JDBC

=====

As if now it is known as trademark.

But earlier it is known as Java Database Connectivity.

RAM is a temperory storage device or medium.

During the program execution our data will store in RAM.

Once the program exeuction is completed we will loss the data.

To overcome this limitation we are making our application writing the data into files or database softwares.

Files and database softwares act like a performance storage medium or device.

Persistence

==========

The process of storing and managing the data for a long period of time is called persistence.

Important terminologies

==========

1) Persistence store

It is a place where we can store and manage the data for a long period of time is called persistence store.

ex:

Files

Database softwares

2) Persistence data

Data of a persistence store is called persistence data.

ex:

records

tables

3) Persistence operations

Insert,Update,Delete,Select,Create and etc are called persistence operations.

In realtime this operations are also known as CURD operation, CRUD operation

or SCUD operation.

ex:

C - create S - select

U - update C - create

R - read U - update

D - delete D - delete

4) Pesistence logic

A logic which is capable to perform persistence operations is called persistence logic.

ex:

JDBC Code

Hibernate code

Ibatis code

IOStream

5) Persistence technology

A technology which is used to develop persistence logics is called persistence technology.

ex:

JDBC

Hibernate

EJB

and etc

Q)What is JDBC?

JDBC is a persistence technology which is used to develop persistence logics having

the capability to perform persistence operations on persistence data of a persistence

store.

Note:

IOStream

JavaApp ---------- Files

Serialization/Deserialization

JDBC Code

JavaApp ---------- Database S/W

Serialization

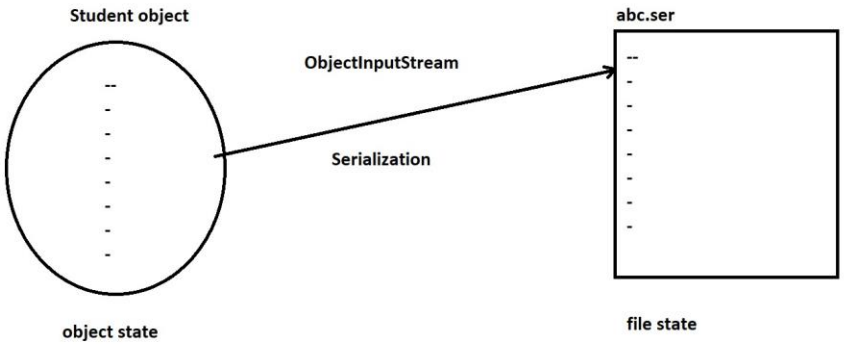
The process of take the data from object and storing in a file is called serialization.

In general , Converting from object state to file state is called Serialization.

Diagram: jdbc1.1

In serialization , object will not store in a file.Object data will store in a file.

Diagram: jdbc1.1

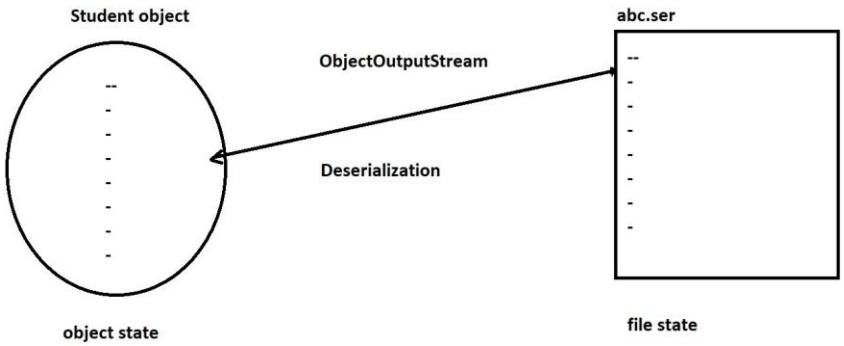


Deserialization

The process of taking the data from file and repreasting an object is called Deserialization.

In general, converting file state to object state.

Diagram: jdbc1.2



Limitations with files as persistence store

==========

> We can store limited amount of data.

> There is no security.

> Fetching the data with multiple conditions is not possible.

> It does not show an application with relationships.

> It does not allow us to apply constraints i.e primary key,foreign key,not null and etc.

> Updation and Deletion of data can't be done directly.

> Merging and comparision of data can't be done easily.

Advantages of Database as persistence store

==========

> We can store unlimited amount of data.

> There is a security.

> Fetching the data with multiple condition is possible.

> It shows an applicaiton with relationships.

> It allows us to apply constraints.

> Updation and Deletion of data can be done directly.

> Merging and comparision of data can be done easily.

Every JDBC application is a two-tier application.Where java with JDBC code acts like a frontend/Tier1/Layer1 and database software acts like a backend/Tier2/Layer2.

The one which is visible to the enduser to perform some operations is called frontend.

The one which is not visible to the enduser but it performs operations based on the instructions given by frontend is called backend.

JDBC Driver

==========

JDBC driver acts like a bridge between java application and database software.

It will convert java calls to database calls and vice versa.

Here calls means instructions.

Diagram: jdbc2.1

ODBC Driver

==========

VBScript, D2k, Perl and etc uses ODBC driver to interact with database software.

Diagram: jdbc2.2

ODBC driver developed in c language by taking the support of pointers.But java does not support pointers.To overcome this limitation Sun Micro System introduced JDBC drivers exclusively.

We will get JDBC softwares from following parties.

1) Sun Micro System (creator of jdbc driver)

2) Database Vendor

3) Third party Vendor

We will get ODBC softwares from following parties.

1) Xopen company (creator of odbc driver)

2) Database vendor

3) Third party vendor

Q)What is JDBC?

JDBC is a open technology given by Sun Micro System having set of rules and guidelines to

develop JDBC drivers to interact with multiple database softwares.

Q)What is ODBC?

ODBC is a open technology given by Xopen company having set of rules and guidelines to

develop ODBC drivers to interact with multiple database softwares.

To use any JDBC driver we need to register with DriverManager service.

Every JDBC application contains one built-in service called DriverManager service.

Class.forName()

==========

It is highly recommanded to used Class.forName() method to register JDBC driver with DriverManager service.

It is used to load the driver class but it won't create an object.

ex:

```
Class.forName("driver-class-name");
```

Connection object

==========

To perform any operation in a database we need to establish the connection with database.

Once the work with database is completed we need to close the Connection with database.

Connection is an interface which is present in java.sql package.

It is an object of underlying supplied java class which implements java.sql.Connection interface.

DriverManager.getConnection()

==========

DriverManager is a class which present in java.sql package.

A getConnection() static method is used to interact with database software and returns JDBC Connection object representing connectivity between java application and database software.

ex:

```
Connection con=DriverManager.getConnection("url");
```

Statement object

==========

Statement is an interface which is present in java.sql package.

It acts like a vehicle between java application and database software.

It is used to sends and executes SQL query in database software.

We can create Statement object as follow.

ex:

```
Statement st=con.createStatement();
```

ResultSet object

==========

Every ResultSet object contains two positions.

