)       //con.rollback(sp);
15)       con.releaseSavepoint(sp);
16)       //System.out.println("Operations are roll back from Savepoint");
17)       con.commit();
18)       con.close();
19)    }
20) }
```

Note:

Some drivers won't provide support for Savepoint. Type-1 Driver won't provide support, but Type-4 Driver can provide support.

Type-4 Driver of Oracle provide support only for *setSavepoint()* and *rollback()* methods but not for *releaseSavepoint()* method.

Transaction Concurrency Problems:

Whenever multiple transactions are executing concurrently then there may be a chance of transaction concurrency problems.

The following are the most commonly occurred concurrency problems.
1. Dirty Read Problem
2. Non Repeatable Read Problem
3. Phantom Read Problem

1. Dirty Read Problem:

Also known as uncommitted dependency problem.
Before committing the transaction, if its intermediate results used by any other transaction then there may be a chance of Data inconsistency problems. This is called Dirty Read Problem.

durga:50000

T1:update accounts set balance=balance+50000 where name='durga'
T2:select balance from accounts where name='durga'
T1: con.rollback();

At the end, T1 point of view, durga has 50000 balance and T2 point of view durga has 1Lakh. There may be a chance of data inconsistency problem. This is called Dirty Read Problem.

2. Non-Repeatable Read Problem:

For the same Read Operation, in the same transaction if we get different results at different times, then such type of problem is called Non-Repeatable Read Problem.

Eg:
T1: select * from employees;
T2: update employees set esal=10000 where ename='durga';
T1: select * from employees;

In the above example Transaction-1 got different results at different times for the same query.

3. Phantom Read Problem:

A phantom read occurs when one transaction reads all the rows that satisfy a where condition and second transaction insert a new row that satisfy same where condition. If the first transaction reads for the same condition in the result an additional row will come. This row is called phantom row and this problem is called phantom read problem.

T1: select * from employees where esal >5000;
T2: insert into employees values(300,'ravi',8000,'hyd');
T1: select * from employees where esal >5000;

In the above code whenever transaction-1 performing read operation second time, a new row will come in the result.
To overcome these problems we should go for Transaction isolation levels.

Connection interface defines the following 4 transaction isolation levels.

1. TRANSACTION_READ_UNCOMMITTED ➔ 1
2. TRANSACTION_READ_COMMITTED ➔ 2
3. TRANSACTION_REPEATABLE_READ ➔ 4
4. TRANSACTION_SERIALIZABLE ➔ 8

1. TRANSACTION_READ_UNCOMMITTED:

It is the lowest level of isolation.

Before committing the transaction its intermediate results can be used by other transactions.

Internally it won't use any locks.

It does not prevent Dirty Read Problem, Non-Repeatable Read Problem and Phantom Read Problem.

We can use this isolation level just to indicate database supports transactions.

This isolation level is not recommended to use.

2. TRANSACTION_READ_COMMITTED:

This isolation level ensures that only committed data can be read by other transactions.

It prevents Dirty Read Problem. But there may be a chance of Non Repeatable Read Problem and Phantom Read Problem.

3. TRANSACTION_REPEATABLE_READ:

This is the default value for most of the databases. Internally the result of SQL Query will be locked for only one transaction. If we perform multiple read operations, then there is a guarantee that for same result.
It prevents Dirty Read Problem and Non Repeatable Read Problems. But still there may be a chance of Phantom Read Problem.

4. TRANSACTION_SERIALIZABLE:

It is the highest level of isolation.
The total table will be locked for one transaction at a time.
It prevents Dirty Read, Non-Repeatable Read and Phantom Read Problems.

Not Recommended to use because it may creates performance problems.
Connection interface defines the following method to know isolation level.

`getTransactionIsolation()`

Connection interface defines the following method to set our own isolation level.
`setTransactionIsolation(int level)`

Eg:

```
System.out.println(con.getTransactionIsolation());
con.setTransactionIsolation(8);
System.out.println(con.getTransactionIsolation());
```

Note:

For Oracle database, the default isolation level is: 2(TRANSACTION_READ_COMMITED).
Oracle database provides support only for isolation levels 2 and 8.

For MySql database, the default isolation level is: 4(TRANSACTION_REPEATABLE_READ).
MySql database can provide support for all isolation levels (1, 2, 4 and 8).

Program to demonstrate Oracle database Isolation levels:

```
1)  import java.sql.*;
2)  public class TransactionDemo2
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","scott","tiger");
7)        System.out.println(con.getTransactionIsolation());//2
8)        con.setTransactionIsolation(8);
9)        System.out.println(con.getTransactionIsolation());
10)
11)    }
12) }
```

Program to demonstrate MySQL database Isolation levels:

```
1)  import java.sql.*;
2)  public class TransactionDemo3
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/durgad
    b","root","root");
7)        System.out.println(con.getTransactionIsolation());//4
8)        con.setTransactionIsolation(8);
9)        System.out.println(con.getTransactionIsolation());//8
10)
11)    }
12) }
```

Q. In JDBC how many transaction isolation levels are defined?

The following 5 isolation levels are defined.

1) **TRANSACTION_NONE ➔ 0**
   It indicates that database won't provide support for transactions.

2) **TRANSACTION_READ_UNCOMMITTED ➔ 1**

3) **TRANSACTION_READ_COMMITTED ➔ 2**

4) **TRANSACTION_REPEATABLE_READ ➔ 4**

5) **TRANSACTION_SERIALIZABLE ➔ 8**

Summary Table of Isolation Levels

Isolation Level	Is Dirty Problem Prevented?	Is Non Repeatable Read Problem Prevented?	Is Phantom Read Problem Prevented?
TRANSACTION_READ_UNCOMMITTED	No	No	No
TRANSACTION_READ_COMMITTED	Yes	No	No
TRANSACTION_REPEATABLE_READ	Yes	Yes	No
TRANSACTION_SERIALIZABLE	Yes	Yes	Yes

MetaData

Metadata means data about data. I.e. Metadata provides extra information about our original data.

Eg:
Metadata about database is nothing but database product name, database version etc..

Metadata about ResultSet means no of columns, each column name, column type etc..

JDBC provides support for 3 Types of Metadata

1. DatabaseMetaData
2. ResultSetMetaData
3. ParameterMetaData

1. DatabaseMetaData

It is an interface present in java.sql package.
Driver Software vendor is responsible to provide implementation.

We can use DatabaseMetaData to get information about our database like database product name, driver name, version, number of tables etc..

We can also use DatabaseMetaData to check whether a particular feature is supported by DB or not like stored procedures, full joins etc..

We can get DatabaseMetaData object by using getMetaData() method of Connection interface.

```
public DatabaseMetaData  getMetaData();
```

Eg: `DatabaseMetaData dbmd=con.getMetaData();`

Once we got DatabaseMetaData object we can call several methods on that object like

```
getDatabaseProductName()
getDatabaseProductVersion()
getMaxColumnsInTable()
supportsStoredProcedures()
etc...
```

App1: Program to display Database meta information by using DataBaseMetaData

```
1)  import java.sql.*;
2)  class DatabaseMetaDataDemo1
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","system","durga");
7)        DatabaseMetaData dbmd=con.getMetaData();
8)        System.out.println("Database Product Name:"+dbmd.getDatabaseProductName());
9)        System.out.println("DatabaseProductVersion:"+dbmd.getDatabaseProductVersion());

10)       System.out.println("DatabaseMajorVersion:"+dbmd.getDatabaseMajorVersion());
11)       System.out.println("DatabaseMinorVersion:"+dbmd.getDatabaseMinorVersion());
12)       System.out.println("JDBCMajorVersion:"+dbmd.getJDBCMajorVersion());
13)       System.out.println("JDBCMinorVersion:"+dbmd.getJDBCMinorVersion());
14)       System.out.println("DriverName:"+dbmd.getDriverName());
15)       System.out.println("DriverVersion:"+dbmd.getDriverVersion());
16)       System.out.println("URL:"+dbmd.getURL());
17)       System.out.println("UserName:"+dbmd.getUserName());
18)       System.out.println("MaxColumnsInTable:"+dbmd.getMaxColumnsInTable());
19)       System.out.println("MaxRowSize:"+dbmd.getMaxRowSize());
20)       System.out.println("MaxStatementLength:"+dbmd.getMaxStatementLength());
21)       System.out.println("MaxTablesInSelect"+dbmd.getMaxTablesInSelect());
22)       System.out.println("MaxTableNameLength:"+dbmd.getMaxTableNameLength());
23)       System.out.println("SQLKeywords:"+dbmd.getSQLKeywords());
24)       System.out.println("NumericFunctions:"+dbmd.getNumericFunctions());
25)       System.out.println("StringFunctions:"+dbmd.getStringFunctions());
26)       System.out.println("SystemFunctions:"+dbmd.getSystemFunctions());
27)            System.out.println("supportsFullOuterJoins:"+dbmd.supportsFullOuterJoins());
28)       System.out.println("supportsStoredProcedures:"+dbmd.supportsStoredProcedures());

29)       con.close();
30)    }
31) }
```

App2: Program to display Table Names present in Database by using DataBaseMetaData

```
1)  import java.sql.*;
2)  import java.util.*;
3)  class DatabaseMetaDataDemo2
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        int count=0;
8)        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","system","durga");
9)        DatabaseMetaData dbmd=con.getMetaData();
```

```
10)       String catalog=null;
11)       String schemaPattern=null;
12)       String tableNamePattern=null;
13)       String[] types=null;
14)       ResultSet rs = dbmd.getTables(catalog,schemaPattern,tableNamePattern,types);
15)       //the parameters can help limit the number of tables that are returned in the ResultSe
      t
16)       //the ResultSet contains 10 columns and 3rd column represent table names.
17)       while(rs.next())
18)       {
19)          count++;
20)          System.out.println(rs.getString(3));
21)       }
22)       System.out.println("The number of tables:"+count);
23)       con.close();
24)    }
25) }
```

Note: Some driver softwares may not capture complete information. In that case we will get default values like zero.

Eg: getMaxRowSize() ➔ 0

ResultSetMetaData:

It is an interface present in java.sql package.
Driver software vendor is responsible to provide implementation.
It provides information about database table represented by ResultSet object.

Useful to get number of columns, column types etc..

We can get ResultSetMetaData object by using getMetaData() method of ResultSet interface.

