# Java Database Connectivity

The process of storing data in permanent storage and using that data in our applications is called persistence.

- **Persistence Storage** :- The storage where data can be stored and managed permanently is called persistence storage.
- **Persistence Logic :-** The logic that is developed in the application to interact with persistence storage and to manipulate data in persistence storage is called persistence logic.
- **Persistence Operation**: - Insert, Update, Select and Delete operation perform in data of persistence store are called persistence operation.
  - ✓ Java application uses IO streams based persistence logic to interact with files.
  - ✓ Java application uses JDBC based persistence logic to interact with database software.
  - ✓ In small scale applications use files as persistence store.
  - ✓ In medium scale and large scale application use database software as persistence store.

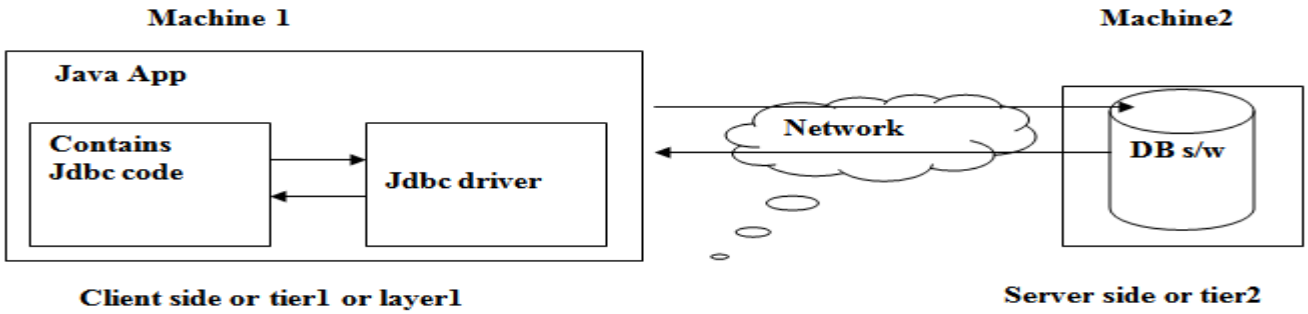
## The Limitations of Persistence Store :-

1. It does not provide security for data.
2. It does not support query language.
3. It cannot manage huge amount of data.
4. The manipulation of data by applying multiple conditions is impossible.
5. Merging and comparison of data is complex.

## Database software as persistance store

| DB SOFTWARE | VENDOR |
|---|---|
| Oracle | Oracle corporation |
| My SQL | Sun micro System |
| SQL server | Microsoft |
| DB2 | IBM |
| Postgre | Open symphony |

- App data is nothing but input values coming to application & the results generated by the application. If this data is not saved in persistence store like file or db software, we will lose this data at the end of the app execution. To solve this problem , we perform persistence operation on app data.




## JDBC :

- Java application uses JDBC code to interact with DB software.
- JDBC driver is the bridge between java application & DB software, that converts db calls to java calls and vice versa.

The JDBC code written in java app is actually required to activate & use JDBC driver to interact with DB software.

JDBC based application are two tier applications. The two layers or two tiers applications can reside in single computer or to reside in two different computers.

**Specification** is document containing rules & guidelines to develop new software.

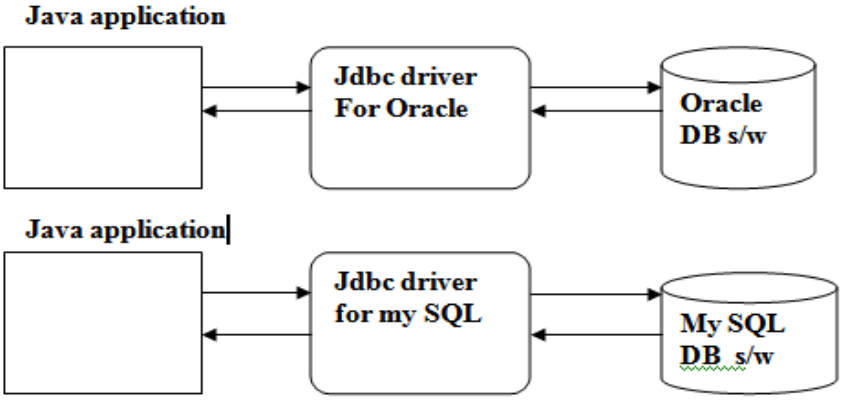
If specification is kept open to market for all companies to develop new software then it is called **open specification.**

Eg :-JDBC, Servlet, Jsp, ODBC & Ejb etc.

If specification is not open to market and only one company allowed to develop software based on the specification then it is called **proprietor specification**.

Eg:- (Dot).net specification.

JDBC is a specification, JDBC driver is a software developed based on JDBC specification. For small and medium application. JDBC is a open specification given by SUN MICROSYS SYSTEM having rule and guidelines to develop JDBC driver, so any software vendor can develop JDBC driver because it is open specification. For every DB software we need one separate JDBC drivers.




We can expect JDBC driver from 3 vendors

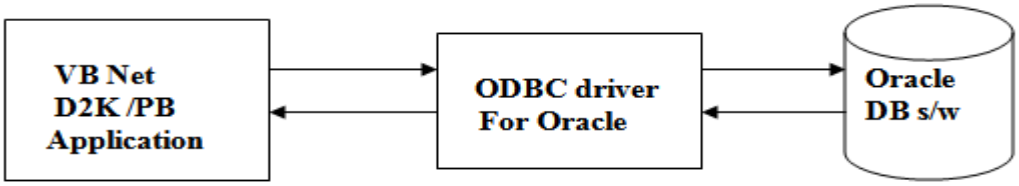
1.Sun micro systems
2.Data base vender
3.Third party vender

Even though JDBC drivers are coming from various software vendor companies for different DB softwares, we can work with all these database drivers in our java application to interact with DB softwares, since it is going to be much similar, because all these JDBC drivers will be given based on common rules and guide lines available in JDBC specification.

**What is JDBC ?** - JDBC is an open specification containing rules and guidelines to develop JDBC driver for DB(data base) softwares.

**What is ODBC(open data base connectivity) ?** - ODBC is an open specification containing rules and guidelines to develop odbc drivers for different DB software.

- Non java application like VB, net, D2K applications use Odbc drivers to interact with DB software.
- Every DB software needs one separate odbc drivers.

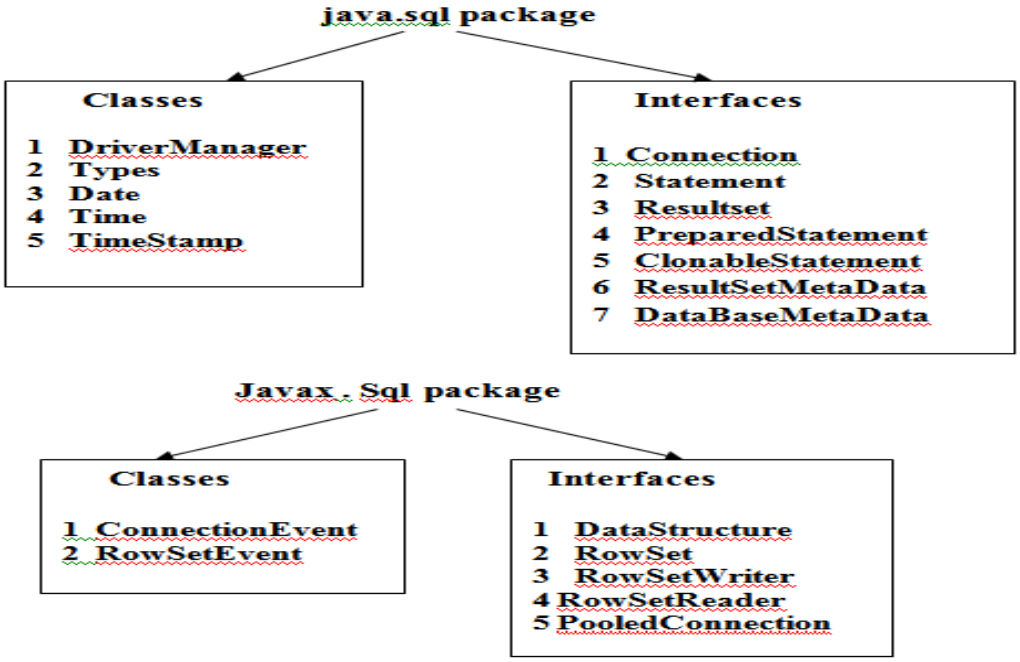


Sunmicrosystem has given JDBC specification based on java language. X-open company has given odbc specification based on c-language.

When all other technologies based application are using odbc drivers to interact with DB software **why java apps are separately looking for JDBC drivers** ?, because Odbc drivers are given based on c-language. So they take the support of pointers since java does not support pointers, hence java application needs separate java based JDBC drivers to interact with DB software.

