JDBC

==========

As if now it is known as trademark.

But earlier it is called Java Database Connectivity.

RAM is a temperory storage device or medium.

During the program execution our data will store inside RAM.

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

To overcome this limitation , we are making our applications writing the data in a file or database software.

Files and database softwares act like a permanent storage or medium.

Persistence

==========

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

Persistence Terminologies

==========

1) Persistence store

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

ex:

Files

Database softwares.

2) Persistence data

A data of a persistence store is called persistence data.

ex:

Records

Tables

3) Persistence operation

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

In the realtime this operation is 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) Persistence logic

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

ex:

IOStream
JDBC Code
Hibernate Code

5) Persistence technology

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

ex:

JDBC
Hibernate
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

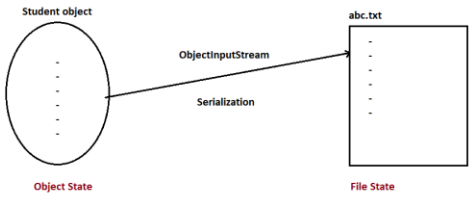
JDBC Code
JavaApp ---------- Database s/w

Serialization

==========

The process of storing object data into a file is called serialization.

The process of converting object state to file state is called serialization.

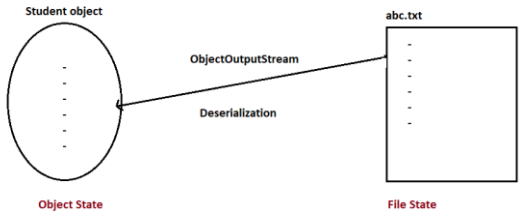


Deserialization

==========

The process of taking the data from file and representing an object is called deserialization.

The process of converting file state to object state is called deserialization.



Limitations with files as a persistence store

==========

> It stores 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 contraints.

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

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

Advantages of database as a persistence store

==========

> It stores unlimited amount of data.

> There is a security.

> Fetching the data with multiple condition is possible.

> It shows an application with relationships.

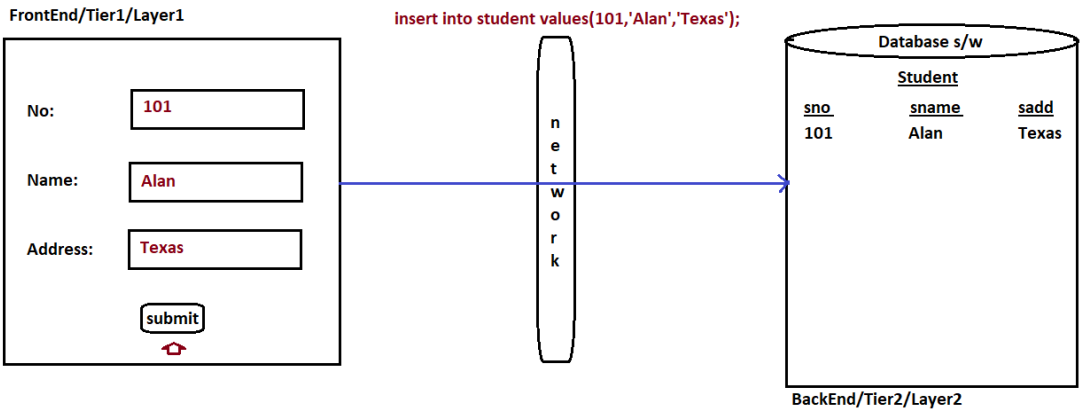
> It allows us to apply constraints.

> 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 act like a frontend/tier1/Layer1 and database software acts like a backend/tier2/layer2.

Enduser is a non-technical person. He can't prepare and execute SQL query in a database software.He depends upon frontend developers having the capability to do that work for him.