```
public ResultSetMetaData getMetaData()
```

Eg: ResultSetMetaData rsmd=rs.getMetaData();

Once we got ResultSetMetaData object, we can call the following methods on that object like

```
getColumnCount()
getColumnName()
getColumnType()
etc...
```

App3: Program to display Columns meta information by using ResultMetaData

```
1)  import java.sql.*;
2)  class ResultSetMetaDataDemo
3)  {
```

```
4)     public static void main(String[] args) throws Exception
5)     {
6)        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","system","durga");
7)        Statement st = con.createStatement();
8)        ResultSet rs = st.executeQuery("select * from employees");
9)        ResultSetMetaData rsmd=rs.getMetaData();
10)       int count=rsmd.getColumnCount();
11)       for(int i=1;i<= count;i++)
12)       {
13)          System.out.println("Column Number:"+i);
14)          System.out.println("Column Name:"+rsmd.getColumnName(i));
15)          System.out.println("Column Type:"+rsmd.getColumnType(i));
16)          System.out.println("Column Size:"+rsmd.getColumnDisplaySize(i));
17)          System.out.println("---------------");
18)       }
19)       con.close();
20)    }
21) }
```

App3: Program to display Table Data including Column Names by using ResultMetaData

```
1)  import java.sql.*;
2)  class ResultSetMetaDataDemo1
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","scott","tiger");
7)        Statement st = con.createStatement();
8)        ResultSet rs = st.executeQuery("select * from movies");
9)        ResultSetMetaData rsmd=rs.getMetaData();
10)       String col1=rsmd.getColumnName(1);
11)       String col2=rsmd.getColumnName(2);
12)       String col3=rsmd.getColumnName(3);
13)       String col4=rsmd.getColumnName(4);
14)       System.out.println(col1+"\t"+col2+"\t"+col3+"\t"+col4);
15)       System.out.println("--------------------------------");
16)            while(rs.next())
17)       {
18)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getString(3)+"\t"+rs.get
    String(4));
19)       }
20)    }
21) }
```

ParameterMetaData (I):

It is an interface and present in java.sql package.
Driver Software vendor is responsible to provide implementation.

In General we can use positional parameters(?) while creating PreparedStatement object.

```
PreparedStatement pst=con.prepareStatement("insert into employees values(?,?,?,?)");
```

We can use ParameterMetaData to get information about positional parameters like parameter count,parameter mode, and parameter type etc...

We can get ParameterMetaData object by using getParameterMetaData() method of PreparedStatement interface.

```
ParameterMetaData pmd=pst.getParameterMetaData();
```

Once we got ParameterMetaData object, we can call several methods on that object like

1. int getParameterCount()
2. int getParameterMode(int param)
3. int getParameterType(int param)
4. String getParameterTypeName(int param)

etc..

Important Methods of ParameterMetaData:

1. int getParameterCount()

Retrieves the number of parameters in the PreparedStatement object for which this ParameterMetaData object contains information.

2.int getParameterMode(int param)

Retrieves the designated parameter's mode.

3. int getParameterType(int param)

Retrieves the designated parameter's SQL type.

4. String getParameterTypeName(int param)

Retrieves the designated parameter's database-specific type name.

5. int getPrecision(int param)

Retrieves the designated parameter's specified column size.

6. int getScale(int param)

Retrieves the designated parameter's number of digits to right of the decimal point.

7. int isNullable(int param)

Retrieves whether null values are allowed in the designated parameter.

8. boolean isSigned(int param)

Retrieves whether values for the designated parameter can be signed numbers.

App14: Program to display Parameter meta information by using ParameterMetaData

```
1)  import java.sql.*;
2)  class ParameterMetaDataDemo
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","system","durga");
7)        PreparedStatement pst = con.prepareStatement("insert into employees values(?,?,?,?)
    ");
8)        ParameterMetaData pmd=pst.getParameterMetaData();
9)        int count=pmd.getParameterCount();
10)       for(int i=1;i<= count;i++)
11)       {
12)          System.out.println("Parameter Number:"+i);
13)          System.out.println("Parameter Mode:"+pmd.getParameterMode(i));
14)          System.out.println("Parameter Type:"+pmd.getParameterType(i));
15)          System.out.println("Parameter Precision:"+pmd.getPrecision(i));
16)          System.out.println("Parameter Scale:"+pmd.getScale(i));
17)          System.out.println("Parameter isSigned:"+pmd.isSigned(i));
18)          System.out.println("Parameter isNullable:"+pmd.isNullable(i));
19)          System.out.println("----------------");
20)       }
21)       con.close();
22)    }
23) }
```

Note: Most of the drivers won't provide support for ParameterMetaData.

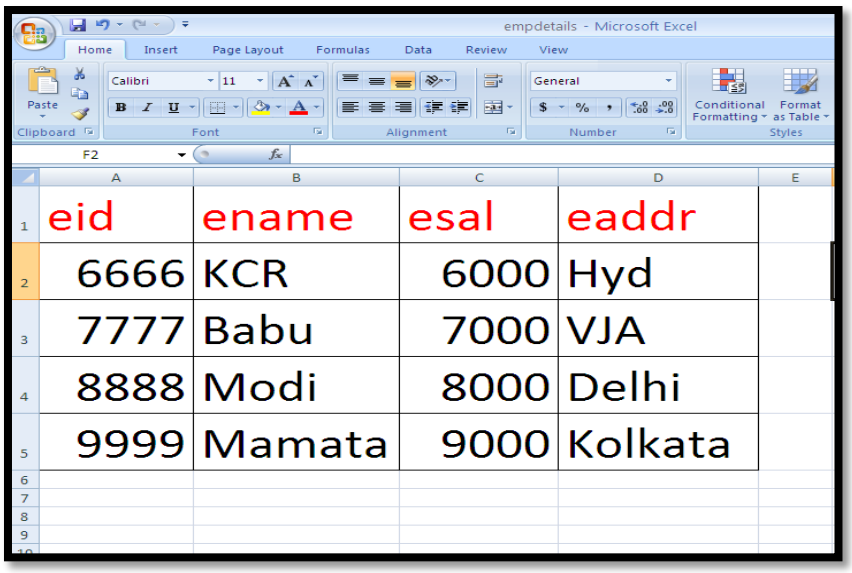
JDBC with Excel Sheets

1) We can read and write Data w.r.t Excel Sheet by using JDBC.
   To work with Excel it is recommended to use Type - 1 Driver.
   While configuring DSN we have to specify Driver Name as "Driver do Microsoft Excel"

2) We have to browse our Excel File by using "Select Work Book" Button.

3) Each Excel Work Book contains several Sheets.
   While writing Query we have to specify Sheet Name also.

```
ResultSet rs=st.executeQuery("select * from [Sheet1$]");
```



	eid	ename	esal	eaddr
2	6666	KCR	6000	Hyd
3	7777	Babu	7000	VJA
4	8888	Modi	8000	Delhi
5	9999	Mamata	9000	Kolkata

Program to Read Data From Excel Sheet by using JDBC:

```
1)  import java.sql.*;
2)  public class  ExcelDemo1
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:demoexcel11");
8)        Statement st = con.createStatement();
9)        ResultSet rs=st.executeQuery("select * from [Sheet1$]");
10)       System.out.println("ENO\tENAME\tESAL\tEADDR");
11)       System.out.println("---------------------------------------- ");
12)       while(rs.next())
13)       {
```

```
14)          System.out.println(rs.getInt(1)+"..."+rs.getString(2)+"..."+rs.getFloat(3)+"..."+rs.get
    String(4));
15)      }
16)      con.close();
17)   }
18) }
```

Progaram to read data from excel and copy into Oracle Database:

```
1)  import java.sql.*;
2)  public class   ExcelDemo2
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:demodsnforexcel2");
8)        Statement st = con.createStatement();
9)        ResultSet rs=st.executeQuery("select * from [Sheet1$]");
10)       Connection con2=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","scott","tiger");
11)       PreparedStatement pst=con2.prepareStatement("insert into employees values(?,?,?,?
    )");
12)       while(rs.next())
13)       {
14)          pst.setInt(1,rs.getInt(1));
15)          pst.setString(2,rs.getString(2));
16)          pst.setFloat(3,rs.getFloat(3));
17)          pst.setString(4,rs.getString(4));
18)          pst.executeUpdate();
19)       }
20)       System.out.println("Data Inserted Successfully from Excel to Oracle");
21)       con.close();
22)       con2.close();
23)    }
24) }
```

ResultSet Types

Division-1:

Based on operations performed on ResultSet, we can divide ResultSet into 2 types

1. Read Only ResultSets (Static ResultSets)
2. Updatable ResultSets (Dynamic ResultSets)

1.Read Only ResultSets:

We can perform only read operations on the ResultSet by using corresponding getter methods and we cannot perform any updations.

By default ResultSet is Read Only.

We can specify explicitly ResultSet as Read only by using the following constant of ResultSet.

```
public static final int CONCUR_READ_ONLY ➔ 1007
```

2. Updatable ResultSets:

The ResultSet which allows programmer to perform updations, such type of ResultSets are called Updatable ResultSets.

In this case we can perform select, insert, delete and update operations.

We can specify ResultSet explicitly as Updatable by using the following constant of ResultSet.

```
public static final int CONCUR_UPDATABLE ➔ 1008
```

Division-2:

Based on Cursor movement, ResultSets will be divided into the following 2 types.

1. Forward only (Non-Scrollable) ResultSet
2. Scrollable ResultSets

1. Forward Only ResultSets:

It allows the programmers to iterate records only in forward direction ie from top to bottom sequentially.

By default every ResultSet is forward only.

We can specify explicitly ResultSet as Forward only by using the following constant of ResultSet

public static final int TYPE_FORWARD_ONLY ➔ 1003

2. Scrollable ResultSets:

It allows the programmers to iterate in both forward and backward directions.

We can also jump to a particular position randomly, or relative to current position. Here we can move to anywhere.

There are two types of Scrollable ResultSets.

1. Scroll Insensitive ResultSet
2. Scroll Sensitive ResultSet

1. Scroll Insensitive ResultSet:

After getting ResultSet if we are performing any change in Database and if those changes are not reflecting to the ResultSet, such type of ResultSets are called scroll insensitive ResultSets.

i.e ResultSet is insensitive to database operations.

We can specify explicitly ResultSet as Scroll insensitive by using the following constant

public static final int TYPE_SCROLL_INSENSITIVE ➔ 1004

2.Scroll sensitive ResultSets:

After getting the ResultSet if we perform any change in the database and if those changes are visible to ResultSet, such type of ResultSet is called Scroll sensitive ResultSet.

i.e ResultSet is sensitive to database operations

We can specify explicitly ResultSet as scroll sensitive by using the following constant..

public static final int TYPE_SCROLL_SENSITIVE ➔ 1005

Differnences Between Scroll Insensitive And Scroll Sensitive ResultSets

Scroll Insensitive	Scroll Sensitive
After getting ResultSet if we perform any updation in the DB then those updation are not visible to the ResultSet i.e. ResultSet is insensitive to DB updations.	After getting ResultSet if we perform any updation in the DB then those updation are by default available to the to the ResultSet i.e. ResultSet is sensitive to DB updations.
As insensitive ResultSet is like snapshot of Data in DB when Query will be executed.	A Sensitive ResultSet doesn't represent snap shot of Data. It contains Pointers to Rows of DB directly, which satisfy Query Condition.
Relatively Performance is High.	Relatively Performance is low because for get Operation a Trip is required to DB.

Differnences between Forward only and Scrollable ResultSets

Non Scrollable (Forward only)	Scrollable
Cursor can move only in Forward Direction.	Cursor can move in both Forward and Backward Direction.
This Cursor can't move randomly.	Scrollable ResultSet Cursor can move randomly.
By using Non Scrollable ResultSet Cursor if we want to move Nth Record (N + 1) Iterations are required.	Performance is high because Cursor can move randomly to any Record.

How to get Required ResultSet:

We can create Statement objects as follows to get desired ResultSets.