1) BFR (Before First Record/Row)

2) ALR (After Last Record/Row)

By default record pointer points to BFR position.

Every record ResultSet having 1 as base index and every column of record ResultSet having 1 as base index.

rs.next()

=========

It will move the record pointer to next position from current position.

If next position is a record then it will return true.

If next position is ALR then it will return false.

We can read the values of record ResultSet by using getXxx() method with index numbers or column names.

Here getXxx() methods means getInt(),getString(),getFloat(),getDouble() and etc.

Diagram: jdbc2.3

Types of Queries in JDBC

==========

According to JDBC point of view , we have two types of queries.

1)Select Query

2)Non-Select Query

1)Select Query

It will return bunch of records from database software.

ex:

```
select * from emp;
```

A JDBC Statement object gave executeQuery() method to execute select query.

ex:

```
ResultSet rs=st.executeQuery("select * from emp");
```

2) Non-Select Query

IT will return numeric value representing number of records effected in a database table.

ex:

```
delete from emp;
```

A JDBC Statement object gave executeUpdate() method to execute non-select query.

ex:

```
int result=st.executeUpdate("delete from emp");
```

Types of Drivers in JDBC

==========

We have following four types of JDBC drivers.

1) Type1 JDBC Driver / JDBC-ODBC Bridge Driver

2) Type2 JDBC Driver / Native API

3) Type3 JDBC Driver / Net Protocol

4) Type4 JDBC Driver / Native Protocol

Type4 JDBC Driver / Native Protocol

==========

Drive Class : oracle.jdbc.driver.OracleDriver

---------- ----------

pkg name classname

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

---------- | | |

sub protocol hostname portno logical db_name

Database username : system

Database password : admin

Steps to develop JDBC Application

==========

We have six steps to develop JDBC Application.

1) Register JDBC driver with DriverManager service.

2) Establish the connection with database software.

3) Create Statement object.

4) Sends and Executes SQL Query in database software.

5) Gather the result from database software to process the result.

6) Close all connection objects.

Eclipse

==========

IDE : JEE

Environment : Java

Flavour : Kepler,Indigo,Luna,Mars and etc.

File Format : zip file

Vendor : Eclipse Foundation

website : www.eclipse.org

Download link :

https://drive.google.com/file/d/1PZYYpd8RKpLWXe1TfHWaOQfNjOq1OIzy/view?usp=drive_link

student table

==========

```
drop table student;
create table student(sno number(3),sname varchar2(10),sadd varchar2(12));
insert into student values(101,'raja','hyd');
insert into student values(102,'ravi','delhi');
insert into student values(103,'ramana','vizag');
commit;
```

Steps to develop First JDBC application to select the records from student table using Eclipse IDE

==========

step1:

Launch Eclipse IDE by choosing workspace location.

step2:

Create a java project i.e IH-JAVA-025 inside eclipse IDE.

ex:

File --> new --> project --> java project --> Next

Name : IH-JAVA-025 --> Next --> Finish.

step3:

Add "ojdbc14.jar" file in project build path.

ex:

right click to IH-JAVA-025 project --> build path -->

configuration build path --> libraries --> Add external jars

--> select ojdbc14.jar file --> open --> ok.

step4:

Create a "com.ihub.www" package inside "src" folder.

ex:

right click to src folder --> new --> package -->

Name : com.ihub.www --> finish.

step5:

Create "SelectApp.java" file inside "src/com.ihub.www" package.

ex:

right click to com.ihub.www pkg --> new --> class -->

Name : SelectApp --> finish.

SelectApp.java

```
package com.ihub.www;

//ctrl+shift+o
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class SelectApp
{
	public static void main(String[] args)throws Exception
	{
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");
		Statement st=con.createStatement();
		ResultSet rs=st.executeQuery("select * from student");
		while(rs.next())
		{
```

```
                        System.out.println(rs.getInt(1)+" "+rs.getString(2)+"
"+rs.getString(3));
                }
                rs.close();
                st.close();
                con.close();

        }
}
```

step6:

Run the jdbc application.

ex:

Right click to SelectApp.java file --> run as --> java application.

Q)Write a jdbc application to select student record based on student number?

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Scanner;
```

```
public class SelectApp2 {


	public static void main(String[] args)throws Exception
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the student number :");
		int no=sc.nextInt();



		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");
		Statement st=con.createStatement();
		String qry="select * from student where sno="+no;
		ResultSet rs=st.executeQuery(qry);


		int cnt=0;
		while(rs.next())
		{
			System.out.println(rs.getInt(1)+" "+rs.getString(2)+"
"+rs.getString(3));
			cnt=1;
		}
```

```
            if(cnt==0)
                System.out.println("No rows selected");

            rs.close();
            st.close();
            con.close();

        }

}
```

Non-Select Queries

==========

Q)Write a JDBC application to insert a record into student table?

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
```

```
import java.util.Scanner;


public class InsertApp
{
	public static void main(String[] args)throws Exception
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the student no :");
		int no=sc.nextInt();
		System.out.println("Enter the student name :");
		String name=sc.next();
		System.out.println("Enter the student address :");
		String add=sc.next();

		//converting inputs according to SQL query
		name="'"+name+"'";
		add="'"+add+"'";

		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");
		Statement st=con.createStatement();

		String qry="insert into student values("+no+","+name+","+add+")";
```

```
            int result=st.executeUpdate(qry);

            if(result==0)
                System.out.println("No Record inserted");
            else
                System.out.println(result+" Record inserted");

            st.close();
            con.close();

        }
}
```

Q)Write a JDBC application to update student name based on student number?

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.util.Scanner;
```

```
public class UpdateApp
{
	public static void main(String[] args)throws Exception
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the student no :");
		int no=sc.nextInt();
		System.out.println("Enter the student name :");
		String name=sc.next();

		//converting inputs according to SQL query
		name="'"+name+"'";

		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");
		Statement st=con.createStatement();

		String qry="update student set sname="+name+" where sno="+no;

		int result=st.executeUpdate(qry);

		if(result==0)
```

```
                System.out.println("No record updated");
        else
                System.out.println(result+" record updated");

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

Q)Write a jdbc application to delete student record based on student number?