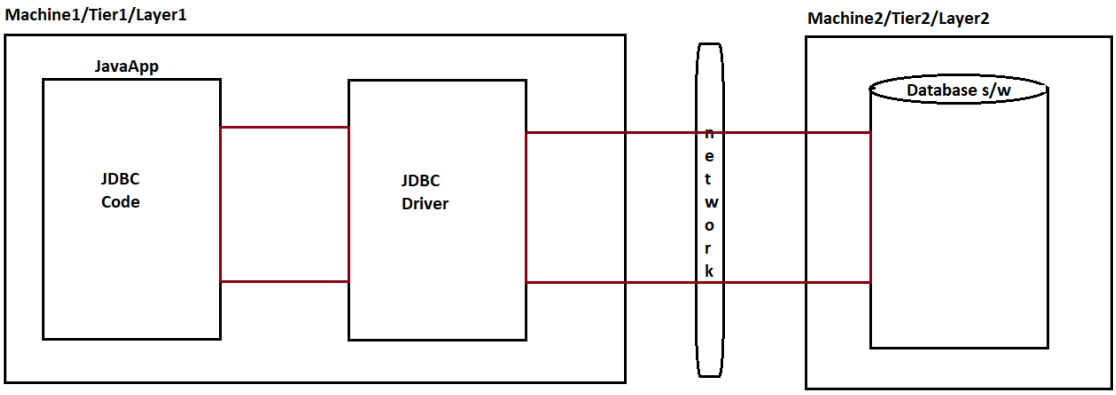


JDBC Driver

==========

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

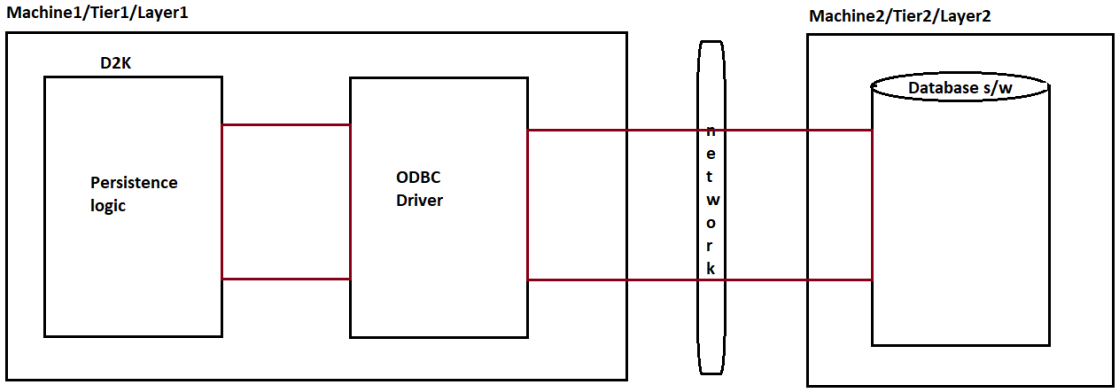
It is used to convert java instructions to database instructions and vice versa.



ODBC Driver

==========

VBScript, D2K, Perl and etc uses ODBC driver to interact with database software.



ODBC drivers developed in C language by taking the support pointers. Java does not support pointers. To overcome this limitation Sun Micro System introduced JDBC Driver exclusively.

Q) Types of drivers in JDBC?

We have four types of JDBC drivers.

1) Type1 JDBC Driver / JDBC-ODBC Bridge Driver

2) Type2 JDBC Driver / Native API Driver

3) Type3 JDBC Driver / Net Protocol Driver

4) Type4 JDBC Driver / Native Protocols Driver

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 always recommanded to register JDBC driver with DriverManager service.

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

ex:

```
Class.forName("Driver-Class-Name");
```

Connection object

==========

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.

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

Once work with database is completed we have to close the connection.

ex:

```
Connection con;
```

DriverManager.getConnection()

==========

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

A getConnection() method is used to interact with database software and returns one JDBC Connection object represent 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 an object of underlying supplied java class which implements java.sql.Statement interface.

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

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

ex:

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

ResultSet object

==========

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

It contains two positions.

1) BFR (Before First Record/Row)

2) ALR (After Last Record/Row)

By default record pointer points to BFR position only.

rs.next()

==========

It is used to move to next position from current position.

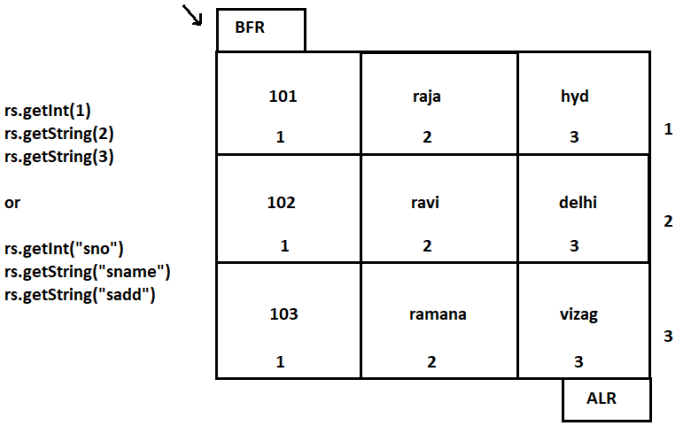
If next position is a record then it will return true.