```
Statement st =con.createStatement(int type,int mode);
PreparedStatement pst=con.prepareStatement(query,int type,int mode);
```

Allowed values for type are:

- ResultSet.TYPE_FORWARD_ONLY → 1003
- ResultSet.TYPE_SCROLL_INSENSITIVE → 1004
- ResultSet.TYPE_SCROLL_SENSITIVE → 1005

Allowed values for mode are:

- ResultSet.CONCUR_READ_ONLY → 1007
- ResultSet.CONCUR_UPDATABLE → 1008

Eg: for Scroll sensitive and updatable ResultSet:

```
Statement st =con.createStatement(1005,1008);
```

Note: Default type is forward only and default mode is read only.

Note:

To use various types of ResultSets underlying database support and driver support must be required

Some databases and some driver softwares wont provide proper support.

We can check whether the database supports a particular type of ResultSet or not by using the following methods of DatabaseMetaData.

1. public boolean supportsResultSetConcurrency(int type, int concurrency)

Retrieves whether this database supports the given concurrency type in combination with the given result set type.

2. public boolean supportsResultSetType(int type)

Retrieves whether this database supports the given result set type.

Program to Check whether database supports particular type of ResultSet or not

```
1)  import java.sql.*;
2)  class ResultSetTypeTest
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","scott","tiger");
7)        DatabaseMetaData dbmd=con.getMetaData();
8)        System.out.println(dbmd.supportsResultSetConcurrency(1003,1007));
9)        System.out.println(dbmd.supportsResultSetConcurrency(1003,1008));
10)       System.out.println(dbmd.supportsResultSetType(1003));
11)       System.out.println(dbmd.supportsResultSetType(1004));
12)       System.out.println(dbmd.supportsResultSetType(1005));
13)    }
14) }
```

List of allowed methods on Non-Scrollable ResultSets(Forward only):

1.rs.next()

it checks whether next record is available. If it is available then cursor will move to that position

2.rs.getXxx()

Read column values from record either with column index or with column name

3.rs.getRow()

It returns current position of cursor in the ResultSet i.e row number

List of allowed methods on Scrollable ResultSets:

1.rs.next()
2.rs.getXxx()
3.rs.getRow()

4.rs.previous()
It checks whether previous record is available or not. If it is available then the cursor will move to that record position

5.rs.beforeFirst();
the cursor will be moved to before first record position

6.rs.afterLast()
moves the cursor after last record position

7.rs.first()
moves the cursor to the first record position

8.rs.last()
moves the cursor to the last record position

9.rs.absolute(int x)
The argument can be either positive or negative.
If it is positive then the cursor will be moved to that record position from top of ResultSet.
If the argument is negative then it will be moved to the specified record position from last.

10.rs.relative(int x)
The argument can be either positive or negative
If the argument is positive then the cursor will move to forward direction to specified number of records from the current position. If the argument is negative then the cursor will move to backward direction to the specified number of records from the current position.

11. rs.isFirst()
returns true if the cursor is locating at first record position

12. rs.isLast()
13. rs.isBeforeFirst()
14. rs.isAfterLast()

15. rs.refreshRow()
We can use this method in scroll sensitive ResultSets to update row with latest values from Database.

Q. What is the difference Between absolute() and relative()methods?

absolute() method will always work either from BFR or from ALR.

relative() method will work wrt to current position.

In both methods +ve number means we have to move forward direction and -ve number means we have to move backward direction.

Note:

1. rs.last() and rs.absolute(-1) both are equal
2. rs.first() and rs.absolute(1) both are equal

Application-1: Iterating records in both forward and backward direction by using SCROLLABLE ResultSet

```
1)  import java.sql.*;
2)  class ResultSetTypesDemo1
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","scott","tiger");
7)        Statement st =con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.C
    ONCUR_READ_ONLY);
8)        ResultSet rs=st.executeQuery("select * from employees");
9)        System.out.println("Records in Forward Direction");
10)       System.out.println("ENO\tENAME\tESAL\tEADDR");
11)       System.out.println("-------------------------");
12)       while(rs.next())
13)       {
14)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
    tring(4));
15)       }
16)       System.out.println("Records in Backword Direction");
17)       System.out.println("ENO\tENAME\tESAL\tEADDR");
18)       System.out.println("-------------------------");
19)       while(rs.previous())
20)       {
21)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
    tring(4));
22)       }
23)       con.close();
24)    }
25) }
```

Application-2: Navigating Records by using SCROLLABLE ResultSet

```
1)  import java.sql.*;
2)  class ResultSetTypesDemo2
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
```

```
6)      Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
","scott","tiger");
7)      Statement st =con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.C
ONCUR_READ_ONLY);
8)      ResultSet rs=st.executeQuery("select * from employees");
9)      System.out.println("Records in Original Order");
10)     System.out.println("ENO\tENAME\tESAL\tEADDR");
11)     System.out.println("--------------------------");
12)     while(rs.next())
13)     {
14)        System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
tring(4));
15)     }
16)     rs.first();// First Record
17)     System.out.println(rs.getRow()+"---
>"+rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
18)     rs.last();//Last Record
19)     System.out.println(rs.getRow()+"---
>"+rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
20)     rs.relative(-1);//2nd Record from the last
21)     System.out.println(rs.getRow()+"---
>"+rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
22)     rs.absolute(2);//2nd Record
23)     System.out.println(rs.getRow()+"---
>"+rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
24)     con.close();
25)   }
26) }
```

Application-3: Reflecting Database updations by using SCROLL SENSITIVE ResultSet (Type-1 Driver)

```
1)  import java.sql.*;
2)  class ResultSetTypesDemo3
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:demodsn","scott","tiger");
8)      System.out.println(con);
9)      Statement st =con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CON
CUR_UPDATABLE);
10)     ResultSet rs=st.executeQuery("select * from employees");
11)     System.out.println("Records Before Updation");
12)     System.out.println("ENO\tENAME\tESAL\tEADDR");
13)     System.out.println("--------------------------");
14)     while(rs.next())
15)     {
```

```
16)         System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
    tring(4));
17)       }
18)       System.out.println("Application is in pausing state,please update database..");
19)       System.in.read();
20)       rs.beforeFirst();
21)       System.out.println("Records After Updation");
22)       while(rs.next())
23)       {
24)          rs.refreshRow();
25)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
    tring(4));
26)       }
27)       con.close();
28)    }
29) }
```

Note: Very few Drivers provide support for SCROLL_SENSITIVE Result Sets. Type-1 Driver will provide support for this feature. But it supports only update operation, but not delete and insert operations.

Type-2 driver also can provide support for SCROLL_SENSITIVE ResultSets. But we should not use * in select query. we should use only column names. It supports only update operation, but not delete and insert operations.

Application-4: Reflecting Database updations by using SCROLL SENSITIVE ResultSet (Type-2 Driver)

```
1)  import java.sql.*;
2)  import java.util.*;
3)  class ResultSetTypesDemo3T2
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        Connection con = DriverManager.getConnection("jdbc:oracle:oci8:@XE","scott","tiger
    ");
8)        System.out.println(con);
9)        Statement st =con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CON
    CUR_UPDATABLE);
10)       ResultSet rs=st.executeQuery("select eno,ename,esal,eaddr from employees");
11)       System.out.println("Records Before Updation");
12)       System.out.println("ENO\tENAME\tESAL\tEADDR");
13)       System.out.println("-------------------------");
14)       while(rs.next())
15)       {
16)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
    tring(4));
17)       }
```

```
18)       System.out.println("Application is in pausing state,please update database..");
19)       System.in.read();
20)       rs.beforeFirst();
21)       System.out.println("Records After Updation");
22)       while(rs.next())
23)       {
24)          rs.refreshRow();
25)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
     tring(4));
26)       }
27)       con.close();
28)    }
29) }
```

Note: Very few Drivers provide support for SCROLL_SENSITIVE Result Sets. Type-1 Driver will provide support for this feature. But it supports only update operation, but not delete and insert operations.

Type-2 driver also can provide support for SCROLL_SENSITIVE ResultSets. But we should not use * in select query. we should use only column names. It supports only update operation, but not delete and insert operations.

Updatable ResultSets:

If we perform any changes to the ResultSet and if those changes are reflecting to the Database, such type of ResultSets are called Updatable ResultSets.

By default ResultSet is Read only. But we can specify explicitly as updatable by using the following constant.

CONCUR_UPDATABLE ➔ 1008

For Updatable ResultSets, we have to create Statement object as follows..