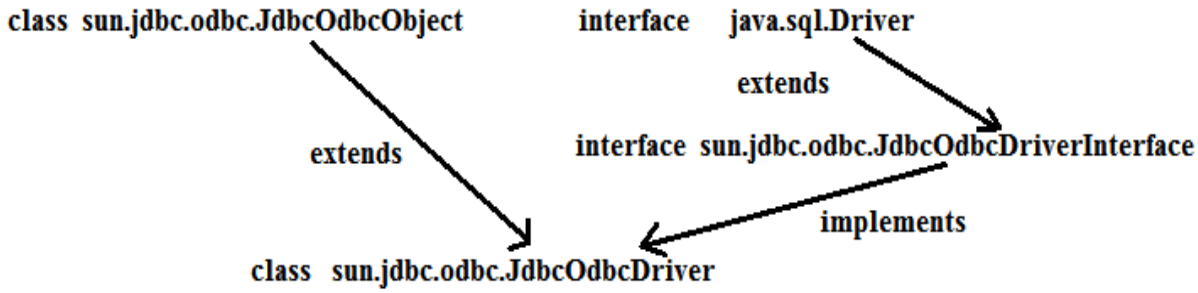
Every software specification contains an API, this API contains rules in the form of method declared in interfaces and contains guide lines in the form of concrete methods definition, defined in the class. While developing a software based on API specification, rules must be implemented and guidelines should be utilized by collecting them from the API of specification.

According to java language, API is nothing but set of classes and interfaces coming in the form of packages. The JDBC specification uses JDBC API in the form of classes and interfaces available in java.sql package.




- Implementation of all these interfaces of JDBC API is not the responsibility of programmer. It is the responsibility of Vendor Company to implement all the interfaces of JDBC API and to develop JDBC drivers.
- The inter faces of JDBC API contain methods declaration and these methods declaration represent rules of JDBC specification.
- The classes of JDBC API contain methods definition and these methods definition represent guide lines of JDBC specification.
- Every JDBC driver is a software which is a collection of multiple java classes implementing all the interfaces of JDBC API and using all the classes of JDBC API.
- JDBC driver for oracle containing lots of java classes implementing all the interfaces of java API. In all these implemented classes logic will be there to interact with oracle DB software. So all these implementation classes together works as JDBC driver for oracle.
- The API of software specification contains more interfaces rules and less classes because software specification contain more rules and less guide lines. Eg, JDBC API.
- The API of directed software technology contains more classes and less interfaces because it is the software that is ready to use and develop the application.
- JDBC is the topic of JSL module.

- Programmer never directly works with JDBC specification. He always work with the JDBC specification based JDBC drivers to make his java application communicate with db software.
- JDBC, JNDI are the specifications given by Sun Microsystems in JSE module, so it is better not to call them as JSE technologies. JNDI – java naming directory inter faces.
- Every JDBC drivers is identifies db with its JDBC driver class name.
- Sun Microsystems supplied a built in JDBC driver along with JDK software installation. The driver class name of this JDBC driver is "sun.jdbc.odbc.JdbcOdbcDriver".




- Every vender supplied JDBC driver contains one unique name as JDBC driver class name for identification purpose.
- Working with JDBC driver class name in java application is nothing but activating the related JDBC driver as a bridge to communicative with DB software.
- Most of ODBC drivers for different DB software are coming along with windows OS installation.
- Every ODBC driver is identified with its DSN creating DSN for ODBC driver is the responsibility of programmer.
- There are 3 types of DSNs.
  - ✓ User DSN (visible only to the current windows user)
  - ✓ System DSN visible to all the windows user of the system)
  - ✓ File DSN (Shareable in multiple computers connected in the n/w)

If we are working with personal local oracle on our computer value for server is optional. If you are working with remote and centralized oracle software installation pass. Host/SID name must be given by collecting it from oracle DBA.

There are 4 mechanism to develop jdbc drivers based on jdbc specification they are,

1>Type-1, 2>Type-2, 3>Type-3, 4>Type-4

Type-1 mechanism based jdbc driver is designed to directly interact with odbc driver, which in turn interacts with DB software. The SUN Microsystem supplied built in jdbc driver, based on Type-1 mechanism in Jdk software, which uses odbc driver to interact with DB software. Type-1 mechanism based jdbc driver contains native code. So it can interact with odbc drivers( Native Code: Declarations are available but implementation are in other languages than java).

**Steps to develop Jdbc logic in java to interact with DB software.**

- ✓ Create an object of driver class and register that object of jdbc driver with DriverManager Service.
- ✓ Use DriverManager Service to establish connection with DB software from java apps.
- ✓ Create Statement's object.
- ✓ Use Statement object to send sql quaries to DB software and to execute the sql quaries in DB software.
- ✓ Gather result from DB software using Statement obj, process the result and display the result
- ✓ Close jdbc connection with DB software and also close other jdbc stream object

NOTE:-

1:Establishing connection with DB software is nothing but creating communication channel b/w java application & Db software.

2:Statement obj acts as courier service call vehicle to carry sql quaries from java app to jdbc software executing them in DB software and to print result DB software to java apps.

3:All jdbc obj like connection ,Statement objects and etc are stream objects. It is recommended to close these stream objects, once tasks with Db software are completed.

/*Write a jdbc like based java app to establish connection with oracle DB software by using type-1 mechanism based jdbc driver.*/

## Steps for creating/making connection and communication with database:-

**Step-1:-** Import the java.sql package which contains all the classes and interfaces for database connection.

**Step-2:-** Load the driver by the static method called forName() of Class given below
Class.forName (String Driver name) throws ClassNotFoundException;

e.g.:- Class.forName ("com.mysql.jdbc.Driver");
The forName() will load the driver, so that we can use DriverManger class in our java program. If the loading of .class file of driver will fail, then we can not use the DriverManeger Class.

Q:- In the above statement class.forName() method just loads the jdbc driver class. Then how can we say jdbc driver is registred with DriverManager service at the end of that statement execution.

ANS:- The forName() loads given jdbc Driver class, because of this loading the static block of driver class executes automatically, & this static block contains the logic to create jdbc driver class obj and logic to register that driver class obj with DriverManager service by calling DriverManager.registerDriver().

➔ The static block of com.mysql.jdbc.Driver contains following logic.

```
static{
        Driver jd=new Driver();
        try{
             DriverManager.registerDriver(jd); →Registering obj with DriverManager service
         }catch(SQLException sqlexception) {      }
 }
```

NOTE:- All vendors supplied jdbc drivers for different DB s/ws contains static blocks havi ng some jdbcDriver registration code in respective jdbcDriver classes.So, the programmer can use the above given class.forName() approach to load JdbcDriver class to register that Jdbc driver with DriverManager service.

**Step-3:-** Establish a logical connection between the java program & the database by getConnection() of DriverManeger class.
Connection con=DriverManager.getConnection (String JDBCURL, String username, String password);

e.g.: Connection con=DriverManager.getConnection ("jdbc:mysql://localhost/college", "root", "");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system", "manager");

The string parameter JDBCURL contains address, name of the driver. The second and third parameter is optional because, the getConnection() is overloaded in DriverManager class which may not accept the 2nd and 3rd parameter.

In case of Microsoft Access, we may not provide the username and password. In that case, the 2nd and 3rd parameter on the above method may be null string or we may use one argument getConnection(). The Connection is an interface that behaves like a logical connection between the java and DBMS.

**Step-4:-** Create a reference of Statement Interface by createStatment() of the Connection interface.

Statement stmt=con.createStatement ();

The Statement interface is generally used to execute SQL que3ries from the java program, such as DDL, DML, DCL, DQL, TCL. TQL etc.

The Statement interface has some methods to execute SQL but mostly two methods are used for DML, SQL and for query SQL.

**public ResultSet executeQuery (String select_quries)**

This method will execute the select qry that returns the no. of rows of a table. Hence to hold the no. of records or rows of the table, the ResultSet interface is used.

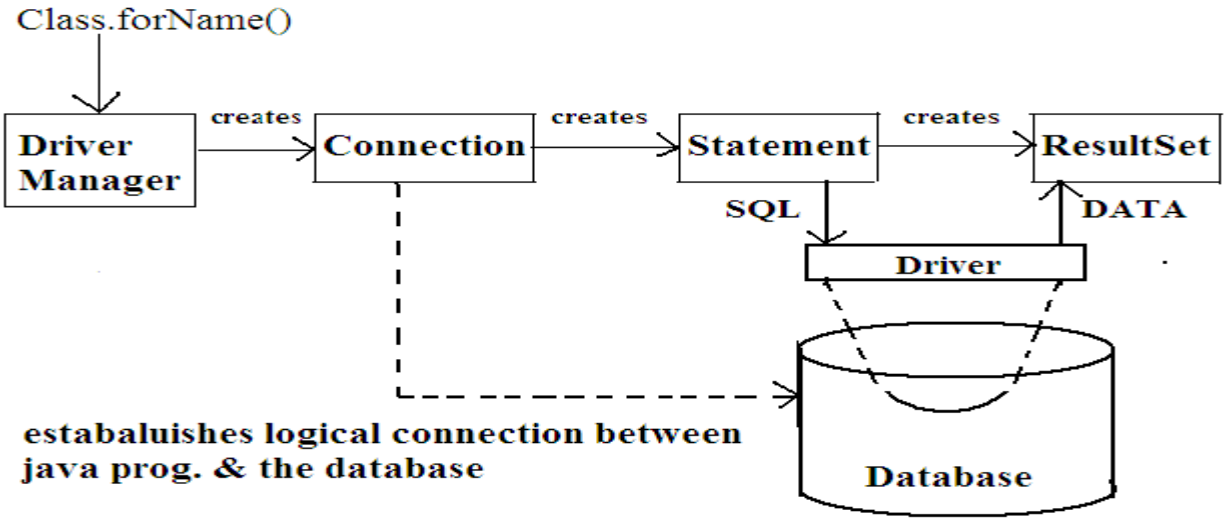
**public int executeUpdate( String inert/update/delete/drop/create/alter_ queries)**

This method will execute DML statement like insert, update, delete, drop, create, alter etc. Hence it will return the no. of rows of the table affected as integer.