If next position is ALR then it will return false.

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

In order to read the data from record ResultSet we need to use rs.getXxx() methods with index numbers or column names.

Here rs.getXxx() method means i.e rs.getInt(), rs.getString(), rs.getFloat(), rs.getDouble() and etc.



Types of queries in JDBC

==========

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

1) Select Queries

2) Non-Select Queries

1) Select Queries

Select queries give bunch of records from database software.
ex:
```
select * from student;
```

A JDBC Statement object gave executeQuery() method to execute select queries.
ex:
```
ResultSet rs=st.executeQuery("select * from student");
```

2) Non-Select Queries

Non-select queries give numeric value representing number of records effected in a database table.
ex:
```
delete from student; // 3
```

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

ex:
```
int result=st.executeUpdate("delete from student");
```

Type4 JDBC Driver / Native Protocol
==========

Driver class : oracle.jdbc.driver.OracleDriver
---------- ----------
pkg name Driver-classname

Driver url : jdbc:oracle:thin:@localhost:1521:XE
---------- | | |
sub protocol hostname portno logical_database_name

Database username : system

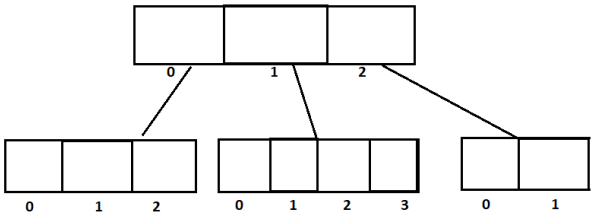
Database password : admin

Interview Question
==========

Jagged Array
==========
Jagged array is also known as array of arrays.

It's a multidimensional array where each row can have a different length.

```
int[][]  arr = new int[3][];
arr[0]=new int[3];
arr[1]=new int[4];
arr[2]=new int[2];
```



ex:

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

		int[][] arr=new int[3][];

		arr[0]=new int[3];
		arr[1]=new int[4];
		arr[2]=new int[2];

		for(int i=0;i<arr.length;i++)
		{
			for(int j=0;j<arr[i].length;j++)
			{
				System.out.println("Enter the element of arr["+i+"]["+j+"] :");
				arr[i][j]=sc.nextInt();
			}
		}

		//reading
		for(int[] a:arr)
		{
			for(int i:a)
			{
				System.out.print(i+" ");
			}
			//new line
			System.out.println();
		}
```

```
        }
}
```

How many steps are there to develop JDBC Application

==========

There are 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 jdbc connection objects.

Eclipse

==========

IDE : JEE

Environment : Java

Vendor : Eclipse Foundation

Website : www.eclipse.org

File Format : Zip

Download link :

https://drive.google.com/file/d/1c8TAX048EjAubIFByqZ0DzWZI3oKuauR/view?usp=drive_link

Steps to develop first JDBC application to read the record from student table

==========

step1:

Create a student table with records.

ex:

```
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;
```

step2:

Launch eclipse IDE by choosing workspace location.

step3:

Create a java project i.e IH-JAVA-037.
ex:
File --> new --> project --> java project --> Next -->
Name : IH-JAVA-037 --> Next -> Finish.

step4:

Add "ojdbc14.jar" file in project build path.
(C:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib)
ex:
right click to IH-JAVA-037 --> build path --> configure build path -->
libraries ---> classpath ---> Add external jar's ---> select ojdbc14.jar
--> open --> apply and close.

step5:

Create a "com.ihub.www" package inside "src" folder.
ex:
right click to src folder --> new --> package -->
Name : com.ihub.www --> finish.

step6:

Create a SelectApp.java file inside "com.ihub.www" package.
ex:
right click to com.ihub.www --> new --> class --->
Name : SelectApp --> Finish.

SelectApp.java

```
package com.ihub.www;

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","system","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();
	}
}
```

step7:

Run JDBC 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 no :");
		int no=sc.nextInt();

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

		String qry="select * from student where sno="+no;

		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();
	}
}
```

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","system","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 no?

```
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();

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

		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","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 the record based on student number ?