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.util.Scanner;

public class DeleteApp
{
    public static void main(String[] args)throws Exception
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the student no :");
```

```
            int no=sc.nextInt();


            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");
            Statement st=con.createStatement();


            String qry="delete from student where sno="+no;


            int result=st.executeUpdate(qry);
            if(result==0)
                  System.out.println("No record deleted");
            else
                  System.out.println(result+" record deleted");


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

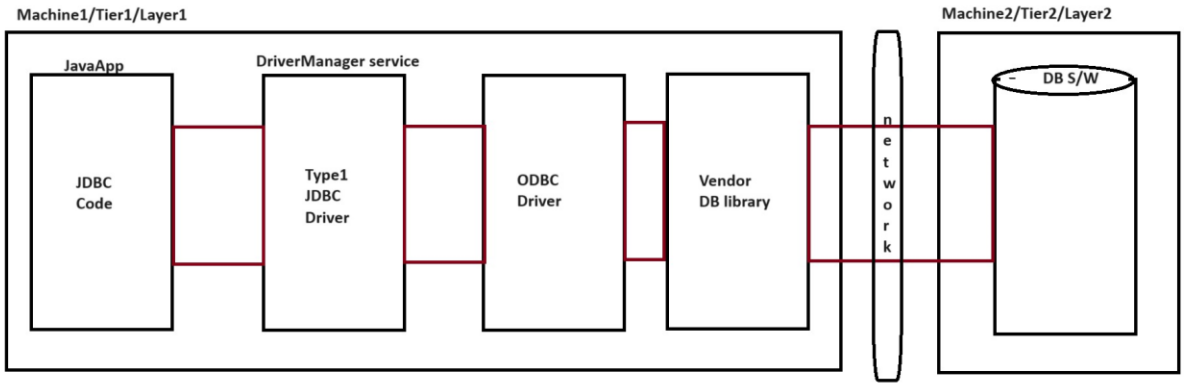
Type1 JDBC Driver Architecture / JDBC-ODBC Bridge Driver (Partly Java Driver)

==========

Type1 JDBC driver is not designed to interact with database software directly.

It is designed to take the support of ODBC driver and Vendor DB library to locate and interact with database software.

Diagram: jdbc3.1



Advantages:

> It is a built-in driver of JDK.

> Using type1 jdbc driver we can interact with any database software.

Disadvantages:

> This driver performance is low.It is not suitable for medium and large scale projects.Hence it is not a industry standard drive.

> To work with Type1 JDBC driver we need to arrange ODBC drivers and vendor db libraries.

> Since ODBC driver and Vendor db library present at client side. So it is not suitable for

untrusted applets to database communication.

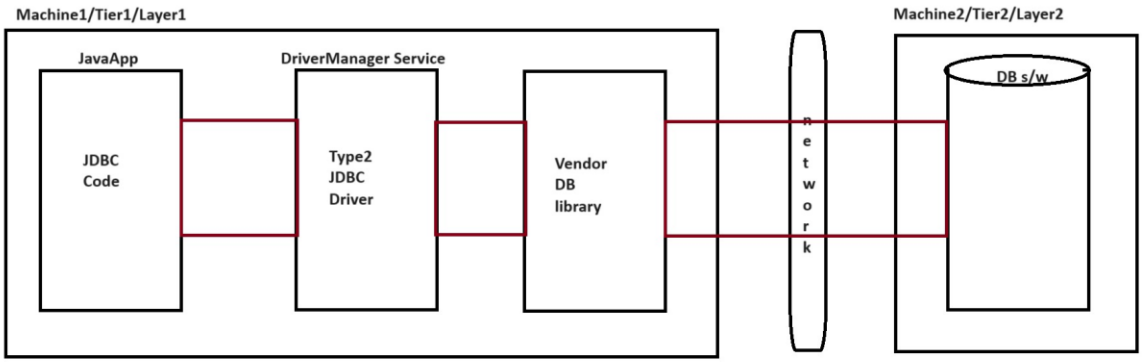
Type2 JDBC Driver Architecture / Native API (partly java driver)

==========

Type2 JDBC driver is not designed to interact with database software directly.

It is designed to take the support of vendor db library to locate and interact with database software.

Diagram: jdbc3.2



Advantages:

> This driver will give better performance when compare to Type1 jdbc driver.

> Type2 jdbc driver will not take the support ODBC driver.

Disadvantages:

> This driver performance is quit slow.It is not suitable for medium and large scale projects.Hence it is not a industry standard driver.

> To work with type2 jdbc driver we need to arrange Vendor db library seperately.

> Since vendor db library present at client side so it is not suitable to perform untrusted applets to database communication.

> For every database software we need to arrange type2 jdbc driver.

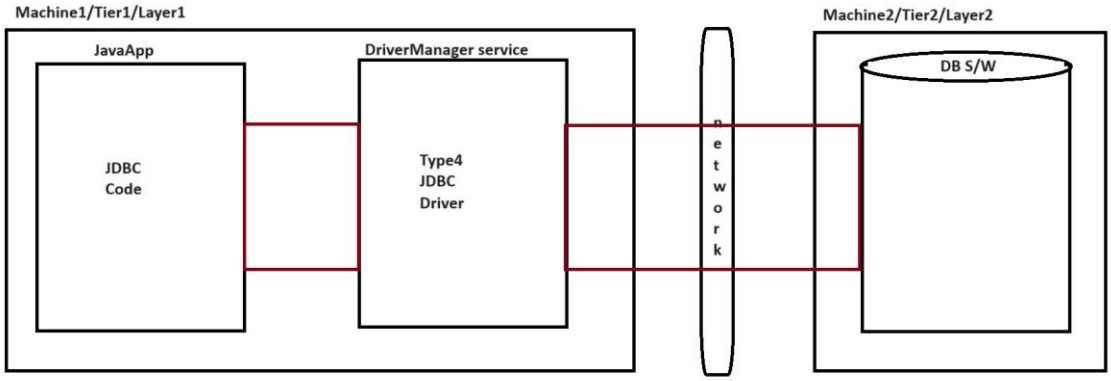
Type4 JDB Driver Archiecture / Native Protocol (Java Driver)

==========

Type4 JDBC driver is not designed to take the support of ODBC driver and Vendor DB library.

It is designed to interact with database software directly.

Diagram: jdbc4.1



Advantages:

> It will give better performance when compare to Type1 and Type2 JDBC driver.

> It is completely developed by using java so it will give platform independency.

> It is suitable for medium and large scale projects.Hence it is a industry standard driver.

> It does not support ODBC driver and Vendor DB library.

> Since ODBC driver and Vendor db library not present at client side so it is suitable

for untrusted applets to database communication.

Disadvantages:

> It is not a built-in driver of JDK.

> For every database we need to arrange type4 jdbc driver seperately.

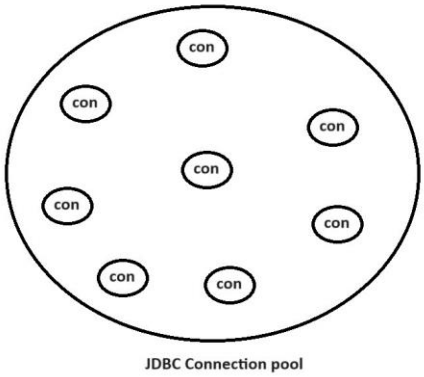
JDBC Connection pool

==========

It is a factory containing set of readily available JDBC Connection objects before actual being used.

JDBC Connection pool represent connectivity with same database software.

Diagram: jdbc4.2



JDBC Connection pool

Advantages:

> It gives resuable JDBC Connection object.

> With minimum number of JDBC Connections we can interact with multiple clients.

> User is not responsible to create,manage and destroy JDBC Connection objects.JDBC Connection pool is responsible to create,manage and destroy JDBC Connection objects.