**Step-5:-** Once all the job will complete, we have to close Connection by following statement

con.close ();

**Interaction of JDBC API with DBMS**:-




| **Stud Table :** | | | Emp Table: | | | Dept Table: | |
|---|---|---|---|---|---|---|---|
| name | Varchar2(30) | | Eno | Int/number | | deptno | Int/number |
| roll | Int/number | | Enam | Varchar2(30) | | dnam | Varchar2(30) |
| sem | Varchar2(5) | | Dob | Date | | dphone | Varchar2(50) |
| mark | Float/number(7,2) | | Deptno | Int/number | | | |
| | | | Sal | Float/number(7,2) | | | |

**Using type-4 driver of Mysql :**

```
import  java.sql.*;
class  Mysqlconn {
 public static void main(String args[]) throws Exception {
  Class.forName("com.mysql.jdbc.Driver");
  Connection con = DriverManager.getConnection( " jdbc:mysql://localhost/college", "root", "");
```

```
 Statement st = con.createStatement();
 ResultSet rs=st.executeQuery("select   * from  stud");
 while(rs.next())
   System.out.println( rs.getString(1) +"\t" + rs.getInt(2)+"\t"+ rs.getString(3)+ "\t" +  rs.getFloat(4));
con.close();
}};
```

The jdbcurl of getConnection() is as follows :
jdbc:mysql://localhost/college
where
jdbc is protocol
mysql is subprotocol
localhost is the ipaddress of the machine in which mysql database is installed.
college in the database in which stud table is present.

The above driver class is present in this "mysql-connector-java-5.1.5-bin.jar" jar file hence, copy this file to the same location where you have stored the current folder.

**Compile as:**

javac -cp mysql-connector-java-5.1.5-bin.jar;. Mysqlconn.java

**Run as:**

java -cp mysql-connector-java-5.1.5-bin.jar;. Mysqlconn

## Using type-4(thin) driver of oracle

```
import   java.sql.*;
class   OraConn {
public static void main(String args[]) throws  Exception {
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system", "manager");
Statement stmt= con.createStatement();
ResultSet  rs=stmt.executeQuery("select  * from stud");
while(rs.next()){
System.out.println(rs.getString("name")+" \t"+rs.getInt("roll") + "\t"+ rs.getString("sem")+"\t"+
rs.getFloat("mark"));
 }
con.close();
}};
```

--> store the file with .java extn
--> copy the classes111.jar(for oracle 8i) classes12.jar(for oracle 9i) or ojdbc6.jar/ojdbc14.jar(for oracle 10g/11g) from oracle installation folder( ex: if oracle is installed under D:\ then the .jar file avaialable in D:\oracle\ora90\jdbc\lib) to the same folder where the .java file is stored.
--> compile the above file from dos as

```
javac  -cp  classes12.jar;. OraConn.java
```

-->Execute as

```
java  -cp  classes12.jar;. OraConn
```

The subname in jdbcurl of getConnection method must be given as follows
thin:@localhost:1521:XE
Here,
thin is the type-IV driver of oracle
@localhost refers to IPAddress of the local machine, but if oracle DB is installed in different machine then we should provide ipaddress something like 192.168.10.4 instead of localhost, but the @ symbol must precede the ipaddress.
1521 is the default port number on which the type-iv driver(thin) or type-ii driver(oci) gnerally runs.

XE is the host string(or SID- System Identifier) of this machine. To know the SID, open control panel - adminstrative tool - services. In services u'll get OracleServiceZZZ, ZZZ will be the SID.

**Data fetching methods of ResultSet:-** These method of the ResultSet are used to fetch the data from the different columns of the tables.

1. public String **getString**(String columnName/ int column_no);
2. public int **getInt**(String columnName/ int column_no);
3. public byte **getByte**(String columnName/ int column_no);
4. public short **getShort**(String columnName/ int column_no);
5. public long **getLong**(String columnName/ int column_no);
6. public float **getFloat**(String columnName/ int column_no);
7. public double **getDouble**(String columnName/ int column_no);
8. public Date **getDate**(String columnName/ int column_no);
9. public Date **getObject**(String columnName/ int column_no);

Note:- The above column index/no.s must refer the column no. of select qry, not that of database

```
import java.sql.*;
class Data3{
public static void main(String args[])throws Exception {
Class.forName("com.mysql.jdbc.Driver");
Connection con =  DriverManager.getConnection("jdbc:mysql://localhost/college,"root","");
Statement st=con.createStatement();
ResultSet rs=st.executeQuery("select a.*,b.* from emp a, dept b where a.deptno=b.deptno");
while(rs.next())
System.out.println(rs.getInt(1) +"\t" +rs.getString(2)+"\t"+rs.getDate(3)
+"\t"+rs.getFloat(5)+"\t"+rs.getString(7)+"\t"+rs.getString(8) ));
con.close();
}};
```

**Inserting a Record :**

```
import java.sql.*;
import java.util.Scanner;
class Data3{
public static void main(String args[])throws Exception {
Class.forName("com.mysql.jdbc.Driver");
Connection con =  DriverManager.getConnection("jdbc:mysql://localhost/college,"root","");
Statement st=con.createStatement();
Scanner s=new Scanner(System.in);
System.out.println("enter name");
String   na=s.next();
System.out.println("enter roll");
int   ro =s.nextInt();
System.out.println("enter sem");
String   se = s.next();
System.out.println("enter sem");
float  ma=s.nextFloat();
String qry="insert into stud values('"+na+"', "+ro+",'"+se+"', "+ma+")";
System.out.println(qry);
int z= st.executeUpdate(qry);
if(z>0)
System.out.println("Data inserted sucessfully");
con.close();
}};
```

**Updating a Record :**

```
import java.sql.*;
import java.util.Scanner;
class Data3{
public static void main(String args[])throws Exception {
Class.forName("com.mysql.jdbc.Driver");
Connection con =  DriverManager.getConnection("jdbc:mysql://localhost/college,"root","");
Statement st=con.createStatement();
Scanner s=new Scanner(System.in);
System.out.println("enter name to update");
String   na=s.next();
System.out.println("enter roll whose name, sem mark to be updated");
int   ro =s.nextInt();
System.out.println("enter sem to update");
String   se = s.next();
System.out.println("enter sem to update");
float  ma=s.nextFloat();
String qry="update stud set name='"+na+"', sem='"+se+"', mark="+ma+" where roll="+ro;
System.out.println(qry);
int z= st.executeUpdate(qry);
if(z>0)
System.out.println("Data updated sucessfully");
con.close();
}};
```

**Deleting a Record :**

```
import java.sql.*;
import java.util.Scanner;
class Data3{
public static void main(String args[])throws Exception {
Class.forName("com.mysql.jdbc.Driver");
Connection con =  DriverManager.getConnection("jdbc:mysql://localhost/college,"root","");
Statement st=con.createStatement();
Scanner s=new Scanner(System.in);
System.out.println("enter roll to delete");
int   ro =s.nextInt();
String qry="delete from stud  where roll="+ro;
System.out.println(qry);
int z= st.executeUpdate(qry);
if(z>0)
System.out.println("Data deleted sucessfully");
con.close();
}};
```

**Updatable methods of ResultSet:-**

```
ResultSet rs=st .executeQuery("select  * from stud")
while(rs.next()){
        int ro=rs.getInt("roll");
        flaot ma=rs.getFloat("mark");
        if(ma<30)
                st.executeUpdate("update stud set mark=mark+10 where roll="+ro);
```

```
}
con.close();
-----------------
```

When the above if condition in the while loop satisfies for the first time. The executeUpdate() method will work ,but rs will be closed ,because con,st,rs all are references, because Connection, Statement, ResultSet are interfaces, and interface cannot have its own object, rather it can hold/point to an object of it's derived class . When con getting the data from object of DriverManager, st is getting the data from con and automatically rs getting data from st. Therefore if st will execute the executeUpdate() method then rs will never get any record from st, during this time rs will be closed. We can adopt following mechanism to overcome this situation :

→Therefore ,if we want to manipulate the database while fetching the data then we have to fetch the data and store it in a 2D object array. After that we can iterate the object array and during this time we can manipulate the database by executeUpdate() method.