```
package com.ihub.www;

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

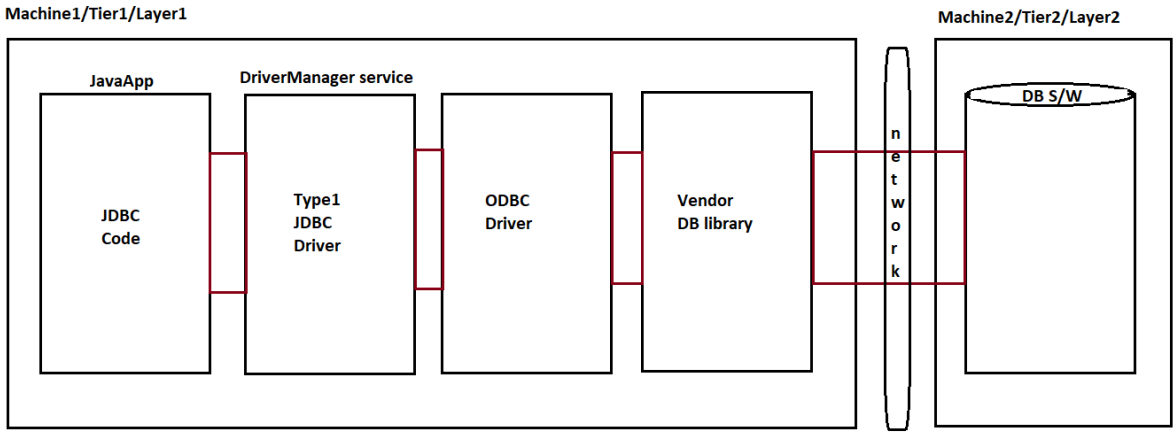
```
import java.util.Scanner;

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","system","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.



Advantages:

1) It is built-in driver of JDK.

2) Using Type1 JDBC driver we can interact with any database software.

Disadvantages:

1) This driver performance is low and it is not suitable for medium and large scale projects.Hence it is not a industry standard driver.

2) To work with Type1 JDBC driver we need to arrange ODBC driver and Vendor DB library seperately.

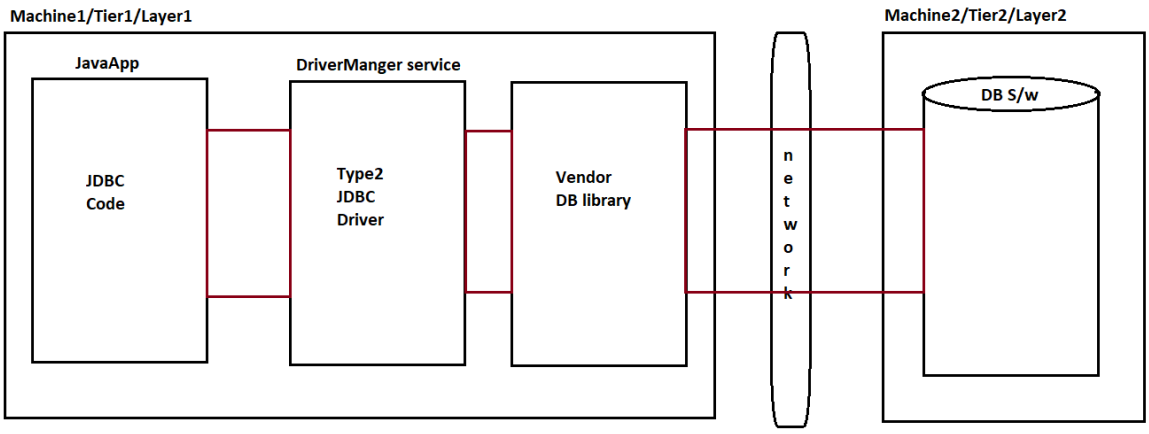
3) Since ODBC driver and vendor db library present at client side so it is not suitable for untrusted applets to database communication.

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.



Advantages:

1) This driver will give better performance compare to Type1 JDBC driver.

2) Type2 JDBC driver will not take the support of ODBC Driver.

Disadvantages:

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

2) To work with Type2 JDBC driver we need to arrange vendor db library seperately.

3) Since vendor db library present at client side so it is not suitable for untrusted applets to database communication.

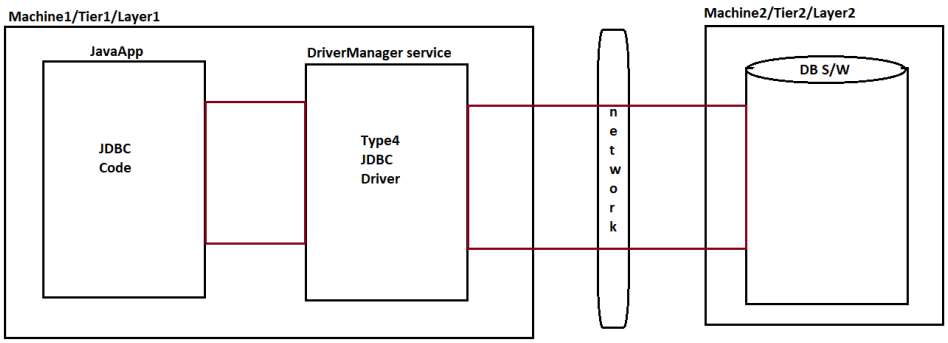
4) For every database we need to arrange type2 jdbc driver seperately.

Type4 JDBC Driver Architecture / Native Protocol (Java driver) / Thin 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.



Advantages:

5) This driver will give better performance compare to Type1 and Type2 JDBC Driver.

2) This driver is completely developed in java so it will give platform independency.

3) This driver will not take the support of ODBC driver and vendor db library.

4) Since odbc driver and vendor db library not present at client side so it is suitable for untrusted applets to database communication.

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

Disadvantages:

==========

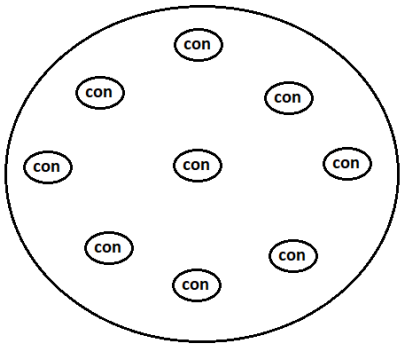