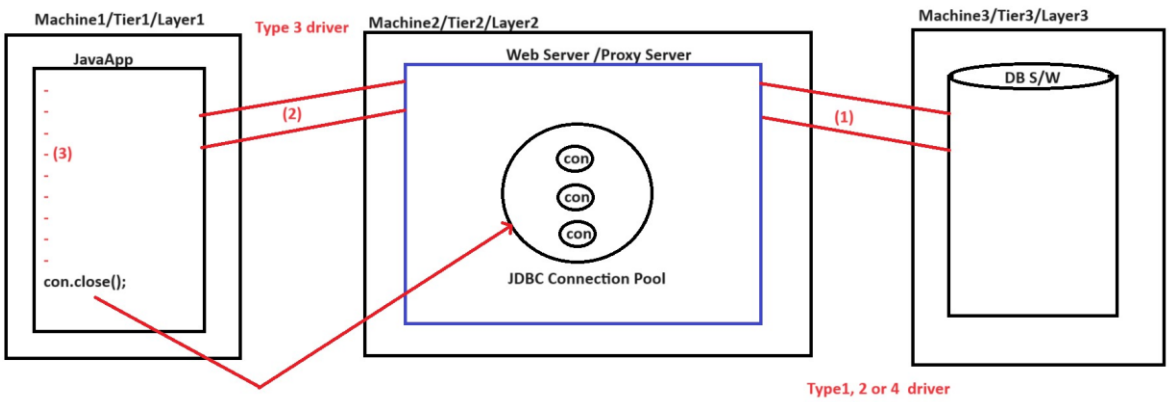
Type3 JDBC Driver Architecture/ Net Protocol

==========

Web Server , Proxy Server , IDE's server contains JDBC Connection pool.

Type3 JDBC driver is designed interact with webserver,proxy server or IDE's server to get resuable JDBC Connection objects from JDBC Connection pool.

Diagram: jdbc4.3



With respect to the diagram

---

1) Web server or proxy server interacts with database software to get reusable JDBC

Connection objects in JDBC Connection pool.

2) Java Application interact with Webserver or Proxyserver to get one reusable JDBC

Connection object from JDBC Connection pool.

3) Java application uses JDBC Connection object to create other JDBC Connnectio objects.

4) Once if we call con.close() then JDBC Connection object goes back to JDBC Connection pool.

Types of Connection objects in JDBC

==========

We have two types of JDBC Connection objects.

1)Direction JDBC Connection object

A JDBC Connection object which is created by the user.

ex:

```
Class.forName("driver-class_name");
Connection
con=DriverManager.getConnection("url","uname","pwd");
```

2)Pooled JDBC Connection object

A JDBC Connection object which is gathered from JDBC Connection pool.

Types of Statement objects in JDBC

==========

We have three Statement objects in JDBC.

1)Simple Statement object

It is an object of underlying supplied java class which implements java.sql.Statement interface.

2)PreparedStatement object

IT is an object of underlying supplied java class which implements java.sql.PreparedStatement interface.

3)CallableStatement object

It is an object of underlying supplied java class which implements java.sql.CallableStatement interface

SQL Injection problem

==========

Along with input values if we pass special SQL instructions which change the behaviour of a query and behaviour of an application is called SQL Injection Problem.

Here special SQL instructions means comments in SQL i.e --.

While dealing with simple Statement object there is a chance of raising SQL injection problem.

To overcome this limitation we need to use PreparedStatement object.

ex:

Enter the username : raja'--

Enter the password : hyd

Valid Credentials

userlist table

```
drop table userlist;
create table userlist(uname varchar2(10),pwd varchar2(10));
insert into userlist values('raja','rani');
insert into userlist values('king','kingdom');
commmit;
```

ex:

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
```

```
import java.sql.Statement;
import java.util.Scanner;

public class SQLInjProbApp {

	public static void main(String[] args)throws Exception
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the username :");
		String name=sc.next();
		System.out.println("Enter the password :");
		String pass=sc.next();

		//convert inputs according to SQL Query
		name="'"+name+"'";
		pass="'"+pass+"'";

		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");
		Statement st=con.createStatement();

		String qry="select count(*) from userlist where uname="+name+"
and pwd="+pass;
```

```
            ResultSet rs=st.executeQuery(qry);

            int cnt=0;
            while(rs.next())
            {
                  cnt=rs.getInt(1);
            }
            if(cnt==0)
                  System.out.println("Invalid Credentials");
            else
                  System.out.println("Valid Crendentials");

            rs.close();
            st.close();
            con.close();
      }

}
```

Limitations with Simple Statement object

==========

> It is not suitable to execute same query for multiple times with same values or different

values.

> It raises SQL Injection problem.

> Framing query with variables is quit complex.

> We can't use string values directly to query parameter without any conversion.

> It does not allow us to insert date values to database table column.

> It does not allow us to insert LOB values to database table column.

To overcome above limitations we need to use PreparedStatement object.

Pre-compiled SQL Query

==========

Our query goes to database software without inputs and becomes parsed query either it is executed or not is called pre-compiled SQL Query.

PreparedStatement object deals with pre-compiled SQL query.

Working with PreparedStatement object

==========

step1:

Create a query with placeholders or parameters.

ex:

```
String qry="insert into student values(?,?,?)";
```

step2:

Convert SQL query to precompiled SQL query.

ex:

```
PreparedStatement ps=con.prepareStatement(qry);
```

step3:

Set the values to query parameters.

ex:

```
ps.setInt(1,no);
ps.setString(2,name);
ps.setString(3,add);
```

step4:

Execute the pre-compiled SQL query.

ex:

```
ps.executeUpdate();
```

step5:

Close PreparedStatement object.

ex:

```
ps.close();
```

Q)Write a jdbc application to insert a record into student table using PreparedStatement object?

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.util.Scanner;

public class PSInsertApp
{
	public static void main(String[] args)throws Exception
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the student no :");
		int no=sc.nextInt();
		System.out.println("Enter the student name :");
		String name=sc.next();
```

```
System.out.println("Enter the student address :");
String add=sc.next();

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");

String qry="insert into student values(?,?,?)";

PreparedStatement ps=con.prepareStatement(qry);

//set the values
ps.setInt(1,no);
ps.setString(2,name);
ps.setString(3,add);

//execute
int result=ps.executeUpdate();
if(result==0)
        System.out.println("No Record inserted");
else
        System.out.println("Record inserted");

ps.close();
```

```
            con.close();

        }
}
```

Q)Write a jdbc application to update student name based on student number?