→We can also create multiple st or con ,but it is not possible in our program .We have to implement connection poolling or connection caching mechanism to get multiple connections.

→Apart from the above mechanism we can use the following **updatable method of ResultSet** to manipulate the data while fetching or browsing the database.

1. public void **updateString**(String columnName/ int column_no, String val);
2. public void **updateInt**(String columnName/ int column_no, int val);
3. public void **updateByte**(String columnName/ int column_no, byte val);
4. public void **updateShort**(String columnName/ int column_no, short val);
5. public void **updateLong**(String columnName/ int column_no, long val);
6. public void **updateFloat**(String columnName/ int column_no, float val);
7. public void **updateDouble**(String columnName/ int column_no, double val);
8. public void **updateDate**(String columnName/ int column_no, Date val);

Note:- The above update methods will work in updatable methods of ResultSet.

1. public void **deleteRow()**
2. public void **insertRow()**
3. public void **updateRow()**
4. public void **moveToInsertRow()**
5. public void **refreshRow()**

**public void updateRow() :-** This method update the current record pointed by ResultSet ,but value to individual coulmn will be given by above 8 update method.

let's update mark and semester of current record.

```
rs.updateFloat("mark",70);
rs.updateString("sem","VIII");
```

similarly we can specify values for other columns then we can update the record by :

```
rs.updateRow();
```

**publicc void insertRow()** :- This method will insert a new record in the current position of the ResultSet,but values for individual fiels will be given by the above 8 methods, then we can use insertRow().

```
rs.updateString("name", "Kalia");
rs.updateInt("roll", 111);
rs.updateString("sem", "vi");
rs.updateFloat("mark",1100.0F);
rs.insertRow();
```

**public void moveToInsertRow():-** If we us this method before **insertRow()** method then a new record will be inserted at the end of all existing record.

```
rs.moveToInsertRow();
rs.updateString("name", "Kalia");
rs.updateInt("roll", 111);
rs.updateString("sem", "vi");
rs.updateFloat("mark",1100.0F);
rs.insertRow();
```

**public void deleteRow():- I**t will delete the current record pointed by ResultSet.

## Scrollable methods of ResultSet : -

1. public boolean **next();**
2. public boolean **first();**
3. public boolean **last();**
4. public boolean **previous();**
5. public boolean **absolute(**int n**);** →It'll move the pointer of ResultSet to nth record from the beginning, where n is the parameter of this method.
6. public boolean **relative(**int n**);** →It'll move the pointer of ResultSet to nth record from the current position of the ResultSet, where n is the parameter of this method.
7. public boolean **isAfterLast();**
8. public boolean **isBeforeFirst();**
9. public boolean **isClosed();**
10. public boolean **isFirst();**
11. public boolean **isLast();**
12. public void **afterLast**();→It'll simply move the pointer of ResultSet to the blank record after the last record**.**
13. public void **beforeFirst();**();→It'll simply move the pointer of ResultSet to the blank record before the first record

All these scrolling methods work in Scrollable ResultSet except the next().

## Scrolling constants of ResultSet:-

1. public static final int TYPE_FORWARD_ONLY= 1003:- The ResultSet will move only in forward direction. It is the default one(Default Scrolling Constatnt).
2. public static final int TYPE_SCROLL_INSENSITIVE= 1004 :- The ResultSet is scrollable but it is not sensitive to changes made to the data that available in the ResultSet.
3. public static final int TYPE_SCROLL_SENSITIVE= 1005:- The ResultSet is scrollable and, it is sensitive to the changes made to the data available in ResultSet.

## Updatable constants of ResultSet:-

1. public static final int CONCUR_READ_ONLY = 1007 :- The concurrency updation of a ResultSet is not possible. It is the default one(Default Updatable Constatnt).
2. public static final int CONCUR_UPDATABLE = 1008 :- The concurrency updation of a ResultSet is possible.

## Creating updatable and scrollable ResultSet:-

In order to create the scrollable and updatable ResultSet, we have to provide the following constants in the createStatement() of the Connection interface.

Ex. Statement stmt=con. createStatement (ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);

If we create the ResultSet by using the above stmt reference, then the ResultSet will be Scrollable & updateable .

Ex. ResultSet rs=stmt.executeQuery ("select * from emp");

//rs will be scrollable, it will not show changes made by others and will be updatable.

**DriverManager class:** The DriverManager class acts as an interface or a broker between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

**Commonly used methods of DriverManager class:**

| | |
|---|---|
| 1) public static void registerDriver(Driver driver): | This method is used to register the given driver with DriverManager. |
| 2) public static void deregisterDriver(Driver driver): | This method is used to deregister the given driver (drop the driver from the list) with DriverManager. |
| 3) public static Connection getConnection(String url): | This method is used to establish the connection with the specified url without username & password. |
| 4) public static Connection getConnection(String url,String userName,String password): | This method is used to establish the connection with the specified url, username and password. |

**Connection interface:** A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(),rollback() etc.

**By default, connection automatically commits the changes after executing queries.**

**Commonly used methods of Connection interface:**

1) public Statement createStatement(): creates a statement object that can be used to execute SQL queries.

2) public Statement createStatement(int resultSetType,int resultSetConcurrency):Creates a Statement object that will generate ResultSet objects with the given type and concurrency.

3) public void setAutoCommit(boolean status): is used to set the commit status.By default it is true.

4) public void commit(): saves the changes made since the previous commit/rollback permanent.

5) public void rollback(): Drops all changes made since the previous commit/rollback.

6) public void close(): closes the connection and Releases a JDBC resources immediately.

**Statement interface :** The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

**Commonly used methods of Statement interface:**

**1) public ResultSet executeQuery(String sql):** is used to execute SELECT query. It returns the object of ResultSet.

**2) public int executeUpdate(String sql):** is used to execute specified query, it may be create, drop, insert, update, delete etc.

**3) public boolean execute(String sql):** is used to execute queries that may return multiple results.

**4) public int[] executeBatch():** is used to execute batch of commands.

**Creating DSN**:-

Start→control panel→administrative tool→data source (ODBC) →choose user or system DSN tab→click add button→"create new data source" dialog box will appear **:->**

**For MS ACCESS:-** Choose Microsoft Access driver (*.mdb,*.accdb) from the list of options→Click finish button→a dialog box will appear as "ODBC Microsoft Access", put the DSN name as gaura (user defined name) in the DSN field→click the select button and a new dialog box will appear as "select database" and here select the database (i.e. MS ACCESS file from the desired location)→click all ok buttons.

**For Oracle:-** Choose "Microsoft ODBC for Oracle" from "create new data source" dialog box→Click finish button→a dialog box will appears as "Microsoft ODBC for Oracle", put gaura as DSN name, Scott as username→Click all ok buttons.

**For MySql:-** Choose "MySQL ODBC 3.51 driver" from "create new data source" dialog box→Click finish button→a dialog box will appears as "Connector/ODBC",-> In the login tab put "gaura" as DSN name, "root" as username and choose the database name as you have given →Click all ok buttons.

**String URL of getConnection ():-**

The 1st parameter of getConnection() is a String which has 3 part as given below

Jdbc : Odbc : gaura

<Protocol: sub-protocol: subname>

The protocol always remains as jdbc, because we are interacting the database from the java program. The sub-protocol is driver specific i.e. it is depended upon the type of driver. Since we have used JdbcOdbcDriver that follows odbc specification, hence the sub-protocol is Odbc. But if we are using Oracle driver then the sub-protocol may be Oracle.

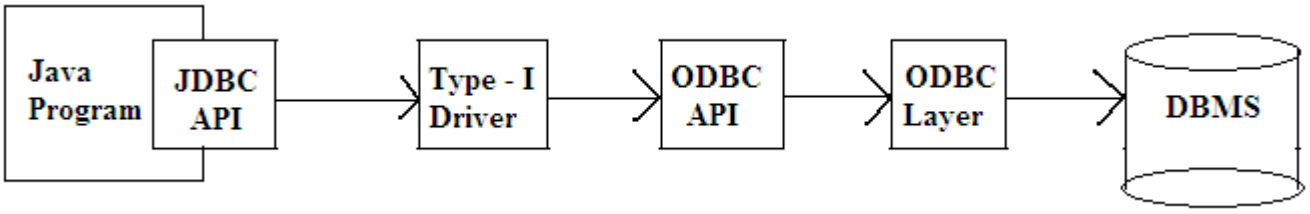
The subname always refers about a driver that is a driver specific. The subname is enough to keep all the driver information such as the name of a driver called as address of driver. In case Odbc, subname refers to DSN name, but if we use Oracle driver (thin OCI) then the subname will contain the address and the name of the driver.