```
Statement st
=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
```

Delete Record From ResultSet:

Sample Code:

```
rs.last();
rs.deleteRow();
```

Application-7: Performing Database updations (DELETE operation) by using UPDATABLE ResultSet (Type-1 Driver)

```
1)  import java.sql.*;
2)  import java.util.*;
3)  class ResultSetTypesDemo5
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
8)       Connection con = DriverManager.getConnection("jdbc:odbc:demodsn","scott","tiger");
9)        Statement st =con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CON
    CUR_UPDATABLE);
10)       ResultSet rs=st.executeQuery("select * from employees");
11)       rs.last();
12)       rs.deleteRow();
13)       con.close();
14)    }
15) }
```

Note: Very few Drivers provide support for CONCUR_UPDATABLE Result Sets. Type-1 Driver will provide support for this feature.

Type-2 driver also can provide support for CONCUR_UPDATABLE ResultSets. But we should not use * In select query. we should use only column names.

Update Record of ResultSet:

Sample Code Eg 2:

```
1)  rs.absolute(3);
2)  rs.updateString(2,"KTR");
3)  rs.updateFloat(3,10000);
4)  rs.updateRow();
```

Sample Code Eg 2:

```
1)  while(rs.next())
2)        {
3)           float esal = rs.getFloat(3);
4)           if(esal<5000)
5)           {
6)              float incr_sal=esal+777;
7)              rs.updateFloat(3,incr_sal);
8)              rs.updateRow();
9)           }
10)       }
```

Application-5: Performing Database updations (UPDATE operation) by using UPDATABLE ResultSet (Type-1 Driver)

```
1)  import java.sql.*;
2)  import java.util.*;
3)  class ResultSetTypesDemo4
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
8)       Connection con = DriverManager.getConnection("jdbc:odbc:demodsn","scott","tiger");
9)        Statement st =con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CON
    CUR_UPDATABLE);
10)       ResultSet rs=st.executeQuery("select * from employees");
11)       while(rs.next())
12)       {
13)          float esal = rs.getFloat(3);
14)          if(esal<5000)
15)          {
16)             float incr_sal=esal+777;
17)             rs.updateFloat(3,incr_sal);
18)             rs.updateRow();
19)          }
20)       }
21)       con.close();
22)    }
23) }
```

Note: Very few Drivers provide support for CONCUR_UPDATABLE Result Sets. Type-1 Driver will provide support for this feature.

Type-2 driver also can provide support for CONCUR_UPDATABLE ResultSets. But we should not use * In select query. we should use only column names.

Application-6: Performing Database updations (UPDATE operation) by using UPDATABLE ResultSet (Type-2 Driver)

```
1)  import java.sql.*;
2)  import java.util.*;
3)  class ResultSetTypesDemo4T2
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        //Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
8)     Connection con = DriverManager.getConnection("jdbc:oracle:oci8:@XE","scott","tiger");
9)        Statement st =con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CON
    CUR_UPDATABLE);
10)       ResultSet rs=st.executeQuery("select eno,ename,esal,eaddr from employees");
```

```
11)       while(rs.next())
12)       {
13)          float esal = rs.getFloat(3);
14)          if(esal<5000)
15)          {
16)             float incr_sal=esal+777;
17)             rs.updateFloat(3,incr_sal);
18)             rs.updateRow();
19)          }
20)       }
21)       con.close();
22)    }
23) }
```

Note: Very few Drivers provide support for CONCUR_UPDATABLE Result Sets. Type-1 Driver will provide support for this feature.

Type-2 driver also can provide support for CONCUR_UPDATABLE ResultSets. But we should not use * In select query. we should use only column names.

Insert operation:

Sample code:

```
1)  rs.moveToInsertRow();
2)  rs.updateInt(1,1010);
3)  rs.updateString(2,"sunny");
4)  rs.updateFloat(3,3000);
5)  rs.updateString(4,"Mumbai");
6)  rs.insertRow();
```

Application-8: Performing Database updations (INSERT operation) by using UPDATABLE ResultSet (Type-1 Driver)

```
1)  import java.sql.*;
2)  class ResultSetTypesDemo6
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:demodsn","scott","tiger");
8)        Statement st =con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CON
    CUR_UPDATABLE);
9)        ResultSet rs=st.executeQuery("select * from employees");
10)       rs.moveToInsertRow();//creates an empty record
11)       rs.updateInt(1,900);
12)       rs.updateString(2,"katrina");
13)       rs.updateFloat(3,3000);
```

```
14)       rs.updateString(4,"Hyd");
15)       rs.insertRow();
16)       con.close();
17)    }
18) }
```

Note: Very few Drivers provide support for CONCUR_UPDATABLE Result Sets. Type-1 Driver will provide support for this feature.

Type-2 driver also can provide support for CONCUR_UPDATABLE ResultSets. But we should not use * In select query. we should use only column names.

Conclusions:

1.Updatable ResultSets allows the programmer to perform following operations on ResultSet.
- select
- insert
- delete
- update

2. Updatable ResultSets allows the programmer to perform insert, update and delete database operations without using SQL Queries.

3.Very few drivers provide support for Updatable ResultSets.
- Type-1 Driver provides support
- Type-2 Driver provides support but we should not use * in SQL Query and we should use column names.

4.ResultSet cannot be updatable if we are using joins and aggregate functions

5.It is not recommended to perform database updations by using updatable ResultSets, b'z most of the drivers and most of the databases won't provide support for Updatable ResultSets.

ResultSet Holdability:

The ResultSet holdability represents whether the ResultSet is closed or not whenever we call commit() method on the Connection object.

There are two types of Holdability

```
HOLD_CURSORS_OVER_COMMIT ➔ 1
CLOSE_CURSORS_AT_COMMIT ➔ 2
```

HOLD_CURSORS_OVER_COMMIT:

It means the ResultSet will be opened for further operations even after calling con.commit() method.

CLOSE_CURSORS_AT_COMMIT:

It means that ResultSet will be closed automatically whenever we are calling con.commit() method.

We can get Current Holdability of the ResultSet as follows.

```
SOP(rs.getHoldability());
```

For most of the databases default holdabiltiy is 1

We can check whether database provides support for a particular holdability or not by using the following method of DatabaseMetaData.

```
supportsResultSetHoldability()
```

We can create Statement object for our required Holdability as follows...

```
Statement st = con.createStatement(1005,1008,2);

RS rs = st.executeQuery("select * from employees");
con.commit();
rs.aboslute(3); ➔ SQLException
```

Note: **Most of the databases like Oracle, MySQL won't provide support for holdability 2.**

Program to check ResultSet Holdability:

```
1)  import java.sql.*;
2)  import java.util.*;
3)  class ResultSetHoldabilityDemo1
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","scott","tiger");
8)        DatabaseMetaData dbmd=con.getMetaData();
9)        if(dbmd.supportsResultSetHoldability(1))
10)       {
11)          System.out.println("HOLD_CURSORS_OVER_COMMIT");
12)       }
13)       if(dbmd.supportsResultSetHoldability(2))
14)       {
15)          System.out.println("CLOSE_CURSORS_AT_COMMIT");
16)       }
17)    }
18) }
```

Program to display properties of ResultSet:

```
1)  import java.sql.*;
2)  import java.util.*;
3)  class ResultSetHoldabilityDemo3
4)  {
5)     public static void main(String[] args) throws Exception
6)     {
7)        Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE
    ","scott","tiger");
8)        Statement st =con.createStatement();
9)        System.out.println("Type :"+st.getResultSetType());
10)       System.out.println("Concurrency :"+st.getResultSetConcurrency());
11)       System.out.println("Holdability:"+st.getResultSetHoldability());
12)    }
13) }
```

Summary of ResultSet Types

ResultSet Type	ResultSet Concurrency	ResultSet Holadability
TYPE_FORWARD_ONLY [1003] TYPE_SROLL_INSENSITIVE [1004] TYPE_ SROLL_SENSITIVE [1005] The Default Concurrency is TYPE_FORWARD_ONLY	CONCUR_READ_ONLY [1007] CONCUR_UPDATABLE [1008] The Default Concurrency is CONCUR_READ_ONLY	HOLD_CURSORS_OVER_ COMMIT [1] CLOSE_CURSORS_AT_ COMMIT [2] The Default Holadability is HOLD_CURSORS_OVER_ COMMIT

RowSets

It is alternative to ResultSet.

We can use RowSet to handle a group of records in more effective way than ResultSet.
RowSet interface present in javax.sql package

RowSet is child interface of ResultSet.

RowSet implementations will be provided by Java vendor and database vendor.

By default RowSet is scrollable and updatable.

By default RowSet is serializable and hence we can send RowSet object across the network. But ResultSet object is not serializable.

ResultSet is connected i.e to use ResultSet compulsary database Connection must be required.

RowSet is disconnected. ie to use RowSet database connection is not required.

Types of RowSets:

There are two types of RowSets

1.Connected RowSets
2. Disconnected RowSets

Connected RowSets:

Connected RowSets are just like ResultSets.
To access RowSet data compulsary connection should be available to database.

We cannot serialize Connected RowSets

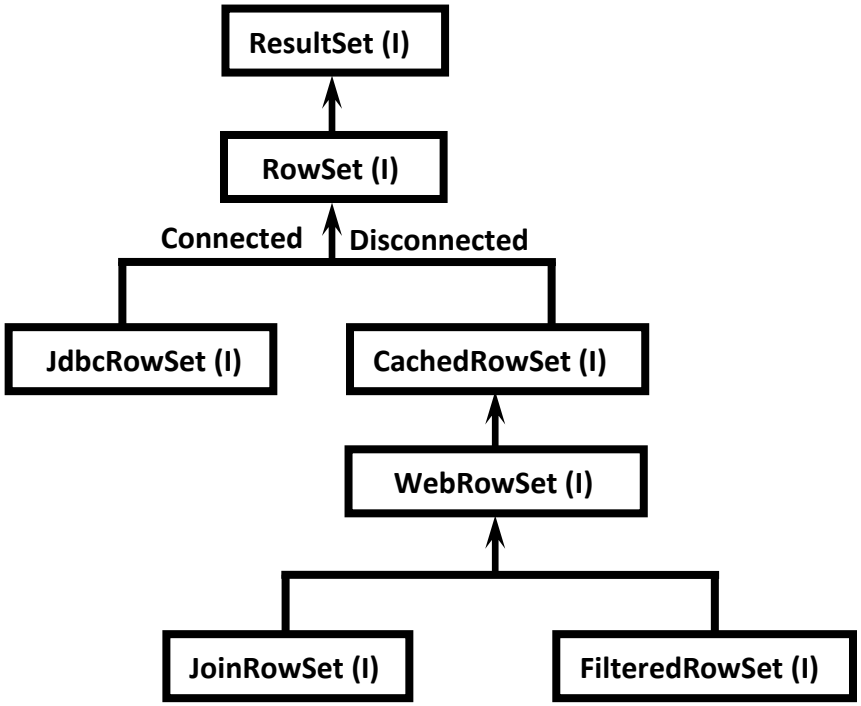
Eg: JdbcRowSet

Disconnected RowSets:

Without having Connection to the database we can access RowSet data.
We can serialize Disconnected RowSets.

Eg:
CachedRowSet
WebRowSet

FilteredRowSet
JoinRowSet



How to create RowSet objects:

We can create different types of RowSet objects as follows

```
RowSetFactory rsf = RowSetProvider.newFactory();
JdbcRowSet jrs = rsf.createJdbcRowSet();
CachedRowSet crs = rsf.createCachedRowSet();
WebRowSet wrs = rsf.createWebRowSet();
JoinRowSet jnrs = rsf.createJoinRowSet();
FilteredRowSet frs = rsf.createFilteredRowSet();
```

Application-1: To create Different RowSet Objects:

```
1)  import javax.sql.rowset.*;
2)  public class Test
3)  {
4)     public static void main(String[] args) throws Exception
5)     {
6)        RowSetFactory rsf=RowSetProvider.newFactory();
7)        JdbcRowSet jrs=rsf.createJdbcRowSet();
8)        CachedRowSet crs=rsf.createCachedRowSet();
9)        WebRowSet wrs=rsf.createWebRowSet();
10)       JoinRowSet jnrs=rsf.createJoinRowSet();
11)       FilteredRowSet frs=rsf.createFilteredRowSet();
12)
```

```
13)       System.out.println(jrs.getClass().getName());
14)       System.out.println(crs.getClass().getName());
15)       System.out.println(wrs.getClass().getName());
16)       System.out.println(jnrs.getClass().getName());
17)       System.out.println(frs.getClass().getName());
18)    }
19) }
```

1.JdbcRowSet(I):

It is exactly same as ResultSet except that it is scrollable and updatable.
JdbcRowSet is connected and hence to access JdbcRowSet compulsary Connection must be required.
JdbcRowSet is non serializable and hence we cannot send RowSet object across the network.

Application-2: To Retrieve records from JdbcRowSet:

```
1)  import javax.sql.rowset.*;
2)  public class JdbcRowSetRetrieveDemo {
3)     public static void main(String[] args)throws Exception {
4)         RowSetFactory rsf=RowSetProvider.newFactory();
5)         JdbcRowSet rs=rsf.createJdbcRowSet();
6)        rs.setUrl("jdbc:mysql://localhost:3306/durgadb");
7)        rs.setUsername("root");
8)        rs.setPassword("root");
9)        rs.setCommand("select * from employees");
10)       rs.execute();
11)       System.out.println("Employee Details In Forward Direction");
12)       System.out.println("ENO\tENAME\tESAL\tEADDR");
13)       System.out.println("--------------------------------");
14)       while(rs.next()){
15)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
    tring(4));
16)       }
17)       System.out.println("Employee Details In Backward Direction");
18)       System.out.println("ENO\tENAME\tESAL\tEADDR");
19)       System.out.println("--------------------------------");
20)       while(rs.previous()){
21)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
    tring(4));
22)       }
23)       System.out.println("Accessing Randomly...");
24)       rs.absolute(3);
25)       System.out.println(rs.getRow()+"---
    >"+rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
26)       rs.first();
27)       System.out.println(rs.getRow()+"---
    >"+rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
```

```
28)       rs.last();
29)       System.out.println(rs.getRow()+"---
      >"+rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
30)       rs.close();
31)    }
32) }
```

Application-3: To Insert Records by using JdbcRowSet

```
1)  import java.util.Scanner;
2)  import javax.sql.rowset.*;
3)  public class JdbcRowSetInsertDemo {
4)     public static void main(String[] args)throws Exception {
5)        RowSetFactory rsf=RowSetProvider.newFactory();
6)        JdbcRowSet rs=rsf.createJdbcRowSet();
7)        rs.setUrl("jdbc:mysql://localhost:3306/durgadb");
8)        rs.setUsername("root");
9)        rs.setPassword("root");
10)       rs.setCommand("select * from employees");
11)       rs.execute();
12)       Scanner s=new Scanner(System.in);
13)       rs.moveToInsertRow();
14)       while(true){
15)          System.out.print("Employee Number    :");
16)          int eno=s.nextInt();
17)          System.out.print("Employee Name     :");
18)          String ename=s.next();
19)          System.out.print("Employee Salary   :");
20)          float esal=s.nextFloat();
21)          System.out.print("Employee Address  :");
22)          String eaddr=s.next();
23)
24)          rs.updateInt(1, eno);
25)          rs.updateString(2, ename);
26)          rs.updateFloat(3, esal);
27)          rs.updateString(4, eaddr);
28)          rs.insertRow();
29)
30)          System.out.println("Employee Inserted Successfully");
31)          System.out.print("Do You Want to insert One more Employee[yes/no]?  :");
32)          String option=s.next();
33)          if(option.equalsIgnoreCase("No")){
34)             break;
35)          }
36)       }
37)       rs.close();
38)    }
39) }
```

Application-4: To Update Records by using JdbcRowSet

```
1)  import javax.sql.rowset.*;
2)  public class JdbcRowSetUpdateDemo {
3)     public static void main(String[] args)throws Exception {
4)        RowSetFactory rsf=RowSetProvider.newFactory();
5)        JdbcRowSet rs=rsf.createJdbcRowSet();
6)        rs.setUrl("jdbc:mysql://localhost:3306/durgadb");
7)        rs.setUsername("root");
8)        rs.setPassword("root");
9)        rs.setCommand("select * from employees");
10)       rs.execute();
11)       while(rs.next()){
12)          float esal=rs.getFloat(3);
13)          if(esal<10000){
14)             float new_Esal=esal+500;
15)             rs.updateFloat(3, new_Esal);
16)             rs.updateRow();
17)          }
18)       }
19)       System.out.println("Records Updated Successfully");
20)       rs.close();
21)    }
22) }
```

Application-5: To Delete Records by using JdbcRowSet

```
1)  import javax.sql.rowset.*;
2)  public class JdbcRowSetDeleteDemo {
3)     public static void main(String[] args)throws Exception {
4)        RowSetFactory rsf=RowSetProvider.newFactory();
5)        JdbcRowSet rs=rsf.createJdbcRowSet();
6)        rs.setUrl("jdbc:mysql://localhost:3306/durgadb");
7)        rs.setUsername("root");
8)        rs.setPassword("root");
9)        rs.setCommand("select * from employees");
10)       rs.execute();
11)       while(rs.next()){
12)          float esal=rs.getFloat(3);
13)          if(esal>5000){
14)             rs.deleteRow();
15)          }
16)       }
17)       System.out.println("Records Deleted Successfully");
18)       rs.close();
19)    }
20) }
```

CachedRowSet:

It is the child interface of RowSet.
It is bydefault scrollable and updatable.
It is disconnected RowSet. ie we can use RowSet without having database connection.

It is Serializable.

The main advantage of CachedRowSet is we can send this RowSet object for multiple people across the network and all those people can access RowSet data without having DB Connection.

If we perform any update operations(like insert,delete and update) to the CachedRowSet,to reflect those changes compulsary Connection should be established.
Once Connection established then only those changes will be reflected in Database.

Application-6: To Demonstrate Disconnected CachedRowSet

```
1)  import java.sql.*;
2)  import javax.sql.rowset.*;
3)  public class CachedRowSetDemo {
4)     public static void main(String[] args)throws Exception {
5)        Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/durgadb
    ","root","root");
6)        Statement st =con.createStatement();
7)        ResultSet rs =st.executeQuery("select * from employees");
8)        RowSetFactory rsf=RowSetProvider.newFactory();
9)        CachedRowSet crs=rsf.createCachedRowSet();
10)       crs.populate(rs);
11)       con.close();
12)       //Now we cannot access RS but we can access CRS
13)       //if(rs.next()){}RE:SQLException:Operation not allowed after ResultSet closed
14)       System.out.println("ENO\tENAME\tESAL\tEADDR");
15)       System.out.println("----------------------------");
16)       while(crs.next()){
17)          System.out.println(crs.getInt(1)+"\t"+crs.getString(2)+"\t"+crs.getFloat(3)+"\t"+crs.
    getString(4));
18)       }
19)    }
20) }
```

Application-7: To Retrieve Records by using CachedRowSet

```
1)  import javax.sql.rowset.*;
2)  public class CachedRowSetRetrieveDemo {
3)     public static void main(String[] args)throws Exception {
4)        RowSetFactory rsf=RowSetProvider.newFactory();
5)        CachedRowSet rs=rsf.createCachedRowSet();
6)        rs.setUrl("jdbc:mysql://localhost:3306/durgadb");
```

```
7)        rs.setUsername("root");
8)        rs.setPassword("root");
9)        rs.setCommand("select * from employees");
10)       rs.execute();
11)       System.out.println("Data In Forward Direction");
12)       System.out.println("ENO\tENAME\tESAL\tEADDR");
13)       System.out.println("-----------------------------");
14)       while(rs.next()){
15)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
     tring(4));
16)       }System.out.println("Data In Backward Direction");
17)       System.out.println("ENO\tENAME\tESAL\tEADDR");
18)       System.out.println("-------------------------------");
19)       while(rs.previous()){
20)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
     tring(4));
21)       }
22)    }
23) }
```

Application-8: To Insert Records by using CachedRowSet

```
1)  import java.util.*;
2)  import javax.sql.rowset.*;
3)
4)  public class CachedRowSetInsertDemo {
5)     public static void main(String[] args)throws Exception{
6)        RowSetFactory rsf=RowSetProvider.newFactory();
7)        CachedRowSet rs=rsf.createCachedRowSet();
8)        rs.setUrl("jdbc:mysql://localhost:3306/durgadb?relaxAutoCommit=true");
9)        rs.setUsername("root");
10)       rs.setPassword("root");
11)       rs.setCommand("select * from employees");
12)       rs.execute();
13)       Scanner s=new Scanner(System.in);
14)       while(true){
15)          System.out.print("Employee Number  :");
16)          int eno=s.nextInt();
17)          System.out.print("Employee Name    :");
18)          String ename=s.next();
19)          System.out.print("Employee Salary  :");
20)          float esal=s.nextFloat();
21)          System.out.print("EMployee Address :");
22)          String saddr=s.next();
23)
24)          rs.moveToInsertRow();
25)          rs.updateInt(1, eno);
26)          rs.updateString(2, ename);
27)          rs.updateFloat(3, esal);
```

```
28)         rs.updateString(4, saddr);
29)         rs.insertRow();
30)
31)         System.out.println("Employee Inserted Successfully");
32)         System.out.print("Do you want to insert One more Employee[Yes/No]?  :");
33)         String option=s.next();
34)         if(option.equalsIgnoreCase("No")){
35)            break;
36)         }
37)       }
38)       rs.moveToCurrentRow();
39)       rs.acceptChanges();
40)       rs.close();
41)    }
42) }
```

Application-9: To Update Records by using CachedRowSet