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.util.Scanner;

public class PSUpdateApp {

        public static void main(String[] args)throws Exception
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the student no :");
                int no=sc.nextInt();

                System.out.println("Enter the student name :");
                String name=sc.next();
```

```
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");

            String qry="update student set sname=? where sno=?";

            PreparedStatement ps=con.prepareStatement(qry);

            //set the values to query parameters
            ps.setString(1,name);
            ps.setInt(2,no);

            //execute
            int result=ps.executeUpdate();
            if(result==0)
                  System.out.println("No Record updated");
            else
                  System.out.println("Record updated");

            ps.close();
            con.close();

      }
```

```
}
```

Q)Write a jdbc application to delete student record based on student number?

ex:

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.util.Scanner;

public class PSDeleteApp
{
	public static void main(String[] args)throws Exception
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the student no :");
		int no=sc.nextInt();

		Class.forName("oracle.jdbc.driver.OracleDriver");
```

```
            Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");

            String qry="delete from student where sno=?";

            PreparedStatement ps=con.prepareStatement(qry);

            //set the values
            ps.setInt(1, no);

            //execute
            int result=ps.executeUpdate();

            if(result==0)
                  System.out.println("No Record deleted");
            else
                  System.out.println("Record deleted");

            ps.close();
            con.close();


      }
}
```

Solution for SQL injection problem

==========

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Scanner;

public class SolForSQLInjProbApp
{
	public static void main(String[] args)throws Exception
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the username :");
		String name=sc.next();
		System.out.println("Enter the password :");
		String pass=sc.next();

		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");
```

```
String qry="select count(*) from userlist where uname=? and
pwd=?";

PreparedStatement ps=con.prepareStatement(qry);

//set the values
ps.setString(1,name);
ps.setString(2,pass);

//execute
ResultSet rs=ps.executeQuery();

int cnt=0;
while(rs.next())
{
    cnt=rs.getInt(1);
}

if(cnt==0)
    System.out.println("Invalid Credentials");
else
    System.out.println("Valid Credentials");
rs.close();
ps.close();
```

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

DatabaseMetaDeta

==========

DatabaseMetaData is an interface which is present in java.sql package.

DatabaseMetaData provides metadata of a database.

DatabaseMetaData gives information about database product name, database product version, database driver name, database driver version, database username and etc.

We can create DatabaseMetaData object as follow.

ex:

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

ex:

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
```

```
public class DBMDApp
{
	public static void main(String[] args)throws Exception
	{
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");
		DatabaseMetaData dbmd=con.getMetaData();

		System.out.println(dbmd.getDatabaseProductName());
		System.out.println(dbmd.getDatabaseProductVersion());

		System.out.println(dbmd.getDriverName());
		System.out.println(dbmd.getDriverVersion());

		System.out.println(dbmd.getUserName());



		con.close();
	}
}
```

ResultSetMetaData

==========

ResultSetMetaData is an interface which is present in java.sql package.

ResultSetMetaData provides metadata of a table.

ResultSetMetaData gives information about number of columns, type of columns , size of columns and etc.

We can create ResultSetMetaData object as follow.

ex:

```
ResultSetMetaData rsmd=rs.getMetaData();
```

ex:

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;

public class RSMDApp
{
	public static void main(String[] args)throws Exception
```

```
	{
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");
		Statement st=con.createStatement();
		String qry="select * from student";
		ResultSet rs=st.executeQuery(qry);
		ResultSetMetaData rsmd=rs.getMetaData();

		System.out.println(rsmd.getColumnCount());
		System.out.println(rsmd.getColumnName(2));
		System.out.println(rsmd.getColumnTypeName(2));
		System.out.println(rsmd.getColumnDisplaySize(2));

		rs.close();
		st.close();
		con.close();

	}
}
```

Standard procedure to write JDBC application

==========

```
package com.ihub.www;
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class StandardApp
{
	public static void main(String[] args)
	{
		final String DRIVER="oracle.jdbc.driver.OracleDriver";
		final String URL="jdbc:oracle:thin:@localhost:1521:XE";
		final String USERNAME="system";
		final String PASSWORD="admin";
		final String QUERY="select * from student";

		Connection con=null;
		Statement st=null;
		ResultSet rs=null;
		try
		{
			Class.forName(DRIVER);

	con=DriverManager.getConnection(URL,USERNAME,PASSWORD);
			st=con.createStatement();
```

```
                    rs=st.executeQuery(QUERY);
                    while(rs.next())
                    {
                        System.out.println(rs.getRow()+" "+rs.getInt(1)+"
"+rs.getString(2)+" "+rs.getString(3));
                    }
                    rs.close();
                    st.close();
                    con.close();

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

JDBC Flexible Application

==========

In jdbc , Connection object consider as heavy weight object.

It is not recommanded to creat Connection object in every java method.

We need to create a seperate class which returns JDBC Connection object.

ex:

DBConnection.java

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;

public class DBConnection
{
	static Connection con=null;

	public static Connection getConnection()
	{
		try
		{
			Class.forName("oracle.jdbc.driver.OracleDriver");

	con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","
system","admin");
		}
		catch(Exception e)
```

```
		{
			e.printStackTrace();
		}

		return con;
	}
}
```

FlexibleApp.java

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;

public class FlexibleApp
{
	public static void main(String[] args)throws Exception
	{
		Connection con=DBConnection.getConnection();
		Statement st=con.createStatement();
```

```
            String qry="select * from student order by sno desc";
            ResultSet rs=st.executeQuery(qry);
            while(rs.next())
            {
                  System.out.println(rs.getRow()+" "+rs.getInt(1)+"
"+rs.getString(2)+" "+rs.getString(3));
            }
            rs.close();
            st.close();
            con.close();

      }
}
```

Working with Date values

==========

While dealing with DOA,DOB,DOD,DOR and etc we need to insert and retrieve data values.

It is never recommanded to store date values in the form of String because it won't give proper comparision between two dates.

Every database software support different date patterns.

ex:

Oracle --> dd-MMM-yy

MySQL --> yyyy-MM-dd

and etc.

While working simple Statement object we can't place date values to query parameters.

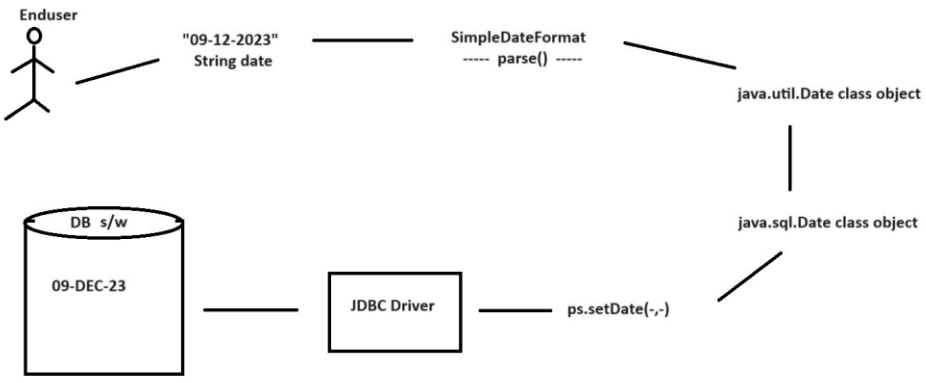
To overcome this limitation we need to use PreparedStatement object.

A java.util.Date class is not suitable to perform database operation.

A java.sql.Date class is suitable to perform database operation.