**Types of Driver:-**

There are 4 types of driver used to interact with the database and they are

1. Jdbc Odbc Bridge Driver( Type - I ).
2. Partly java, partly native driver( Type - II ).
3. Intermediate Database Access Server Driver(Jdbc net driver) ( Type - III ).
4. Network Driver or pure java driver( Type - IV ).

**Type-I driver:-** This driver is given to us by Sun Microsystems(but SUN has made it obsolete from Java 1.8). It translates the jdbc API call into Odbc API call. Then it will pass the Odbc API call to the odbc layer. The API and driver have to be installed and the driver should be configured with the DSN. Each SQL statement from the java program will undergo in several layers of interaction (steps).Hence it is slower. It is not used in platform independent java application such as in applet, because it should be installed and configured with DSN.



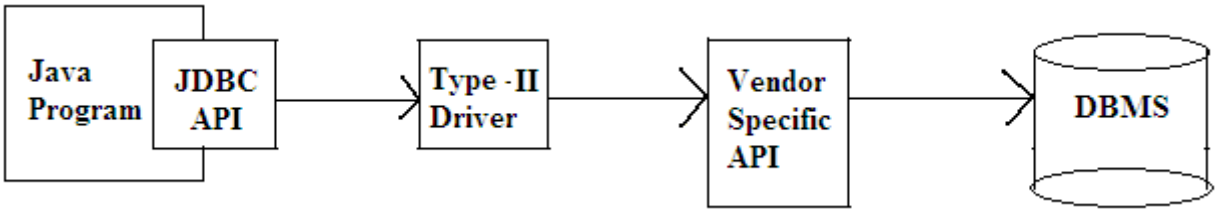

Type-1 driver can interact with any DB software by specific odbc driver. The relationship between Type-1 jdbc driver and odbc driver is 1:N, because for each DB software we need separate odbc driver, hence relationship between odbc drivers and DB software is 1:1.

All jdbc driver will be managed by DriverManager service, which is the built in service of Jdk software. It can be activated in our java app by using java.sql.DriverManager class. Programmers register the jdbc driver with

DriverManager service from java application. Registering jdbc drivers with DriverManager service is nothing but creating jdbc driver class object and placing that obj with DriverManagerService.

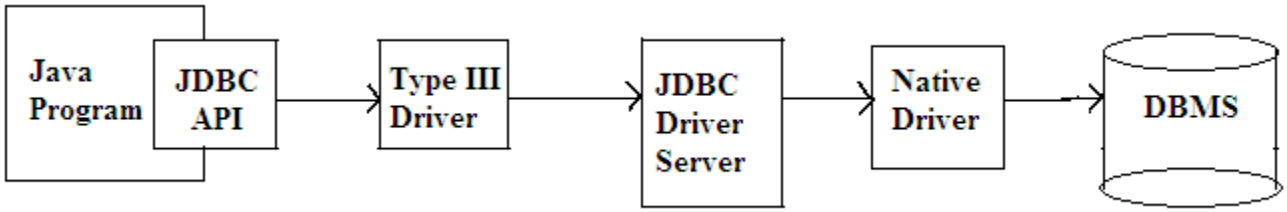
Java apps use jdbc code based persistence logic to interact with DB software and to manipulate table data(performing cord operation).

**Type-II driver:-** This driver is partially written in java, partially in vendor's specification API. It converts jdbc call to vendor's specific API or call. Then the vendor's specific call (statement) will be passed to the database. The database fetches the result and returns the result in vendor specific API to this driver. Then the driver returns the result back to the java program in the jdbc API format. This driver faster. This driver must be installed; hence it is not used in platform independent java application. Since it is faster, it is generally used in web server for connection, polling, connection caching( oci driver given by oracle is type-II driver).

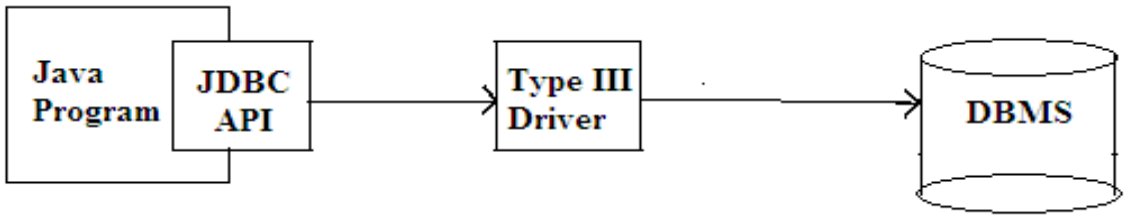



**Type-III driver:-** This driver is used to connect multiple websites with multiple clients. This driver has 2 parts i.e. (a) Client part which is interact with client, (b) Server part which interact with server.

The client part is written in java, wherein server part is written in native language. Generally it needs to be installed the server part is written in native language which interact with database hence it is not used in platform independent java application.




**Type-IV Driver:-** This driver is purely written in java. It has the ability to interact with the database directly. Hence it us faster. It uses the TCP/IP protocol to make connection with the database. No need to install driver because the address of database is generally kept with this driver. Hence it is used in platform independent java application(thin driver given by oracle is type-IV driver).




**Prepared Statement:-** The PreparedStatement is a sub interface of Statement interface. The prepareStatement() method of Connection interface is used to return the object of PreparedStatement.

| Method | Description |
|---|---|
| public void setInt(int paramIndex, int value) | sets the integer value to the given parameter index. |

| | |
|---|---|
| public void setString(int paramIndex, String value) | sets the String value to the given parameter index. |
| public void setFloat(int paramIndex, float value) | sets the float value to the given parameter index. |
| public void setDouble(int paramIndex, double value) | sets the double value to the given parameter index. |
| public int executeUpdate() | executes the query. It is used for create, drop, insert, update, delete etc. |
| public ResultSet executeQuery() | executes the select query. It returns an instance of ResultSet. |

```
import  java.sql.*;
class  Prepdemo {
public static void main(String args[]) throws Exception {
  Class.forName("com.mysql.jdbc.Driver");
  Connection con = DriverManager.getConnection( "jdbc:mysql://localhost/college", "root", "");
  PreparedStatement st = con.prepareStatement("insert into stud values(?,?,?,?)");
  st.setString(1, "Ganesha");
  st.setInt(2, 513);
  st.setString(3, "I");
  st.setFloat(4, 23.2332F);
  int  z=st.executeUpdate();
  if(z>0)
    System.out.println("data inserted successfully");
  con.close();
}};
```

The PreparedStement is a sub interface of the Statement interface. It can be created by the prepareStatement() of the Connection interface.

Whenever the Statement interface executes an SQL by the executeUpdate(), then DBMS compiles the correctness of SQL. After that it will create an execution plan, so that the execution plan will get executed on database. If we use the same SQL for several times by providing different values (insert statement in a loop), then for each SQL going to be executed by the Statement interface a new execution plan is created by the DBMS. Hence it will make the execution of SQL slower.

If we use same SQL for several times by providing different values using the PreparedStaement, then the DBMS will create execution plan only once. When the SQL will execute for the first time by the PreparedStatement then the DBMS will create the execution plan only once and stores the plan in the cache memory allotted to DBMS, when same SQL executes second time with different set of values then the DBMS will use the execution plan created during 1st execution.. Hence the execution of PreparedStatement is faster compared to Statement interfce.

The bind variable in PreparedStatement is represented by ?, and it is used like a place holder for different values. In case of Statement interface we can use normal variable to provide the value dynamically like bind variable in PreparedStatement, but in the Statement interface will accept the value dynamically for the variables to create SQL, whereas in case of PreparedStatement the SQL is created by bind variable without any value we can supplied values to the bind variables after creating the SQL statement. We have to use setXXX() of the Prepared statement. Generally the Prepared statement will execute by the execute by executeUpdate().

setXXX (int bind_var_number, YYY values )
Where XXX is wrapper class name
YYY is predefined data type value.

The 1st parameter: The bind_var_number will be 1 for 1st question mark, 2 for 2nd question mark and so on.

The 2nd parameter: The values of the bind variable must be of same data type as that of this function's wrapper class name.

## Using PreparedStatement to Fetch data :

```
import  java.sql.*;
class  Prepdemo1 {
public static void main(String args[]) throws Exception {
  Class.forName("com.mysql.jdbc.Driver");
  Connection con = DriverManager.getConnection( "jdbc:mysql://localhost/college", "root", "");
  PreparedStatement st = con.prepareStatement("select * from stud");
  ResultSet  rs=st.executeQuery();
  while(rs.next())
System.out.println(rs.getString(1)+"\t"+rs.getInt(2) +"\t"+rs.getString(3)+"\t"+rs.getFloat(4));
  con.close();
}}
```

## Creating a Stored Procedure for Stud table in oracle :

**create or replace procedure StdIns**(a IN varchar2, b IN number, c IN varchar2, d IN number ) IS
begin
insert into stud values(a,b,c,d);
end;
/
to invoke the procedure from sql prompt we have to write :
**call stdins**('laxman', 102, 'V', 654654.64);

**Callable Statement:-** This interface is used to call the **stored procedures and functions**. It is the sub interface of the PreparedStatement. It is used to execute a plsql block such as a function or procedure from a java program. We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.