1) It is not a built-in driver of JDK.

2) For every database software 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

JDBC Connection pool represent connectivity with same database software.

Advantages:

> It gives reusable JDBC Connection objects.

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

> A programmer is not responsible to create, manage and destroy JDBC Connection objects in JDBC Connection pool.

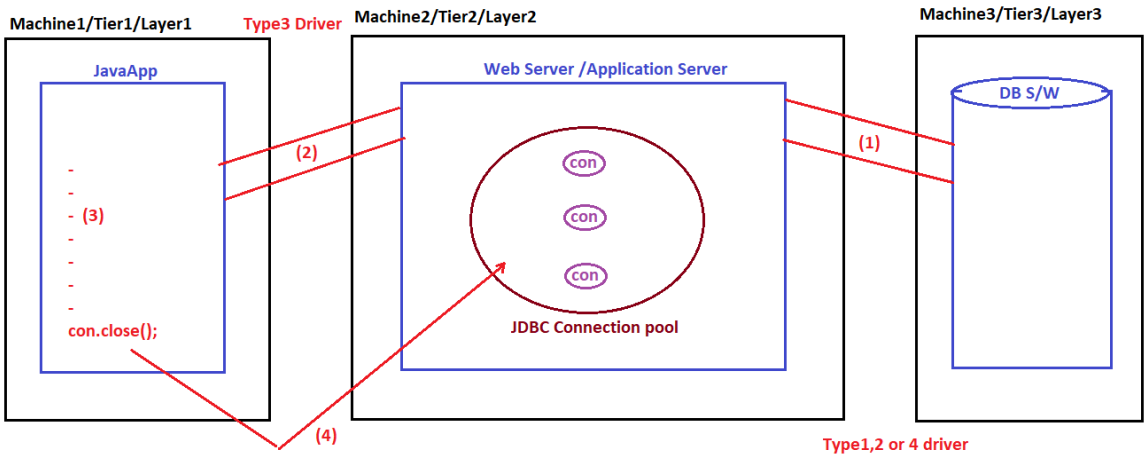
Type3 JDBC Driver Architecture / Net Protocol

==========

Webserver or proxy server or IDE's server contains set of readily available JDBC Connection objects in JDBC Connection pool.

Type3 JDBC driver is not designed to interact with databse software directly.

It is designed to interact with web server or proxy server to get one JDBC Connection object from JDBC Connection object.



With respect to the diagram:

1) Webserver or proxy server interacts with database software and gets reusable JDBC Connection objects in JDBC Connection pool.

2) Our application interacts with web server or proxy server and gets one reusable JDBC Connection object.

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

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

Types of JDBC Connection objects

==========

We have two types of Connection objects.

1) Direct JDBC Connection object

2) Pooled JDBC Connection object

1) Direct JDBC Connection object

A JDBC Connection object which is created by the user is called direct JDBC Connection object.
ex:

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

2) Pooled JDBC Connection object

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

Types of Statement objects

==========
We have three Statement objects.

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 instruction which change the behaviour of a query and behaviour of an application is called SQL injection problem.

Here special SQL instruction means comment 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.

input:

username : raja'--

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');

commit;
```