Standard procedure to insert Date values

Diagram: jdbc6.1



With the respect to the diagram:

1) Enduser will give date value in the form String.

2) A parse() method of SimpleDateFormat class converts String date to java.util.Date class object.

3) Our application converts java.util.Date class object to java.sql.Date class object.

4) A ps.setDate(-,-) method is used to set the date value to query parameter.

5) Once JDBC driver will get date value then it will insert in the pattern which is supported

by underlying databse software.

emp1 table

=========

```
drop table emp1;
create table emp1(eid number(3),ename varchar2(10),edoj date);
```

DateInsertApp.java

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.text.SimpleDateFormat;
import java.util.Scanner;

public class DateInsertApp
{
	public static void main(String[] args)throws Exception
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the employee id :");
		int id=sc.nextInt();
		System.out.println("Enter the employee name :");
		String name=sc.next();
		System.out.println("Enter the DOJ (dd-MM-yyyy) :");
		String doj=sc.next();

		//converting string date to util date
		SimpleDateFormat sdf=new SimpleDateFormat("dd-MM-yyyy");
		java.util.Date udoj=sdf.parse(doj);
```

```
            //converting util date to sql date

            long ms=udoj.getTime();
            java.sql.Date sqldoj=new java.sql.Date(ms);

            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");
            String qry="insert into emp1 values(?,?,?)";
            PreparedStatement ps=con.prepareStatement(qry);

            //set the values
            ps.setInt(1,id);
            ps.setString(2,name);
            ps.setDate(3,sqldoj);

            int result=ps.executeUpdate();
            if(result==0)
                  System.out.println("No Record inserted");
            else
                  System.out.println("Record inserted");

            ps.close();
            con.close();
```

```
	}
}
```

DateRetrieveApp.java

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.text.SimpleDateFormat;

public class DateRetrieveApp
{
	public static void main(String[] args)throws Exception
	{
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");
		Statement st=con.createStatement();
		String qry="select * from emp1";
		ResultSet rs=st.executeQuery(qry);
```

```
            while(rs.next())
            {
                  int id=rs.getInt(1);
                  String name=rs.getString(2);
                  java.sql.Date sqldoj=rs.getDate(3);

                  //convert sql date to util date
                  java.util.Date udoj=(java.util.Date)sqldoj;

                  SimpleDateFormat sdf=new SimpleDateFormat("dd-MM-
yyyy");
                  String doj=sdf.format(udoj);

                  System.out.println(id+" "+name+" "+doj);
            }
            rs.close();
            st.close();
            con.close();
      }
}
```

Working with LOB values

==========

Files are known as LOB's.

We have two types of LOB's.

1) BLOB (Binary Large Object)

ex:

images,audios,videos,avi files and etc.

2) CLOB (Character Large Object)

ex:

text file,doc file, advanced text file and etc.

While dealing with matrimonial applications, job portal applications , profile management applications, we need to insert and retrive LOB values.

Using simple Statement object we can't place LOB values to query parameter.

To overcome this limitation we need to use PreparedStatement object.

We can set the LOB values to query parameter as follow.

ex:

ps.setBinaryStream(-,-,-)/ps.setBLOB(-,-,-)

ps.setCharacterStream(-,-,-)/ps.setCLOB(-,-,-)

emp2 table

==========

```
drop table emp2;
```

```
create table emp2(eid number(3),ename varchar2(10),ephoto BLOB);
```

PhotoInsertApp.java

```
package com.ihub.www;

import java.io.File;
import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.util.Scanner;

public class PhotoInsertApp
{
	public static void main(String[] args)throws Exception
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the employee Id :");
		int id=sc.nextInt();

		System.out.println("Enter the employee name :");
```

```
String name=sc.next();

//locate a photo
File f=new File("src/com/ihub/www/rock.jpg");
FileInputStream fis=new FileInputStream(f);

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");

String qry="insert into emp2 values(?,?,?)";

PreparedStatement ps=con.prepareStatement(qry);

//set the values
ps.setInt(1,id);
ps.setString(2,name);
ps.setBinaryStream(3,fis,(int)f.length());

//execute
int result=ps.executeUpdate();
if(result==0)
        System.out.println("No Record inserted");
else
```

```
                System.out.println("Record inserted");

            ps.close();
            con.close();

        }
}
```

PhotoRetrieveApp.java

```
package com.ihub.www;

import java.io.FileOutputStream;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

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

```
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");
		Statement st=con.createStatement();
		String qry="select * from emp2";
		ResultSet rs=st.executeQuery(qry);
		while(rs.next())
		{
			InputStream is=rs.getBinaryStream(3);

			FileOutputStream fos=new FileOutputStream("E:\\IHUB-
TRAINING-BATCHES\\IH-JAVA-025\\ramakrishna.png");

			int byteReads=0;
			byte[] buff=new byte[255];

			while((byteReads=is.read(buff))!= -1)
			{
				fos.write(buff,0,byteReads);
			}

			fos.close();
		}
		System.out.println("Please check the location");
		rs.close();
```

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

Assignment

Q)Write a jdbc application to create a student table ?

Working with properties file

==========

In regular intervals, our DBA will change username and password for security reason.

It is never recommanded to pass database properties directly to the application.

It is always recommanded to read database properties from properties file.

Properties file contains key and value pair.

dbdetails.properties

```
driver=oracle.jdbc.driver.OracleDriver
url=jdbc:oracle:thin:@localhost:1521:XE
username=system
```

```
password=admin
```

PropertiesFileApp.java

```
package com.ihub.www;

import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Properties;

public class PropertiesFileApp
{
	public static void main(String[] args)throws Exception
	{
		//locate a properties file
		FileInputStream fis=new
FileInputStream("src/com/ihub/www/dbdetails.properties");

		//create Properties class object
		Properties p=new Properties();
```

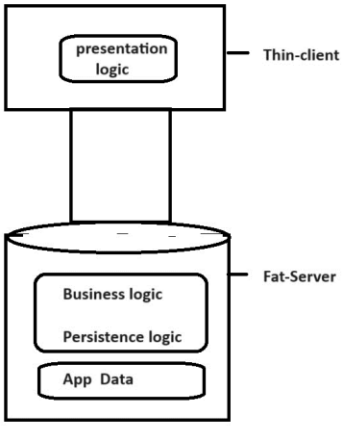
```
		//load the data from file to class
		p.load(fis);

		//read the data from Properties class
		String s1=p.getProperty("driver");
		String s2=p.getProperty("url");
		String s3=p.getProperty("username");
		String s4=p.getProperty("password");

		//reading the data from student table
		Class.forName(s1);
		Connection con=DriverManager.getConnection(s2,s3,s4);
		Statement st=con.createStatement();
		String qry="select * from student";
		ResultSet rs=st.executeQuery(qry);
		while(rs.next())
		{
			System.out.println(rs.getInt(1)+" "+rs.getString(2)+"
"+rs.getString(3));
		}
		rs.close();
		st.close();
		con.close();