Suppose you need the get the age of an employee based on the employee date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

The differences between stored procedures and functions are given below:

| Stored Procedure | Function |
|---|---|
| is used to perform business logic. | is used to perform calculation. |
| must not have the return type. | must have the return type. |
| may return 0 or more values. | may return only one values. |
| We can call functions from the procedure. | Procedure cannot be called from function. |
| Procedure supports input and output parameters. | Function supports only input parameter. |
| Exception handling using try/catch block can be used in stored procedures. | Exception handling using try/catch can't be used in user defined functions. |

How to get the instance of CallableStatement?

The prepareCall() method of Connection interface returns the instance of CallableStatement. Syntax is given below:

**public** CallableStatement prepareCall("{ call stdins(?,?, ?, ?)}");

Ex:

```
import   java.sql.* ;
import   java.util.* ;
public   class  CallProc {
   public static void main(String args[]) throws Exception {
       Class.forName("oracle.jdbc.driver.OracleDriver");
      Connection  con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system", "manager");
       Scanner  sc = new  Scanner(System.in);
       CallableStatement  cst  =  con.prepareCall("Call StdIns(?,?,?,?)");
       System.out.print("Enter Name");                    cst.setString(1 , sc.nextLine() );
       System.out.print("Enter Roll");                    cst.setInt(2 , sc.nextInt() );
       System.out.print("Enter Sem");                     cst.setString(3 , sc.next() );
       System.out.print("Enter Mark");                    cst.setString(4 , sc.nextFloat() );
       cst.execute();
       System.out.println("procedure executed");
       con.close();
}}
```

IN type parameter values are set using the set methods inherited from PreparedStatement. The type of all OUT type parameters must be registered prior to executing the stored procedure; their values are retrieved after execution via the get methods provided here.

### Creating a procedure in mysql to insert a record in stud table

```
DELIMITER $$

create procedure  simp(in a varchar(20), in b int, in  c varchar(5),in d float)
begin
  insert into stud(name, roll, sem,mark) values(a,b,c,d);
end

$$
```

to call the procedure write the following in SQL prompt

**call simp('aam', 304,'vi', 5555.55)**

**Example:**

```
import  java.sql.*;
class  Calldemo1 {
 public static void main(String args[]) throws Exception {
Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection("jdbc:mysql://localhost/college", "root", "");
CallableStatement st=con.prepareCall("call simp(?, ?, ?, ?)");

  st.setString(1,"Santoshi");
  st.setInt(2, 106);
  st.setString(3, "VI");
  st.setFloat(4, 999.32F);
```

```
  int  z=st.executeUpdate();
      System.out.println("procedure completed successfully");
   con.close();
}};
```

**Creating a stored procedure in mysql to retrieve the data**

```
DELIMITER $$

create procedure st(out a VARCHAR(20), IN b INT, OUT c VARCHAR(5), OUT d FLOAT)

BEGIN
  DECLARE count INT;
  SELECT count(*), NAME, SEM, MARK FROM  STUD WHERE ROLL=B;
        IF count > 0  THEN
                SET a = NAME;
                SET c = SEM;
                SET d = MARK;
        END IF;
END

$$
```

**Example :**

```
import java.sql.*;
class Callable2 {
public static void main(java.lang.String[] args) {
 try {
      Class.forName("com.mysql.jdbc.Driver");
      Connection con = DriverManager.getConnection("jdbc:mysql://localhost/college", "root","");
      CallableStatement cs = con.prepareCall("call  ST(?, ?, ?,?)");

          /* All input parameters must be set and all output parameters must  be registered. */
      cs.registerOutParameter(1, Types.VARCHAR);
      cs.setInt(2, 104);  //assume 104 is the roll no. of stud table
      cs.registerOutParameter(3, Types.VARCHAR);
      cs.registerOutParameter(4, Types.FLOAT);

      // Run the procedure
       cs.execute();

      // Get the values in ResultSet.
       ResultSet rs = cs.getResultSet();
       if(rs.next()){
       System.out.println("The no. of record :  " + rs.getString(1));
       System.out.println("The  name:  " + rs.getString(2));
       System.out.println("The sem :  " + rs.getString(3));
       System.out.println("The  mark :  " + rs.getFloat(4));
       }
       con.close();   // close the Connection object.
```

```
 } catch (Exception e) {
       System.out.println("Something failed..");
       System.out.println("Reason: " + e.getMessage());
       e.printStackTrace();
}
}};
```

In this example, we are calling the ST procedure that receives one input(i.e. roll) and returns the name, sem & mark of the given roll number. Here, we have used the **registerOutParameter** method of CallableStatement interface, that registers the output parameter with its corresponding type. It provides information to the CallableStatement about the type of result being displayed. The **Types** class defines many constants such as INTEGER, VARCHAR, FLOAT, DOUBLE, BLOB, CLOB etc.

### Creating a function in oracle to get the value :

```
create or replace function sumIt(x  in number, y in number) return number
is
  temp number(8);
begin
  temp := x+y;
  return temp;
end;
/
```

Example :

```
import java.sql.*;
class Callable3 {
public static void main(java.lang.String[] args) throws Exception {
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system", "gitesh");

 CallableStatement cs=con.prepareCall("{ ?=call sumIt(?,? ) }");
 cs.registerOutParameter(1, Types.INTEGER);
 cs.setInt(2, 15);
 cs.setInt(3, 20);

 // Run the procedure
 cs.execute();

 // Get the result from cs.
  System.out.println("The value :  " + cs.getInt(1));
   con.close();   // close the Connection object.
}};
```

**ResultSetMetaData:**- The information about the data is known as metadata. This interface provides the information about ResultSet such as the no. of columns available in Resultset, column name, column's data type.

**Commonly used methods of ResultSetMetaData interface**

| Method | Description |
|---|---|
| public int getColumnCount()throws SQLException | it returns the total number of columns in the ResultSet object. |
| public String getColumnName(int index)throws SQLException | it returns the column name of the specified column index. |
| public String getColumnTypeName(int index)throws SQLException | it returns the column type name for the specified index. |
| public String getTableName(int index)throws SQLException | it returns the table name for the specified column index. |

```
import java.sql.*;
class MetaData1 {
   public static void main(String args[]) throws Exception {
       Class.forName("oracle.jdbc.driver.OracleDriver");
       Connection con = DriverManager. getConnection("jdbc:oracle:thin:@localhost:1521:xe","system",
"manager");
       System.out.println("Connected...");
       Statement st = con.createStatement();
       ResultSet rs = st.executeQuery( "select * from emp");
       ResultSetMetaData md= rs.getMetaData();
       int x = md.getColumnCount();
       for( int i = 1 ; i <= x; i++)
              System.out.println(md.getColumnName(i) + "\t" +md.getColumnTypeName(i) );
       con.close();
 }} ;
```

**Display records of any table :**

```
import  java.sql.*;
import  javax.swing.*;
import  java.awt.*;
import  java.awt.event.*;
class Meta1 extends  JFrame implements ActionListener {
  JTable tab;
  JButton b1;
  JTextField t1;
  JScrollPane sp ;

public Meta1() {
  Panel p1 = new Panel();
  p1.add(t1= new JTextField(10));
  p1.add(b1 = new JButton("Ok"));
  b1.addActionListener(this);
  t1.setFont(new Font("Asdf", Font.BOLD,30));
  b1.setFont(new Font("Asdf", Font.BOLD,30));
  add("South", p1);
  setSize(1000,600);
```

```
  setVisible(true);
}

public void actionPerformed(ActionEvent ae) {
try{
 sp.remove(tab);
 pack();
}catch(Exception eeE){   }
try{
Class.forName("com.mysql.jdbc.Driver").newInstance();
Connection con=DriverManager.getConnection("jdbc:mysql://localhost/college", "root" , "");
Statement st= con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
ResultSet rs=st.executeQuery("select * from "+t1.getText());
ResultSetMetaData rd=rs.getMetaData();
int col=rd.getColumnCount();
String na[]=new String[col];
for(int i=1; i<= col ; i++)
  na[i-1]=rd.getColumnName(i);

rs.last();
int row=rs.getRow();
rs.beforeFirst();
int k=0;
Object ar[][]= new Object[row][col];
while(rs.next()) {
   for(int i=1; i<= col ; i++)
      ar[k][i-1] = rs.getObject(i);
   k++;
}

tab = new JTable(ar,na);
tab.setRowHeight(35);
tab.setFont(new Font("Asdf", Font.BOLD,30));
sp= new JScrollPane(tab);
add("Center", sp);
pack();
con.close();
}catch(Exception e){  }
}
 public static void main(String args[]) throws Exception {
   new  Meta1();
}};
```

**DatabaseMetaData interface:** DatabaseMetaData interface provides methods to get meta data of a database such as database product name, database product version, driver name, name of total number of tables, name of total number of views etc.