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();

		//converting 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","system","admin");
		Statement st=con.createStatement();

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

		ResultSet rs=st.executeQuery(qry);

		int result=0;
		while(rs.next())
		{
```

```
                result=rs.getInt(1);
            }

            if(result==0)
                System.out.println("Invalid Credentials");
            else
                System.out.println("Valid Credentials");

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

        }
}
```

Limitations with Simple Statement object

==========

1) It is not suitable to execute same query for multiple times with same or different values.

2) It raises SQL Injection problem.

3) We can't use string inputs directly to query parameter without any conversion.

4) Framing query with variables is quit complex.

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

6) It does not allow us to insert LOB(Large Object) values to databaset table column.

To overcome this 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 pre-compiled 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 pre-compiled SQL query.
ex:
ps.executeUpdate();

step5:

Close PreparedStatement object.

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","system","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);

                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 no?

```
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","system","admin");

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

                PreparedStatement ps=con.prepareStatement(qry);

                //set the values
```

```
		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?

```
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","system","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 SolForSSQLInjProbApp
{
	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","system","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 query
		ResultSet rs=ps.executeQuery();

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

```
                        System.out.println("Valid Credentials");

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

        }
}
```

DatabaseMetaData

==========

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 by using getMetaData() method of Connection object.
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","system","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, name of the columns, type of columns,
size of columns and etc.

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

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","system","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(1));
		System.out.println(rsmd.getColumnTypeName(2));
		System.out.println(rsmd.getColumnDisplaySize(2));

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

Interview Question

==========

Q) Write a java program to display sub array equals to given sum?

input:

arr = 1 2 3 7 6

sum = 12

output:

2 3 7

ex:

```
package com.ihub.www;

public class SubArray
{
	public static void main(String[] args)
	{
		int[] arr={1,2,3,7,5};
		int sum=12;

		for(int start=0;start<arr.length;start++)
		{
			int currentSum=0;

			for(int end=start;end<arr.length;end++)
			{
				currentSum+=arr[end];

				if(sum == currentSum)
				{
					for(int k=start;k<=end;k++)
					{
						System.out.print(arr[k]+" ");
					}
				}

			}
		}
	}
}
```

approach2

```
package com.ihub.www;

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

```
		int[] arr={1,2,3,7,6};
		int sum=12;

		int cnt=0;

		for(int start=0;start<arr.length;start++)
		{
			int currentSum=0;

			for(int end=start;end<arr.length;end++)
			{
				currentSum+=arr[end];

				if(sum == currentSum)
				{
					cnt=1;

					for(int k=start;k<=end;k++)
					{
						System.out.print(arr[k]+" ");
					}
				}

			}
			if(cnt==1)
			{
				break;
			}
		}
	}
}
```

Working with Date values

==========

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

It is never recommanded to store date values in the form string because we can't compare two dates.

Every database software designed to support date values.

Every database software support different date patterns.
ex:

oracle - dd-MMM-yy
mysql - yyyy-MM-dd

Using simple Statement object we can't insert date values.

To overcome this limitation we need to use PreparedStatement object.

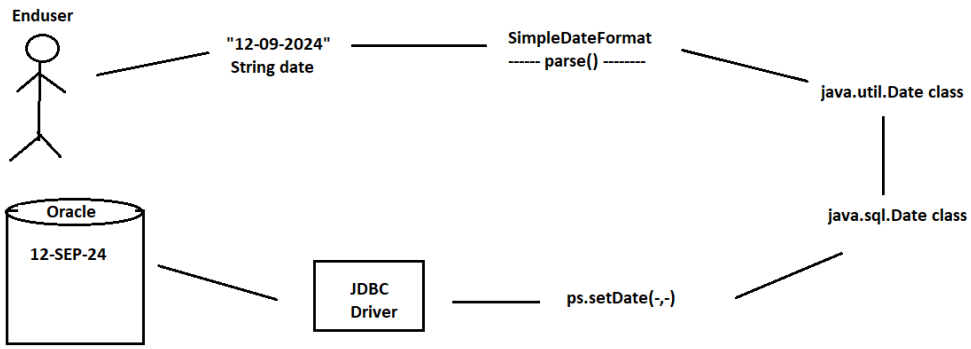
We can set date values to query parameter by using setDate(-,-) method.