	}
```

}

Thin-Client/Fat-Server application

==========

Diagram: jdbc7.1



Every JDBC application is a thin-client/fat-server application.

To develop thin-client/fat-server application we need to place business logic and persistence logic in database software in the form of PL/SQL procedures and functions.

To deal with PL/SQL procedures and functions we need to use CallableStatement object.

PL/SQL procedure

==========

```
create or replace procedure first_proc(A IN number,B IN number,C OUT number)
is
begin
C:=A+B;
END;
/
```

ex:

```
package com.ihub.www;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Types;

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

```
            Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");

            //create callable stmt object
            CallableStatement cst=con.prepareCall("{CALL first_proc(?,?,?)}");

            //register out parameter
            cst.registerOutParameter(3,Types.INTEGER);

            //set the values to IN parameter
            cst.setInt(1, 100);
            cst.setInt(2, 20);

            //execute the procedure
            cst.execute();

            //gather the result
            int result=cst.getInt(3);
            System.out.println("Sum of two numbers is ="+result);

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

Q)Write a jdbc application to create a student table ?

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class CreateTableApp
{
	public static void main(String[] args)throws Exception
	{
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");

		String qry="create table student(sno number(3),sname
varchar2(10),sadd varchar2(12))";

		PreparedStatement ps=con.prepareStatement(qry);

		//execute the qry
		ps.executeUpdate();
```

```
            System.out.println("Table Created");

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

Types of ResultSet objects

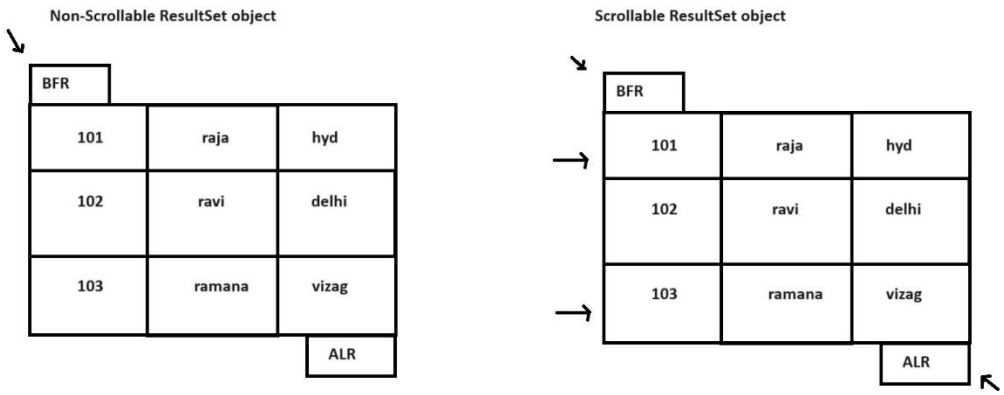
==========

We have two types of ResultSet objects.

1) Non-Scrollable ResultSet object

2) Scrollable ResultSet object

Diagram: jdbc8.1



1) Non-Scrollable ResultSet object

A ResultSet object which allows us to read the records sequentially, unidirectionally is called non-scrollable ResultSet object.

By default every ResultSet object is a non-scrollable ResultSet object.

If JDBC Statement object is created without type,mode value then that ResultSet object is non-scrollable ResultSet object.

ex:

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

2) Scrollable ResultSet object

A ResultSet object which allows us to read the records non-sequentially, bidirectionally or randomly is called scrollable ResultSet object.

If JDBC Statement object is created with type,mode value then that ResultSet object is scrollable ResultSet object.

ex:

```
Statement st=con.createStatement(type,mode);
ResultSet rs=st.executeQuery("select * from student");
```

We have two type values.

ex:

```
ResultSet.TYPE_SCROLL_SENSITIVE
ResultSet.TYPE_SCROLL_INSENSITIVE
```

We have two mode values.

ex:

```
ResultSet.CONCUR_READ_ONLY
ResultSet.CONCUR_UPDATABLE
```

Various methods present in Scrollable ResultSet object

==========

1) getRow()

It will return position of record pointer.

2) next()

It will move record pointer to next position from current position.

3) getXxx()

It is used to read the values from record ResultSet.

4) close()

It is used to close ResultSet object.

5) previous()

It will move record pointer to previous position.

6) first()

It will set record pointer to first record.

7) isFirst()

It is used to check record pointer is in first record or not.

8) last()

It will set record pointer to last record.

9) isLast()

It is used to check record pointer is in last record or not.

10) beforeFirst()

It is used to set the record pointer to BFR position.

11) afterLast()

It is used to set the record pointer to ALR position.

12) relative(+/-)

It will move record pointer to next position based on current position.

13) abosluate(+/-)

It will move record pointer to next position based on BFR and ALR position.

ex:

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class ScrollableStmtApp
{
	public static void main(String[] args)throws Exception
	{
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");
		Statement
st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_R
EAD_ONLY);
		String qry="select * from student";
		ResultSet rs=st.executeQuery(qry);
```

```
		//top to bottom
		while(rs.next())
		{
			System.out.println(rs.getRow()+" "+rs.getInt(1)+"
"+rs.getString(2)+" "+rs.getString(3));
		}

		rs.afterLast();
		while(rs.previous())
		{
			System.out.println(rs.getRow()+" "+rs.getInt(1)+"
"+rs.getString(2)+" "+rs.getString(3));
		}

		rs.first();
		System.out.println(rs.isFirst());
		System.out.println(rs.getRow()+" "+rs.getInt(1)+" "+rs.getString(2)+"
"+rs.getString(3));

		rs.last();
		System.out.println(rs.isLast());
		System.out.println(rs.getRow()+" "+rs.getInt(1)+" "+rs.getString(2)+"
"+rs.getString(3));

		//rs.relative(-2);
```

```
            //System.out.println(rs.getRow()+" "+rs.getInt(1)+"
"+rs.getString(2)+" "+rs.getString(3));


            rs.absolute(-2);

            System.out.println(rs.getRow()+" "+rs.getInt(1)+" "+rs.getString(2)+"
"+rs.getString(3));


            st.close();

            con.close();

            rs.close();

        }

}
```

Batch Processing

==========

Batch processing is used to declare multiple queries in a batch and makes a single call to the database.