- public String getDriverName()throws SQLException: it returns the name of the JDBC driver.
- public String getDriverVersion()throws SQLException: it returns the version number of the JDBC driver.
- public String getUserName()throws SQLException: it returns the username of the database.
- public String getDatabaseProductName()throws SQLException: it returns the product name of the

database.

- public String getDatabaseProductVersion()throws SQLException: it returns the product version of the database.
- public ResultSet getTables(String catalog, String schemaPattern, String tableNamePattern, String[] types)throws SQLException: it returns the description of the tables of the specified catalog. The table type can be TABLE, VIEW, ALIAS, SYSTEM TABLE, SYNONYM etc.

The **getMetaData()** method of Connection interface **returns** the object of **DatabaseMetaData**

```
import java.sql.*;
class Dbmd{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system", "manager");
DatabaseMetaData dbmd=con.getMetaData();

System.out.println("Driver Name: "+dbmd.getDriverName());
System.out.println("Driver Version: "+dbmd.getDriverVersion());
System.out.println("UserName: "+dbmd.getUserName());
System.out.println("Database Product Name: "+dbmd.getDatabaseProductName());
System.out.println("Database Product Version: "+dbmd.getDatabaseProductVersion());
con.close();
}catch(Exception e){ System.out.println(e);}

}}
```

**DatabaseMetaData interface that prints total number of tables :**

```
import java.sql.*;
class Dbmd2{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection  con=DriverManager.getConnection( "jdbc:oracle:thin:@localhost:1521:xe", "system","manager");
DatabaseMetaData dbmd=con.getMetaData();
String table[]={"TABLE"};
ResultSet rs=dbmd.getTables(null,null,null,table);
while(rs.next()){
System.out.println(rs.getString(3));
}
con.close();
}catch(Exception e){ System.out.println(e);}
}}
```

**DatabaseMetaData interface that prints total number of views :**

```
import java.sql.*;
class Dbmd3{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system","manager");
DatabaseMetaData dbmd=con.getMetaData();
```

```
String table[]={"VIEW"};
ResultSet rs=dbmd.getTables(null,null,null,table);
while(rs.next()){
System.out.println(rs.getString(3));
}
con.close();
}catch(Exception e){ System.out.println(e);}
}}
```

**Storing image in Oracle database :** We can store images in the database in java by the help of **PreparedStatement** interface. The **setBinaryStream()** method of PreparedStatement is used to set Binary information into the parameterIndex.

The syntax of setBinaryStream() method is given below:
1) **public void** setBinaryStream(**int** paramIndex,InputStream stream) **throws** SQLException
2) **public void** setBinaryStream(**int** paramIndex,InputStream stream,**long** length) **throws** SQLException

For storing image into the database, BLOB (Binary Large Object) datatype is used in the table.

```
CREATE TABLE  IMGTABLE  (
      NAME  VARCHAR2(40),
      PHOTO BLOB
  )
```

Let's write the jdbc code to store the image in the database. Here we are using aa.jpg which is available in the same location where the program is stored. You can change it according to the image location.

```
import java.sql.*;
import java.io.*;
public class InsertImage {
public static void main(String[] args) {
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system", "manager");
PreparedStatement ps=con.prepareStatement("insert into imgtable values(?,?)");
ps.setString(1,"soma");

FileInputStream fin=new FileInputStream("aa.jpg");
ps.setBinaryStream(2,fin,fin.available());
int i=ps.executeUpdate();
System.out.println(i+" records affected");

con.close();
}catch (Exception e) {e.printStackTrace();}
}}
```

If you select the table, record is stored in the database but image will not be shown.

**Example to retrieve image from Oracle database :** By the help of **PreparedStatement** we can retrieve and store the image in the database. The **getBlob()** method of PreparedStatement is used to get Binary information, it returns the instance of Blob. After calling the **getBytes()** method on the blob object, we can get the array of binary information that can be written into the image file.

1. **public** Blob getBlob()**throws** SQLException
2. **public byte**[] getBytes(**long** pos, **int** length)**throws** SQLException

Now let's write the code to retrieve the image from the database and write it into the directory so that it can be displayed. In AWT, it can be displayed by the Toolkit class. In servlet, jsp, or html it can be displayed by the img tag.

```
import java.sql.*;
import java.io.*;
public class RetrieveImage {
public static void main(String[] args) {
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system",  "manager");

PreparedStatement ps=con.prepareStatement("select * from imgtable");
ResultSet rs=ps.executeQuery();
if(rs.next()){//now on 1st row

Blob b=rs.getBlob(2);//2 means 2nd column data
byte barr[]=b.getBytes(1,(int)b.length());//1 means first image

FileOutputStream fout=new FileOutputStream("pagal.jpg");
fout.write(barr);

fout.close();
}//end of if
System.out.println("ok");
con.close();
}catch (Exception e) {e.printStackTrace();  }
}}
```

**Example to store file in Oracle database:** The setCharacterStream() method of PreparedStatement is used to set character information into the parameterIndex.

1) public void setBinaryStream(int paramIndex,InputStream stream)throws SQLException
2) public void setBinaryStream(int paramIndex,InputStream stream,long length)throws SQLException

For storing file into the database, CLOB (Character Large Object) datatype is used in the table.

```
CREATE TABLE  FILETABLE  (
   IDS NUMBER,
     NAME CLOB  )
/

import java.io.*;
import java.sql.*;
public class StoreFile {
public static void main(String[] args) {
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
PreparedStatement ps=con.prepareStatement("insert into filetable values(?,?)");

File f=new File("a.txt");
FileReader fr=new FileReader(f);
```

```
ps.setInt(1,101);
ps.setCharacterStream(2,fr,(int)f.length());
int i=ps.executeUpdate();
System.out.println(i+" records affected");

con.close();
}catch (Exception e) {e.printStackTrace();}
}}
```

**Example to retrieve file from Oracle database:** The getClob() method of PreparedStatement is used to get file information from the database.

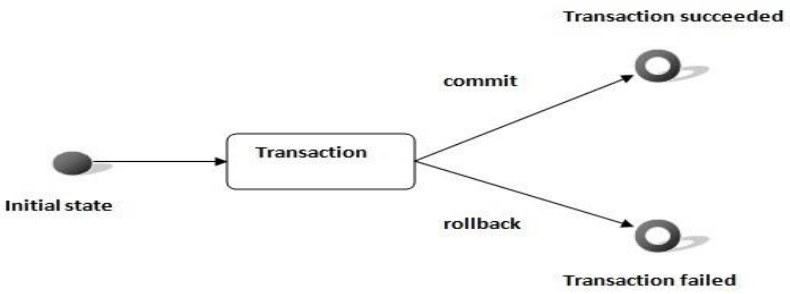
```
public Clob getClob(int columnIndex){  }
```

The example to retrieve the file from the Oracle database is given below.

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

public class RetrieveFile {
public static void main(String[] args) {
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
PreparedStatement ps=con.prepareStatement("select * from filetable");
ResultSet rs=ps.executeQuery();
rs.next();//now on 1st row

Clob c=rs.getClob(2);
Reader r=c.getCharacterStream();

FileWriter fw=new FileWriter("bb.txt");

int i;
while((i=r.read())!=-1)
fw.write((char)i);

fw.close();
con.close();

System.out.println("success");
}catch (Exception e) {e.printStackTrace();  }
}}
```

**Transaction Management**:- The transaction management means the entire job on the database must remain atomic i.e. the transaction should be atomic. A transaction will become atomic if it allows us to perform all the transaction i.e. SQL statement to execute once.
By default, all the SQL statement in java are auto commit. To achieve the transaction management, we have to make the autocommit functionality or SQL statements as false by using the followings method of the Connection interface.

Transaction represents **a single unit of work**. The ACID properties describes the transaction management well. ACID stands for Atomicity, Consistency, isolation and durability. **Atomicity** means either all successful or none. **Consistency** ensures bringing the database from one consistent state to another consistent state. **Isolation** ensures that transaction is isolated from other transaction. **Durability** means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.




| Method | Description |
|---|---|
| void setAutoCommit(boolean status) | It is true bydefault means each transaction is committed bydefault. |
| void commit() | commits the transaction. |
| void rollback() | cancels the transaction. |

```
import  java.sql.*;
import  java.util.*;
class Trans {
 public static void main(String args[]) throws Exception {
Connection con=null;
try{
  Class.forName("com.mysql.jdbc.Driver").newInstance();
con=DriverManager.getConnection("jdbc:mysql://localhost/college", "root" , "");
Statement st = con.createStatement();
Scanner  s =new  Scanner(System.in);
con.setAutoCommit(false);
for(int i=0;i<2;i++) {
System.out.println("Enter name");
String   na=s.next();
System.out.println("Enter roll");
int   ro=s.nextInt();
System.out.println("Enter sem");
String   se=s.next();
System.out.println("Enter mark");
float  ma=s.nextFloat();
String qry="insert into stud values('"+na+"', "+ro+",'"+se+"', "+ma+")";
System.out.println(qry);
st.executeUpdate(qry);
}
System.out.println("Enter roll  to update");
int  r=s.nextInt();
System.out.println("Enter mark to update");
float  m=s.nextFloat();
```