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

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

Once JDBC driver gets the date value then it will insert date value in the pattern which is supported by underlying database software.

Standard procedure to insert date value



1) Enduser gives date value in the form of 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 gets the date value then it will insert date value in the pattern which is supported by underlying database 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 employee DOJ(dd-MM-yyyy) :");
		String sdoj=sc.next();

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

		//convert 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","system","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);

		//execute
		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.PreparedStatement;
import java.sql.ResultSet;
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","system","admin");

		String qry="select * from emp1";
		PreparedStatement ps=con.prepareStatement(qry);

		ResultSet rs=ps.executeQuery();
		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;

			//convert util date to string date
			SimpleDateFormat sdf=new SimpleDateFormat("dd-MM-yyyy");
			String sdoj=sdf.format(udoj);


			System.out.println(id+" "+name+" "+sdoj);
		}

		rs.close();
		ps.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, audio,video, avi file and etc.

2) CLOB (Character Large Object)

ex:

text file, advanced text file and etc

While dealing with matronial applications, job protal applications, profile management application and etc we need to insert and retrieve LOB values.

While dealing with simple Statement we can't place LOB value directly to query parameter.We need to use PreparedStatement object.

In PreparedStatement object we can set LOB values by using following methods.
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();
```

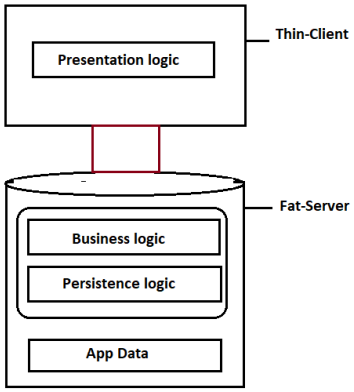
```
                System.out.println("Enter the employee name :");
                String name=sc.next();

                //locate a file
                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","system","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.PreparedStatement;
import java.sql.ResultSet;

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","system","admin");
```

```
                String qry="select * from emp2";
                PreparedStatement ps=con.prepareStatement(qry);

                ResultSet rs=ps.executeQuery();

                while(rs.next())
                {
                        InputStream is=rs.getBinaryStream(3);
                        FileOutputStream fos=new FileOutputStream("D:\\Training\\IH-JAVA-
038\\rock.jpg");

                        int byteReads=0;
                        byte[] buff=new byte[100];

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

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

        }
}
```

Thin-Client/Fat-Server application



Every JDBC Application is a thin-client/Fat-Server application.

To create a thin-client/fat-server application we need to store business logic and persistence logic in the 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","system","admin");

		CallableStatement cst=con.prepareCall("{CALL first_proc(?,?,?)}");

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

		//set the IN parameter
		cst.setInt(1, 20);
		cst.setInt(2, 30);

		//execute
		cst.execute();

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

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

Working with Properties file

==========

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

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

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

A properties file will store the data in the form of key and value.

dbdetails.properties

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

ex:

```
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 file
		FileInputStream fis=new FileInputStream("src/com/ihub/www/dbdetails.properties");

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

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

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

		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();
        }
}
```

JDBC Flexible Application

==========

Connection object is a heavy weight object and it is not recommaned to create Connection object in every jdbc application.

It is recommaned to create a seperate class which returns Connection object.

ex:

DBConnection.java

```
package com.ihub.www;

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

public class DBConnection
{
        static Connection con=null;

        private DBConnection()
        {
        }

        public static Connection getConnection()
        {
                try
                {
                        if(con==null)
                        {
                                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";
            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();
        }
}
```

Batch Processing

==========

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

To add the queries to the batch we need to use addBatch() method of Statement object.

To execute the batch we need to use executeBatch() method of Statement object.

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","system","admin");
		Statement st=con.createStatement();

		//declare the queries
		String qry1="insert into student values(103,'ramana','vizag')";
		String qry2="delete from student where sno=102";
		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[] result=st.executeBatch();

		//for each loop
		int sum=0;
		for(int i:result)
		{
			sum+=i;
		}

		System.out.println("No of records effected are = "+sum);
	}
}
```

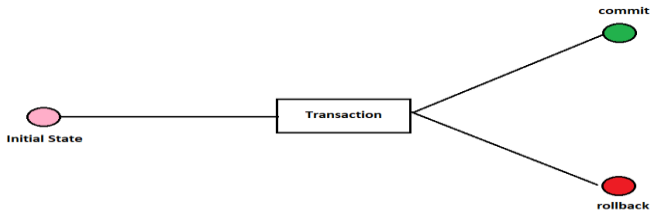
Transaction Management

==========

Transaction means a single unit of work.