```
1)  import javax.sql.rowset.*;
2)
3)  public class CachedRowSetUpdateDemo {
4)     public static void main(String[] args)throws Exception{
5)
6)       RowSetFactory rsf=RowSetProvider.newFactory();
7)       CachedRowSet rs=rsf.createCachedRowSet();
8)       rs.setUrl("jdbc:mysql://localhost:3306/durgadb?relaxAutoCommit=true");
9)       rs.setUsername("root");
10)      rs.setPassword("root");
11)      rs.setCommand("select * from employees");
12)      rs.execute();
13)      while(rs.next()){
14)         float esal=rs.getFloat(3);
15)         if(esal<10000){
16)            esal=esal+500;
17)            rs.updateFloat(3, esal);
18)            rs.updateRow();
19)         }
20)      }
21)      rs.moveToCurrentRow();
22)      rs.acceptChanges();
23)      System.out.println("Records Updated Successfully");
24)      rs.close();
25)   }
26) }
```

Application-10: To Delete Records by using CachedRowSet

```
1)  import javax.sql.rowset.*;
2)
3)  public class CachedRowSetDeleteDemo {
4)     public static void main(String[] args)throws Exception{
5)
6)        RowSetFactory rsf=RowSetProvider.newFactory();
7)        CachedRowSet rs=rsf.createCachedRowSet();
8)        rs.setUrl("jdbc:mysql://localhost:3306/durgadb?relaxAutoCommit=true");
9)        rs.setUsername("root");
10)       rs.setPassword("root");
11)       rs.setCommand("select * from employees");
12)       rs.execute();
13)       while(rs.next()){
14)          float esal=rs.getFloat(3);
15)          if(esal>6000){
16)            rs.deleteRow();
17)          }
18)       }
19)       rs.moveToCurrentRow();
20)       rs.acceptChanges();
21)       rs.close();
22)       System.out.println("Records deleted successfully");
23)    }
24) }
```

WebRowSet(I):

It is the child interface of CachedRowSet.
It is bydefault scrollable and updatable.
It is disconnected and serializable

WebRowSet can publish data to xml files,which are very helpful for enterprise applications.

```
FileWriter fw=new FileWriter("emp.xml");
rs.writeXml(fw);
```

We can read XML data into RowSet as follows

```
FileReader fr=new FileReader("emp.xml");
rs.readXml(fr);
```

Application-11: To Retrieve Records by using WebRowSet

```
1)  import java.io.*;
2)  import javax.sql.rowset.*;
3)
```

```
4)  public class WebRowSetRetrieveDemo {
5)     public static void main(String[] args)throws Exception {
6)        RowSetFactory rsf=RowSetProvider.newFactory();
7)        WebRowSet rs=rsf.createWebRowSet();
8)        rs.setUrl("jdbc:mysql://localhost:3306/durgadb");
9)        rs.setUsername("root");
10)       rs.setPassword("root");
11)       rs.setCommand("select * from employees");
12)       rs.execute();
13)       FileWriter fw=new FileWriter("emp.xml");
14)       rs.writeXml(fw);
15)       System.out.println("Employee Data transfered to emp.xml file");
16)       fw.close();
17)    }
18) }
```

Application-12: To Insert Records by using WebRowSet

```
1)  import java.io.*;
2)  import javax.sql.rowset.*;
3)  public class WebRowSetInsertDemo {
4)     public static void main(String[] args)throws Exception {
5)        RowSetFactory rsf=RowSetProvider.newFactory();
6)        WebRowSet rs=rsf.createWebRowSet();
7)        rs.setUrl("jdbc:mysql://localhost:3306/durgadb?relaxAutoCommit=true");
8)        rs.setUsername("root");
9)        rs.setPassword("root");
10)       rs.setCommand("select * from employees");
11)       rs.execute();
12)       FileReader fr=new FileReader("emp.xml");
13)       rs.readXml(fr);
14)       rs.acceptChanges();
15)       System.out.println("emp data inserted successfully");
16)       fr.close();
17)       rs.close();
18)    }
19) }
```

Note: In emp.xml file, <insertRow> tag must be provided under <data> tag

Application-13: To Delete Records by using WebRowSet

```
1)  import java.io.*;
2)  import javax.sql.rowset.*;
3)  public class WebRowSetDeleteDemo {
4)     public static void main(String[] args)throws Exception {
5)        RowSetFactory rsf=RowSetProvider.newFactory();
6)        WebRowSet rs=rsf.createWebRowSet();
```

```
7)        rs.setUrl("jdbc:mysql://localhost:3306/durgadb?relaxAutoCommit=true");
8)        rs.setUsername("root");
9)        rs.setPassword("root");
10)       rs.setCommand("select * from employees");
11)       rs.execute();
12)       FileReader fr=new FileReader("emp.xml");
13)       rs.readXml(fr);
14)       rs.acceptChanges();
15)       System.out.println("emp data deleted successfully");
16)       fr.close();
17)       rs.close();
18)    }
19) }
```

Note: In emp.xml file, <deleteRow> tag must be provided under <data> tag

JoinRowSet:

It is the child interface of WebRowSet.
It is by default scrollable and updatable
It is disconnected and serializable
If we want to join rows from different rowsets into a single rowset based on matched column(common column) then we should go for JoinRowSet.

We can add RowSets to the JoinRowSet by using addRowSet() method.

```
addRowSet(RowSet rs,int commonColumnIndex);
```

Application-14: To Retrieve Records by using JoinRowSet

```
1)  import java.sql.*;
2)  import javax.sql.rowset.*;
3)  public class JoinRowSetRetriveDemo {
4)     public static void main(String[] args)throws Exception {
5)        //Class.forName("com.mysql.jdbc.Driver");
6)        Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/durgadb
    ","root", "root");
7)        RowSetFactory rsf=RowSetProvider.newFactory();
8)
9)        CachedRowSet rs1=rsf.createCachedRowSet();
10)       rs1.setCommand("select * from student");
11)       rs1.execute(con);
12)
13)       CachedRowSet rs2=rsf.createCachedRowSet();
14)       rs2.setCommand("select * from courses");
15)       rs2.execute(con);
16)
17)       JoinRowSet rs=rsf.createJoinRowSet();
```

```
18)       rs.addRowSet(rs1, 4);
19)       rs.addRowSet(rs2, 1);
20)       System.out.println("SID\tSNAME\tSADDR\tCID\tCNAME\tCCOST");
21)       System.out.println("-----------------------------------------");
22)       while(rs.next()){
23)          System.out.print(rs.getString(1)+"\t");
24)          System.out.print(rs.getString(2)+"\t");
25)          System.out.print(rs.getString(3)+"\t");
26)          System.out.print(rs.getString(4)+"\t");
27)          System.out.print(rs.getString(5)+"\t");
28)          System.out.print(rs.getString(6)+"\n");
29)       }
30)       con.close();
31)    }
32) }
```

Note: students and courses tables must require in database with a matched column[Join column] cid.

students				courses		
SID(PK)	SNAME	SADDR	CID	CID(PK)	CNAME	CCOST

addRowSet(RowSet rowset, int columnIdx)
Adds the given RowSet object to this JoinRowSet object and sets the designated column as the match column for the RowSet object.

FilteredRowSet(I):

It is the child interface of WebRowSet.
If we want to filter rows based on some condition then we should go for FilteredRowSet.

We can define the filter by implementing Predicate interface.

```
1)  public class EmpSalFilter implements Predicate
2)  {
3)    evaluate(Object value,String columnName)
4)    {
5)      This method will be called at the time of insertion
6)    }
7)    evaluate(Object value,int columnIndex)
8)    {
9)      this method will be called at  the time of insertion
10)   }
11)   evaluate(RowSet rs)
12)   {
13)     filtering logic
14)   }
15) }
```

We can set Filter to the FilteredRowSet as follows...

```
EmployeeSalaryFilter f=new EmployeeSalaryFilter(2500,4000);
rs.setFilter(f);
```

Application-15: To Retrieve Records by using FilteredRowSet

```
1)  import java.sql.*;
2)  import javax.sql.*;
3)  import javax.sql.rowset.*;
4)  class EmployeeSalaryFilter implements Predicate{
5)     float low;
6)     float high;
7)     public EmployeeSalaryFilter(float low,float high) {
8)        this.low=low;
9)        this.high=high;
10)       }
11)     //this method will be called at the time of row insertion
12)     public boolean evaluate(Object value, String columnName) throws SQLException {
13)      return false;
14)    }
15)     //this method will be called at the time of row insertion
16)    public boolean evaluate(Object value, int column) throws SQLException {
17)       return false;
18)    }
19)     public boolean evaluate(RowSet rs) {
20)       boolean eval=false;
21)       try{
22)       FilteredRowSet frs=(FilteredRowSet)rs;
23)       float esal=frs.getFloat(3);
24)       if((esal>=low) && (esal<=high)){
25)          eval=true;
26)       }else{
27)          eval=false;
28)       }
29)       }catch(Exception e){
30)          e.printStackTrace();
31)       }
32)       return eval;
33)    }
34) }
35) public class FilteredRowSetRetriveDemo {
36)    public static void main(String[] args)throws Exception {
37)       RowSetFactory rsf=RowSetProvider.newFactory();
38)       FilteredRowSet rs=rsf.createFilteredRowSet();
39)       rs.setUrl("jdbc:mysql://localhost:3306/durgadb");
40)       rs.setUsername("root");
41)       rs.setPassword("root");
42)       rs.setCommand("select * from employees");
```

```
43)       rs.execute();
44)       System.out.println("Data Before Filtering");
45)       System.out.println("ENO\tENAME\tESAL\tEADDR");
46)       System.out.println("------------------------------");
47)       while(rs.next())
48)       {
49)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
     tring(4));
50)       }
51)       EmployeeSalaryFilter f=new EmployeeSalaryFilter(100,5000);
52)       rs.setFilter(f);
53)       rs.beforeFirst();
54)       System.out.println("Data After Filtering");
55)       System.out.println("ENO\tENAME\tESAL\tEADDR");
56)       System.out.println("------------------------------");
57)       while(rs.next())
58)       {
59)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
     tring(4));
60)       }
61)      rs.close();
62)    }
63) }
```

Event Handling Mechanism for RowSets:

We can implement event handling for RowSets.
To perform event handling,we have to implement RowSetListener interface.