```
st.executeUpdate("update stud set mark = "+m+" where roll="+r);
System.out.println("Mark updated sucessfully");
con.commit();
}catch(Exception e){
  try{
     con.rollback();
   }catch(Exception ee){    }
  System.out.println(e);
}
try{
     con.close();
}catch(Exception ee){  }
}};
```

**Using ArrayList in JDBC :**

```
import  java.sql.*;
import  java.util.*;
class  Hash2 {
 public static void main(String args[]) throws Exception {
  Class.forName("com.mysql.jdbc.Driver");
  Connection con = DriverManager.getConnection( "jdbc:mysql://localhost/college", "root", "");
  Statement st = con.createStatement();
  ResultSet rs=st.executeQuery("select   * from  stud");
 Hashtable h= new Hashtable();
  while(rs.next()) {
    String na= rs.getString("name");
    int ro=rs.getInt("roll");
    ArrayList a1= new ArrayList();
    a1.add(na);
    a1.add(rs.getString("sem"));
    a1.add(rs.getFloat("mark"));
    h.put(ro, a1);
  }
 con.close();
 String z="y";
 Scanner s =new Scanner(System.in);
 while( z.equalsIgnoreCase("Y")) {
   System.out.println("enter roll");
   int k=s.nextInt();
   ArrayList a2=(ArrayList) h.get(k);
   System.out.println(a2.get(0) + " : "+ a2.get(1) + " : " + a2.get(2));
   System.out.println("continue(y) : ");
   z=s.next();
  }
}};
```

**Batch Update (It is driver dependant) :**

Instead of executing a single query, we can execute a batch (group) of queries. It makes the performance fast. The java.sql.Statement and java.sql.PreparedStatement interfaces provide methods for batch processing. Advantage of Batch Processing is **Fast Performance**

**Methods of Statement interface**

The required methods for batch processing are given below:

| Method | Description |
|---|---|
| void addBatch(String query) | It adds query into batch. |
| int[] executeBatch() | It executes the batch of queries. |

```
import  java.sql.*;
class Ins {
 public static void main(String args[]) throws Exception {
  Class.forName("com.mysql.jdbc.Driver").newInstance();
Connection con=DriverManager.getConnection("jdbc:mysql://localhost/college", "root" , "");
Statement st = con.createStatement();
for(int i=400, j=65; i<426 ; i++, j++){
  String qry="insert into stud values('"+ (char)j+"', "+i+", 'vii', "+ (i*10) +")";
  st.addBatch(qry);
}
st.executeBatch();
con.close();
}};
```

**JDBC RowSet :** The instance of **RowSet** is the java bean component because it has properties and java bean notification mechanism. It is introduced since JDK 5. It is the wrapper of ResultSet. It holds tabular data like ResultSet but it is easy and flexible to use.

The implementation classes of RowSet interface are as follows:

- JdbcRowSet
- CachedRowSet
- WebRowSet
- JoinRowSet
- FilteredRowSet

Let's see how to create and execute RowSet.

```
JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
rowSet.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
rowSet.setUsername("system");
rowSet.setPassword("oracle");

rowSet.setCommand("select * from emp400");
rowSet.execute();
```

*It is the new way to get the instance of JdbcRowSet since JDK 7.*

**Advantage of RowSet :** The advantages of using RowSet are given below:

1. It is easy and flexible to use
2. It is Scrollable and Updatable bydefault

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

```
import javax.sql.RowSetEvent;
import javax.sql.RowSetListener;
import javax.sql.rowset.JdbcRowSet;

import javax.sql.rowset.RowSetProvider;

public class RowSetExample {
public static void main(String[] args) throws Exception {
Class.forName("oracle.jdbc.driver.OracleDriver");
        //Creating and Executing RowSet
        JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
        rowSet.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
        rowSet.setUsername("system");
        rowSet.setPassword("manager");

        rowSet.setCommand("select * from stud");
        rowSet.execute();

        while (rowSet.next()) {
                // Generating cursor Moved event
                System.out.println("Name: " + rowSet.getString(1));
                System.out.println("Roll: " + rowSet.getInt(2));
                System.out.println("Sem: " + rowSet.getString(3));
                System.out.println("Sem: " + rowSet.getFloat(4));
        }

}}
```

**Full example of Jdbc RowSet with event handling :** To perform event handling with JdbcRowSet, you need to add the instance of**RowSetListener** in the addRowSetListener method of JdbcRowSet.

The RowSetListener interface provides 3 method that must be implemented. They are as follows:

1) public void cursorMoved(RowSetEvent event);

2) public void rowChanged(RowSetEvent event);

3) public void rowSetChanged(RowSetEvent event);

Let's write the code to retrieve the data and perform some additional tasks while cursor is moved, cursor is changed or rowset is changed. The event handling operation can't be performed using ResultSet so it is preferred now.

```
import java.sql.*;
import javax.sql.RowSetEvent;
import javax.sql.RowSetListener;
import javax.sql.rowset.JdbcRowSet;
import javax.sql.rowset.RowSetProvider;

public class RowSetExample1 {
public static void main(String[] args) throws Exception {
Class.forName("oracle.jdbc.driver.OracleDriver");

//Creating and Executing RowSet
JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
rowSet.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
```

```
rowSet.setUsername("system");
rowSet.setPassword("manager");

rowSet.setCommand("select * from stud");
rowSet.execute();

//Adding Listener and moving RowSet
rowSet.addRowSetListener(new MyListener());

 while (rowSet.next()) {
                    // Generating cursor Moved event
                    System.out.println("Name: " + rowSet.getString(1));
                    System.out.println("Roll: " + rowSet.getInt(2));
                    System.out.println("Sem: " + rowSet.getString(3));
                    System.out.println("Sem: " + rowSet.getFloat(4));
}

}}

class MyListener implements RowSetListener {
  public void cursorMoved(RowSetEvent event) {
                            System.out.println("Cursor Moved...");
  }
 public void rowChanged(RowSetEvent event) {
                            System.out.println("Cursor Changed...");
 }
 public void rowSetChanged(RowSetEvent event) {
                            System.out.println("RowSet changed...");
}}
```

**Retrieving Data from from Book1.xlsx file :**

Create an excel sheet store it as Book1.xlsx and store it under d:/datas folder. Create a table in Sheet1 or Sheet2 whatever you like.

```
import java.sql.*;

/* A demo show how to use Statement.executeQuery(sql). */
public class Example1 {
    public static void main(String argv[]) {
        try {
            Class.forName("com.hxtt.sql.excel.ExcelDriver").newInstance();

            //Please see Connecting to the Database section of Chapter 2. Installation in Development Document
            //Please change "demodata" to your database directory
            String url = "jdbc:Excel:///D:/datas/Book1.xlsx";

            //Please replace with your query statement.
            //You should read SQL syntax in HXTT Excel Development Document
            String sql = "select * from Sheet1";
            Connection con = DriverManager.getConnection(url, "", "");

            Statement stmt = con.createStatement();
            stmt.setFetchSize(10);

            ResultSet rs = stmt.executeQuery(sql);

            ResultSetMetaData resultSetMetaData = rs.getMetaData();
            int iNumCols = resultSetMetaData.getColumnCount();
            for (int i = 1; i <= iNumCols; i++) {
               System.out.println(resultSetMetaData.getColumnLabel(i)
```

```
                    + "  " +
                    resultSetMetaData.getColumnTypeName(i));
        }

        Object colval;
        while (rs.next()) {
           for (int i = 1; i <= iNumCols; i++) {
              colval = rs.getObject(i);
              System.out.print(colval + "  ");
           }
           System.out.println();
        }

        rs.close();
        stmt.close();
        con.close();
     }
     catch (Exception e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
     }
  }
}
```

--> store the file with .java extn
--> copy the Excel_JDBC30.jar to the same location where this program is stored.

--> compile the above file from dos as
javac -cp Excel_JDBC30.jar;. OraConn.java
-->Execute as
java -cp Excel_JDBC30.jar;. OraConn

Question 8: What do you mean by cold backup, hot backup?
Answer : This question is not directly related to JDBC but some time asked during JDBC interviews. Cold back is the backup techniques in which backup of files are taken before the database restarted. In hot backup backup of files and table is taken at the same time when database is running. A warm is a recovery technique where all the tables are locked and users cannot access at the time of backing up data.

Question 9: What are the locking system in JDBC
Answer : One more tough JDBC question to understand and prepare. There are 2 types of locking in JDBC by which we can handle multiple user issue using the record. if two user are reading the same record then there is no issue but what if users are updating the record , in this case changes done by first user is gone by second user if he also update the same record .so we need some type of locking so no lost update.

Optimistic Locking: optimistic locking lock the record only when update take place. Optimistic locking does not use exclusive locks when reading

Pessimistic locking: in this record are locked as it selects the row to update