In batch process, we need to add all the queries to batch by using addBatch() method.

ex:

```
st.addBatch(qry);
```

We can execute the batch by using executeBatch() method.

ex:

```
int[] arr=st.executeBatch();
```

ex:

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class BatchProcessingApp {

	public static void main(String[] args)throws Exception
	{
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");
		Statement st=con.createStatement();

		String qry1="insert into student values(104,'ramulu','pune')";
		String qry2="delete from student where sno=103";
		String qry3="update student set sname='rani' where sno=101";

		//add the queries to batch
		st.addBatch(qry1);
```

```
            st.addBatch(qry2);
            st.addBatch(qry3);

            //execute the batch
            int[] arr=st.executeBatch();

            //for each loop
            int sum=0;
            for(int i:arr)
            {
                  sum+=i;
            }
            System.out.println("No of records effected are ="+sum);

            st.close();
            con.close();
      }

}
```

Transaction Management

==========

A transaction is a set of one or more statements that is executed as a unit.

If transaction done successfully then will commit.

If transaction failed then it will rollback.

ex:

SBI table

==========

drop table sbi;

create table sbi(accno number(6), accholder varchar2(10), accbal number(10));

insert into sbi values(100001,'vamshi',5000);

insert into sbi values(200002,'sham',6000);

commit;

Kotak table

===========

drop table kotak;

create table kotak(accno number(6),accholder varchar2(10),accbal number(10));

```
insert into kotak values(111111,'venkat',80000);
insert into kotak values(222222,'pulley',90000);
commit;

ex:
---
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
import java.util.Scanner;

public class TXNManagementApp
{
	public static void main(String[] args)throws Exception
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the source Account No :");
		int sno=sc.nextInt();

		System.out.println("Enter the destination Account No :");
		int dno=sc.nextInt();
```

```
System.out.println("Enter the amount to transfer :");
int amt=sc.nextInt();

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","syste
m","admin");

//set auto commit false
con.setAutoCommit(false);

Statement st=con.createStatement();

//create queries
String qry1="update kotak set accbal=accbal-"+amt+" where
accno="+sno;
String qry2="update sbi set accbal=accbal+"+amt+" where
accno="+dno;

//add the queries to batch
st.addBatch(qry1);
st.addBatch(qry2);

//execute the batch
int[] arr=st.executeBatch();
```

```
//for each loop
boolean flag=true;
for(int i:arr)
{
	if(i==0)
	{
		flag=false;
		break;
	}
}

if(flag==true)
{
	System.out.println("Transaction Done Successfully");
	con.commit();
}
else
{
	System.out.println("Transaction Failed!!");
	con.rollback();
}

st.close();
```

```
            con.close();

        }
}
```

Q) What is the difference between RDBMS vs MongoDB ?

RDBMS	MongoDB
It is a relational database.	It is a non-relational database or document oriented database.
It will not store the data in key and value pair.	It will store the data in key and value pair.
It is not suitable for hierarchical data.	It is suitable for hierarchical data.
Tables	Collections
Rows	Documents
Columns	Fields

Steps to develop java application to interact with MongoDB

==========

step1:

Download and Install MongoDB Atlas/Compass.

ex:

https://drive.google.com/file/d/1hqzPIqjl-TUfSzIq8xpiJHaYVuu2Ut52/view?usp=sharing

step2:

Download and extract Mongo shell.

ex:

https://www.mongodb.com/try/download/shell

step3:

Copy and Paste Mongoshell inside MongoDB folder.

ex:

C:\Program Files\MongoDB

step4:

Copy "bin" directory of mongosh and paste in environment variables.

ex:

Right click to my pc --> properties --> advanced system settings -->

environmental variables --> system variables --> click on path --> edit button

--> new --> C:\Program Files\MongoDB\mongosh-2.1.0-win32-x64\bin

(add the bin directory) --> ok --> ok -->ok.

step5:

Open the command and type below command to use mongo client.

ex:

cmd> mongosh

step6:

Check list of commands of mongodb.

ex:

```
show dbs;

use mydb;

db.createCollection("student");

db.student.insertOne({
    "id":1,
```

```
        "name":"Alan Morries",
        "add":"Texas"
    })

    db.student.findOne();

    db.student.findOne({name:"Alan Morries"});

    db.dropDatabase();
```

step7:

Launch eclipse IDE by choosing workspace location.

step8:

Create a java project i.e MongoDBProj.

ex:

File --> new --> project --> java project --> Next -->

Project Name : MongoDBProj --> Next --> Finish.

step9:

Download and Extract mongodb jar files.

ex:

https://jar-download.com/artifact-search/mongodb-driver-sync

step10:

Add the jar files to project build path.

ex:

Right click to MongDBProj --> build path --> configuration build path --> libraries --> add external jars --> add 4 jar files of mongodb --> open --> ok.

step11:

Create a TestApp.java file inside "src/com.ihub.www" package

```
package com.ihub.www;

import org.bson.Document;

import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
```

```
public class TestApp
{
        public static void main(String[] args)
        {

                // Connect to MongoDB
      try (MongoClient mongoClient =
MongoClients.create("mongodb://localhost:27017"))
      {
         // Access a database
         MongoDatabase database = mongoClient.getDatabase("mydb");

         // Access a collection
         MongoCollection<Document> collection =
database.getCollection("mycollection");

         // Insert a document
         Document document = new Document("no", 1)
               .append("name", "Alan")
               .append("add", "Texas");
         collection.insertOne(document);

         // Query the collection
         Document query = new Document("no", 1);
         Document result = collection.find(query).first();
```

```
            System.out.println("Result: " + result);
        }
    }

}
```

step12:

Run the project.

Steps to interact with mysql database

==========

step1:

Download and install mysql database.

ex:

https://drive.google.com/file/d/1ZXgkKHVaBocltXcTQKfWhbLQGpzEZJ-C/view?usp=drive_link

Note:

username : root (default)

password : root

step2:

Connect with mysql database.

ex:

password : root

step3:

Check list of databases present in mysql.

ex:

show databases;

step4:

Create a schema in mysql database.

ex:

create schema ih_java_025;

use ih_java_025;

step5:

Create a student table with records.

ex:

```
create table student(sno int(3),sname varchar(10), sadd varchar(12));
insert into student values(101,'raja','hyd');
insert into student values(102,'ravi','delhi');
insert into student values(103,'ramana','vizag');
commit;
select * from student;
```

step6:

Launch eclipse IDE by choosing workspace location.

step7:

Create a java project i.e MySQLProj.

ex:

File --> new --> project --> java project --> new -->

Project Name : MySQLProj --> next --> finish.

step8:

Download and extract mysql jar file.

ex:

http://www.java2s.com/Code/Jar/m/Downloadmysqlconnectorjavajar.htm

step9:

Add the jar file to project build path.

ex:

Right click to MySQLProj --> build path --> configuration build path --> libraries --> add external jars --> select mysql-connector-java.jar file

--> open -->ok.

step10:

Create a TestApp.java file inside "src/com.ihub.www" package.

```
package com.ihub.www;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class TestApp
```

```
{
	public static void main(String[] args)
	{
		final String DRIVER="com.mysql.jdbc.Driver";
		final String
URL="jdbc:mysql://localhost:3306/IH_JAVA_025?characterEncoding=utf8";
		final String USERNAME="root";
		final String PASSWORD="root";
		final String QUERY="select * from student";

				Connection con=null;
				Statement st=null;
				ResultSet rs=null;
				try
				{
					Class.forName(DRIVER);

	con=DriverManager.getConnection(URL,USERNAME,PASSWORD);
					st=con.createStatement();
					rs=st.executeQuery(QUERY);
					while(rs.next())
					{
						System.out.println(rs.getRow()+"
"+rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3));
					}
```

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

step11:

Run jdbc application.