```
1)  class RowSetListenerImpl implements RowSetListener
2)  {
3)    rowSetChanged(RowSetEvent e)
4)    {
5)      this method will be executed whenever total RowSet content changed
6)    }
7)    rowChanged(RowSetEvent e)
8)    {
9)      this method will be executed whenever any change performed in rows of RowSet like in
     sertion,deletion and updation
10)   }
11)   cursorMoved(RowSetEvent e)
12)   {
13)     this method will be executed whenever cursor moved from one row to another row
14)   }
15) }
```

We can add RowSetListener to the RowSet by using addRowSetListener() method.
Eg: rs.addRowSetListener(new RowSetListenerImpl());

Application-16: To Demonstrate Event Handling by using JdbcRowSet

```
1)  import javax.sql.*;
2)  import javax.sql.rowset.*;
3)  class RowSetListenerImpl implements RowSetListener{
4)
5)     public void rowSetChanged(RowSetEvent event) {
6)        System.out.println("RowSetChanged");
7)     }
8)
9)     public void rowChanged(RowSetEvent event) {
10)       System.out.println("RowChanged");
11)    }
12)
13)    public void cursorMoved(RowSetEvent event) {
14)       System.out.println("CursorMoved");
15)    }
16) }
17) public class RowSetListenerDemo {
18)
19)    public static void main(String[] args)throws Exception {
20)       RowSetFactory rsf=RowSetProvider.newFactory();
21)       JdbcRowSet rs=rsf.createJdbcRowSet();
22)       rs.setUrl("jdbc:mysql://localhost:3306/durgadb");
23)       rs.setUsername("root");
24)       rs.setPassword("root");
25)       rs.setCommand("select * from employees");
26)       rs.addRowSetListener(new RowSetListenerImpl());
27)       rs.execute();
28)       while(rs.next()){
29)          System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getS
     tring(4));
30)       }
31)       rs.moveToInsertRow();
32)       rs.updateInt(1, 777);
33)       rs.updateString(2, "malli");
34)       rs.updateFloat(3,9000);
35)       rs.updateString(4, "Hyd");
36)       rs.insertRow();
37)       rs.close();
38)    }
39) }
```

Methods of RowSetListener

void cursorMoved(RowSetEvent event)
Notifies registered listeners that a RowSet object's cursor has moved.

void rowChanged(RowSetEvent event)
Notifies registered listeners that a RowSet object has had a change in one of its rows.

void rowSetChanged(RowSetEvent event)
Notifies registered listeners that a RowSet object in the given RowSetEvent object has changed its entire contents.

Differences Between ResultSet and RowSet

ResultSet	RowSet
1) ResultSet present in java.sql Package.	1) RowSet present in javax.sql Package.
2) By Default ResultSet is Non Scrollable and Non Updatable (Forward only and Read only).	2) By Default RowSet is Scrollable and Updatable.
3) ResultSet Objects are Non Serializable and we can't send over Network.	3) RowSet Objects are Serializable and hence we can send over Network.
4) ResultSet Objects are Connection oriented i.e. we can access ResultSet Data as long as Connection is available once Connection closed we can't access ResultSet Data.	4) RowSet Objects are Connection Less Objects i.e. we can access RowSet Data without having Connection to DB (except JdbcRowSet).
5) ResultSet Object is used to store Records returned by Select Query.	5) RowSet Object is also used to store Records returned by Select Query.

6) We can createResultSet Object as follows
Connection con = DriverManager.getConnection (url, uname, pwd);
Statement st = con.createStatement();
ResultSet rs = st.executeQuery(SQLQuery);
	6) RowSetFactory rsf = RowSetProvider.newFactory();
	JdbcRowSet rs = rsf.createJdbcRowSet();
	rs.setUserName(user);
	rs.setUrl(jdbcurl);
	rs.setPassword(pwd);
	rs.setCommand(query);
	rs.execute();

7) ResultSet Object is not having Event Notification Model.	7) RowSet Object is having Event Notification Model.

Top Most Important JDBC FAQ's

Q1. What is Driver and how many types of drivers are there in JDBC?

The Main Purpose of JDBC Driver is to convert Java (JDBC) calls into Database specific calls and Database specific calls into Java calls. i.e. It acts as a Translator.

There are 4 Types of JDBC Drivers are available

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)

Q2. Explain Differences between executeQuery(), executeUpdate() and execute() methods?

We can use execute Methods to execute SQL Queries.
There are 4 execute Methods in JDBC.

1. executeQuery():

can be used for Select Queries

2. executeUpdate():

Can be used for Non-Select Queries (Insert|Delete|Update)

3. execute()

Can be used for both Select and Non-Select Queries
It can also be used to call Stored Procedures.

4. executeBatch()

Can be used to execute Batch Updates

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.

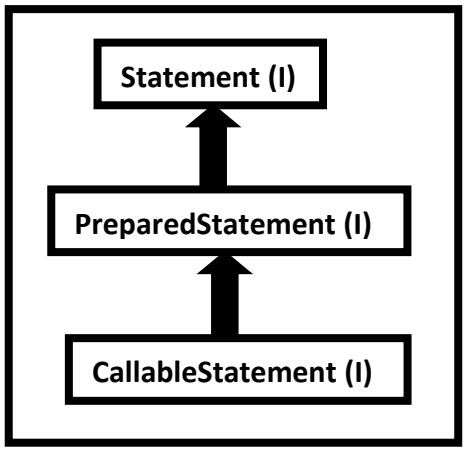
3. If we don't know the Type of SQL Query at the beginning and it is available dynamically at Runtime (may be from Properties File OR from Command Prompt etc) then we should go for execute() Method.

Q3. What is Statement and How many Types of Statements are available?

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

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

There are 3 Types of Statements



1. Statement:

If we want to execute multiple Queries then we can use Statement Object.
Every time Query will be compiled and executed. Hence relatively performance is low.

2. PreparedStatement:

If we want to execute same Query multiple times then we should go for PreparedStatement.
Here Query will be compiled only once even though we executed multiple times. Hence relatively performance is high.

PreparedStatement is always associated with precompiled SQL Queries.

3. CallableStatement:

We can use CallableStatement to call Stored Procedures and Functions from the Database.

Q4. Explain differences between Statement and PreparedStatement?

Differences Between Statement And PreparedStatement

Statement	PreparedStatement
1) At the time of creating Statement Object, we are not required to provide any Query. Statement st = con.createStatement(); Hence Statement Object is not associated with any Query and we can use for multiple Queries.	1) At the time of creating PreparedStatement, we have to provide SQL Query compulsory and will send to the Database and will be compiled. PS pst = con.prepareStatement(query); Hence PS is associated with only one Query.
2) Whenever we are using execute Method, every time Query will be compiled and executed.	2) Whenever we are using execute Method, Query won't be compiled just will be executed.
3) Statement Object can work only for Static Queries.	3) PS Object can work for both Static and Dynamic Queries.
4) Relatively Performance is Low.	4) Relatively Performance is High.
5) Best choice if we want to work with multiple Queries.	5) Best choice if we want to work with only one Query but required to execute multiple times.
6) There may be a chance of SQL Injection Attack.	6) There is no chance of SQL Injection Attack.
7) Inserting Date and Large Objects (CLOB and BLOB) is difficult.	7) Inserting Date and Large Objects (CLOB and BLOB) is easy.

Q5. Explain Steps to develop JDBC Application?

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

Q6. Explain main Important components of JDBC?

The Main Important Components of JDBC are:

1. Driver
2. DriverManager
3. Connection
4. Statement
5. ResultSet

1.Driver(Translator):

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

2. DriverManager:

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);
```

3. Connection (Road):

By using Connection, Java Application can communicate with Database.

4. Statement (Vehicle):

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

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

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

There are 3 types of Statements

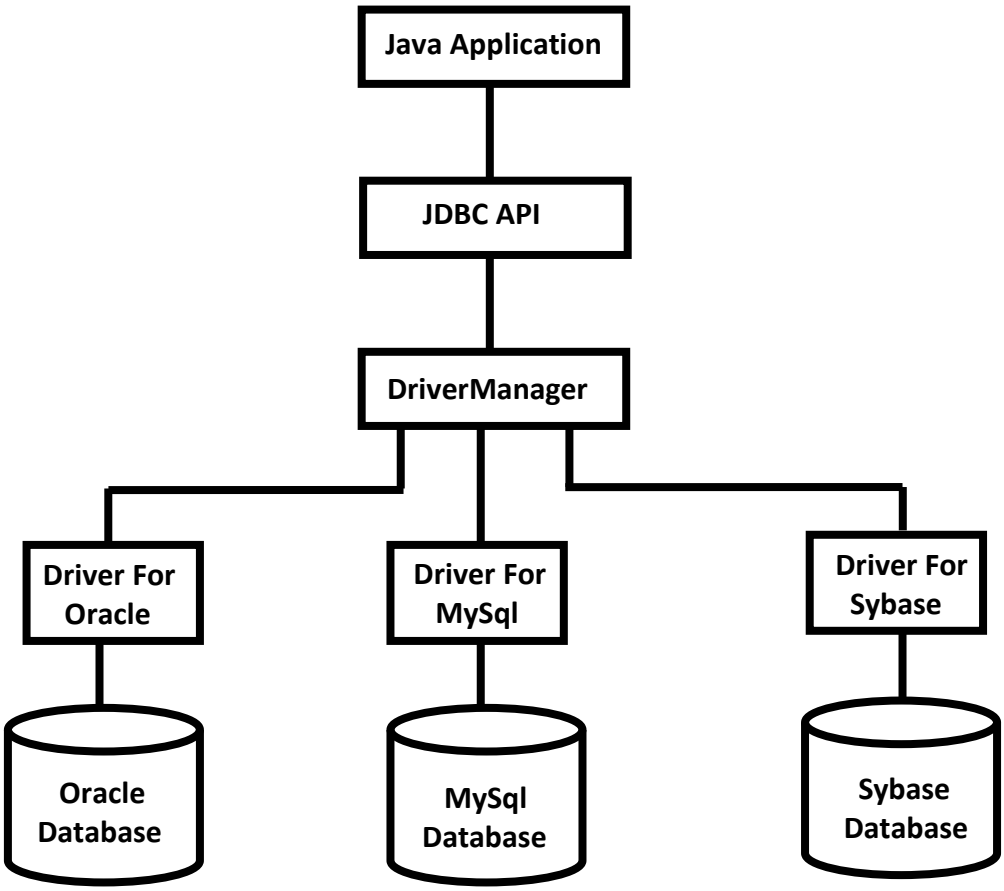
1.Statement
2.PreparedStatement
3.CallableStatement

5. ResultSet:

Whenever we are executing Select Query, Database engine will provide Result in the form of ResultSet. Hence ResultSet holds Results of SQL Query. By using ResultSet we can access the Results.

Q7. Explain JDBC Architecture?

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.**

Q8. Explain about BLOB and CLOB?

Sometimes as the Part of programming Requirement, we have to Insert and Retrieve Large Files like Images, Video Files, Audio Files, Resume etc wrt Database.

Eg:
Upload Image in Matrimonial Web Sites
Upload Resume in Job related Web Sites

To Store and Retrieve Large Information we should go for Large Objects (LOBs).

There are 2 Types of Large Objects.

1. Binary Large Object (BLOB)
2. Character Large Object (CLOB)

1. Binary Large Object (BLOB):

A BLOB is a Collection of Binary Data stored as a Single Entity in the Database.

BLOB Type Objects can be Images, Video Files, Audio Files etc..

BLOB Data Type can store Maximum of "4GB" Binary Data.

2. Character Large Objects (CLOB):

A CLOB is a Collection of Character Data stored as a Single Entity in the Database.

CLOB can be used to Store Large Text Documents (May Plain Text OR XML Documents)

CLOB Type can store Maximum of 4 GB Data.

Eg: hydhistory.txt

Q9. Explain about Batch Updates?

When we Submit Multiple SQL Queries to the Database one by one then lot of time will be wasted in Request and Response.

For Example our Requirement is to execute 1000 Queries. If we are trying to submit 1000 Queries to the Database one by one then we need to communicate with the Database 1000 times. It increases Network Traffic between Java Application and Database and even creates Performance Problems also.

To overcome these Problems, we should go for Batch Updates. We can Group all related SQL Queries into a Single Batch and we can send that Batch at a time to the Database.

Sample Code:

```
st.addBatch(sqlQuery-1);
st.addBatch(sqlQuery-2);
st.addBatch(sqlQuery-3);
st.addBatch(sqlQuery-4);
st.addBatch(sqlQuery-5);
st.addBatch(sqlQuery-6);
...
st.addBatch(sqlQuery-1000);
st.executeBatch();
```

Advanced Java

DURGA SOFTWARE SOLUTIONS®

JDBC Interview FAQ's

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

1) What is JDBC?

2) What is Latest version of JDBC available?

3) Explain about JDBC Architecture?

4) Explain about common JDBC Components?

5) Explain about DriverManager?

6) What is JDBC API?

8) Who has provided JDBC API?

9) What are the classes and interfaces available in JDBC API?

10) Who has provided implementation of JDBC API?

11) What are the steps to write JDBC Program?

12) What is JDBC Driver?

13) How many types of JDBS Drivers available?

14) Explain TYPE I Driver?

15) Which version of Java has excluded TYPE I Driver?

16) I have loaded both Oracle and MySQL drivers, Which database connection will be established when we call getConnection(...)method?

Ans: Based on jdbc url the Connection object will be created to the database.

17) I have loaded Oracle driver and trying to get the connection with MySQL URL What will happen?

Code:

```
Class.for Name("oracle.jdbc.OracleDriver");
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/durgadb","root","root");
```

Ans: We will get ClassNotFoundException

18) What the DriverManager.getConnection() method doing?

In JDBC API or in java.sql package, SUN has given more interfaces like Connection, Statement, ResultSet, Etc., How Instances will be created?

19) Can I register the Driver Explicitly?

20) Can I unregister the Driver?

21) How can i find list of drivers registered?

22) How many types of JDBC Drivers are available? Which is best?

23) Explain the cases when each driver should be used?

24) Which Type of JDBC Driver is the Fastest One?

25) Explain two important approaches to Register a Driver?

26) Whenever we are using Class.forName() method to load Driver class automatically Driver will be Registered with DriverManager. Then what is the need of DriverManager class registerDriver() method.

Ans: This method is useful whenever we are using Non-JDK Complaint Driver.

27) Can I establish two database connections at a time?

28) What are the difference among 3 getConnections() method?

1) **public static Connection getConnection(String url)**
2) **public static Connection getConnection(String url,String uname, String pword)**
3) **public static Connection getConnection(String url,Properties info)**

29) Can we specify the column name in the select statement or not?

30) What is the use of execute() if we have the executeUpdate() or executeQuery()?

31) What is Statement?

32) How many types of JDBC Statements are available?

33) In which package the statement is defined?

34) Is there any super type defined for statement?

35) Who is responsible to define implementation class for statement?

36) How to get /create the object of statement type?

37) While creating the statement do we need to provide any SQL statement?

38) What are the methods can be used from statement to submit the SQL Query to database.?

39) What is the difference among executeUpdate(), executeQuery() and execute() methods?

40) How many Queries we can submit by using one statement object?

41) How many types of queries I can submit using one statement object?

42) When exactly SQL statement will be submitted to the database?

43) When you submit the SQL statement to database using statement then how many times the SQL statement will be compiled/verified?

44) How to use dynamic value to the SQL statement in the case of statement object?

45) What is the PreparedStatement?

46) In which package the PreparedStatement is defined?

47) Is there any super type defined for PreparedStatement?

48) Who is responsible to define implementation class for PreparedStatement?

49) How to get/create the object of PreparedStatement type?

50) While creating the prepared Statement do we need to provide any SQL Statement?

51) What are the methods can be used from Prepared Statement to submit the SQL Query to database?

52) How many Queries we can submit using one Prepared Statement object?

53) How many types of queries We can submit using one PreparedStatement object?

54) When we submit the SQL statement to database using Prepared Statement then how many times the SQL Statement will be compiled/vefified?

55) How to use dynamic value to the SQL statement in the case of PreparedStatement object?

56) What is the difference between Statement and PreparedStatement ?

57) What is the benefit of PreparedStatement over Statement?

58) What is CallableStatement?

59) In Which package the CallableStatement is defined?

60) Is there any super type defined for CallableStatement?

61) Who is responsible to defined implementation class for CallableStatement?

62) How to get/create the object of CallableStatement type?

63) While creating the Callable Statement do we need to provide any SQL Statement?

64) What is the purpose/benefit of CallableStatement?

65) What are the methods can be used from CallableStatement to call the procedure from database?

66) When we call the procedure from database using CallableStatement then how many times the SQL Statement will be compiled/verified?

67) How to use dynamic value to the procedure in the case of CallableStatement object?

68) How can we call the procedure from Java application using input parameter?

69) How can you call the procedure from Java Application using output parameter of the procedure?

70) How to get the value of output parameter of the procedure?

71) Can we write different types of SQL statement in procedure?

72) Can we submit select statement using batch update?

73) How to get the result from the callable statement if you invoke any stored function?

74) How can you access column information from ResultSet?

75) Can I access Statement and ResultSet after closing the connection?

76) What is the Batch Update? OR What is the advantage of Batch Update?

77) How to use Batch Update with Statement?

78) How to use Batch Update with Preparedstatement?

79) Can I submit insert Statement using Batch Update?

80) Can I submit update Statement using Batch Update?

81) Can I submit delete Statement using Batch Update?

82) Can I submit select Statement using Batch Update?

83) Can I submit different types of SQL statement with Batch Update using Statement?

84) Can I submit different types of SQL statement with Batch Update using Prepared Statement?

85) What is Metadata?

86) What is DatabaseMetadata?

87) In Which package the DatabaseMetaData is available?

88) Who has defined the implementation class for DatabaseMetaData?

89) How can we get the object of DatabaseMetaData type?

90) What is the use of DatabaseMetaData?

91) How can I access the Database Product Name?

92) How can I access the Database Product version?

93) How can I access the Driver Name?

94) How can I access the Driver version?

95) How can I check whether Database supports batch update or not?

96) How can I check whether Database supports Full Outer Join or not?

97) What is ResultSetMetadata?

98) In Which package the ResultSetMetadata is available.?

99) Who has defined the implementation class for ResultSetMetadata?

100) How to get/create the object of ResultSetMetadata type?

101) What is the use of ResultSetMetadata type?

102) How can I get the number of columns available in Resultset?

103) How can I access the name & order of the columns available in Resultset?

104) How can I access the type of the columns available in Resultset?

105) What is transaction?

106) What is transaction management?

107) What is ACID properties?

108) What will happen when auto commit is true?

109) By using which methods we can implement Transactions in JDBC?

110) What are the Transactional concurrency problems?

111) Explain about Dirty Read Problem?

112) Explain about Repeatable Read Problem?

113) Explain about Phantom Read Problem?

114) What are the Transactional isolation levels?

115) Which isolation levels prevent Dirty Read Problem?

116) Which isolation levels prevent Repeatable Read Problem?

117) Which isolation levels prevent Phantom Read Problem?

118) What will happen when Iam not specifying the isolation Level with JDBC?

119) How can I get Database Vendor Specific Default Transactional Isolation Level?

120) What is the Default Transactional Isolation Level My SQL?

121) What is the Default Transactional Isolation Level Oracle?

122) What are the ways to manage the Connections in JDBC?

123) What are the advantages of DataSource Connections over Driver Manager connections ?

124) What is ResultSet?

125) In Which package , ResultSet is available.?

126) Who has defined the implementation class of ResultSet?

127) How can we get the Object of ResultSet Type?

128) What does the ResultSet represent?

129) What are the types of ResultSet available as per Cursor movement?

130) What is forward only ResultSet?

131) How can you get the Forward Only ResultSet?

132) Can I call the following method with Forward Only ResultSet?

a. previous() b. first() c. last() d. absolute() e. relative()

133) What is Scrollable ResultSet?

134) How can I get the Scrollable ResultSet?

135) Can I call the following method with Scrollable ResultSet?

- **previous()**
- **first()**
- **last()**
- **absolute()**
- **relative()**

136) What are the types of Resultset available as per Operation?

137) What are the Read Only ResultSet?

138) How can you get the Read Only Resultset?

139) Can I call the following method with Read Only Resultset?

- **moveToInsertRow()**
- **updateRow()**
- **deleteRow()**
- **insertRow()**
- **updateX(int col_Index, X value)**

140) What is updatable Resultset?

141) How can you get the updatable Resultset?

79. Can I call the following method with Updatable Resultset?

- **moveToInsertRow()**
- **updateRow()**
- **deleteRow()**
- **insertRow()**
- **updateX(int col_Index, X value)**

142) What is the default type of Resultset?

143) What are the constants defined to specify the Resultset type?

144) What is the default concurrency of Resultset?

145) What are the constants defined to specify the Resultset concurrency?

146) What is difference between Scroll SENSITIVE and INSENSITIVE?

147) What are various Types of ResultSet based on cursor movement?

148) What are various Types of ResultSet based on operations?

149) What are various Types of ResultSet based on holdability?

150) What is Rowset?

151) What is the super type for RowSet?

152) How to get the object of RowSet?

153) How many types of RowSet available as per connection?

154) How many sub types of RowSet interface available?

155) What is the default type of RowSet?

156) What is the default concurrency RowSet?

157) Can I serialize the Cached RowSet?

158) Can I serialize the JDBC RowSet?

159) What is the difference between ResultSet and RowSet?

160) What is the use of RowSet Factory and RowSet Provider?

161) What are the new features of JDBC 4.0?

162) What are the new features of JDBC 4.1?

163) What is ResultSet holdability?