We commit if transaction done successfully.

We rollback if transaction failed.



SBI TABLE:

```
drop table sbi;

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

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

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

commit;
```

KOTAK Table

```
==========
drop table kotak;

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

insert into kotak values(111111,'sharath',90000);

insert into kotak values(222222,'babu',80000);

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 number :");
		int sno=sc.nextInt();
```

```
                System.out.println("Enter the destination account number :");
                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","system","admin");

                //set auto complete off
                con.setAutoCommit(false);

                Statement st=con.createStatement();

                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[] result=st.executeBatch();

                boolean flag=true;

                for(int i:result)
                {
                        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();
        }
}
```

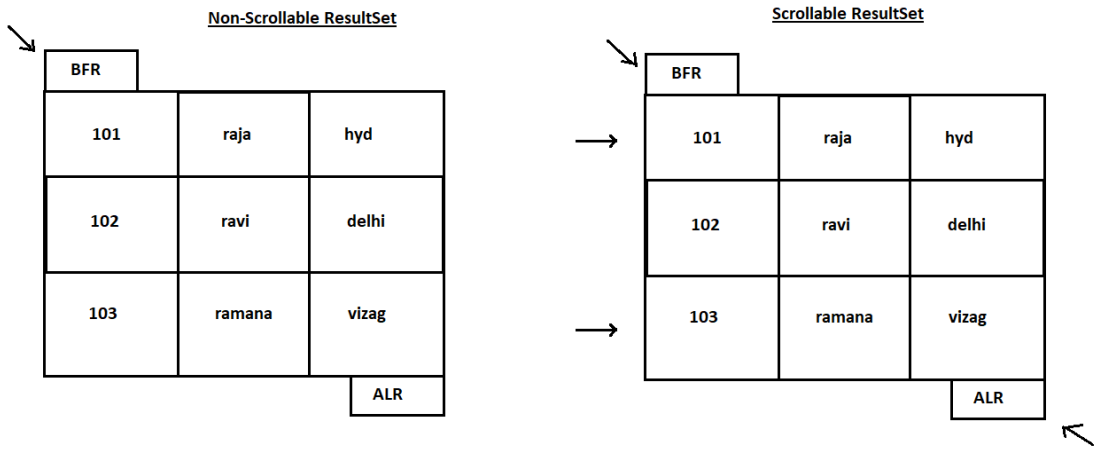
Types of ResultSet Object

==========

We have two types of ResultSet object.

1) Non-Scrollable ResultSet object

2) Scrollable ResultSet object



1) Non-Scrollable ResultSet object

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

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

If JDBC Statement object is create without TYPE , MODE value then that ResultSet object is called non-scrollable ResultSet object.

ex:

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

2) Scrollable ResultSet object

If a ResultSet object which allows us to read the records non-sequentially, bi-directionally or randomly is called Scrollalbe ResultSet object.

If JDBC Statement object is created with TYPE,MODE value then that ResultSet object is called Scrollable ResultSet object.

ex:

```
Statement st =con.createStatement(TYPE_VALUE,MODE_VALUE);
```

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

Note:

We have following list of TYPE values.

ex:

- ResultSet.TYPE_SCROLL_SENSITIVE
- ResultSet.TYPE_SCROLL_INSENSITIVE

We have following list of MODE values.

ex:

- ResultSet.CONCUR_UPDATABLE
- ResultSet.CONCUR_READ_ONLY

ex:

```
package com.ihub.www;

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

public class ScrollableRSApp
{
	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","admin");
		Statement st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_READ_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));
		}

		//bottom to top
		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.getRow()+" "+rs.getInt(1)+" "+rs.getString(2)+"
"+rs.getString(3));

                rs.last();
                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));


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

Steps to interact with MYSQL Database

==========

https://repo1.maven.org/maven2/com/mysql/mysql-connector-j/8.0.31/

step1:

Download and Install MYSQL Database.

ex:

https://dev.mysql.com/downloads/installer/

step2:

Copy "bin" directory of mysql database and paste in environmental variables.

ex:

JDBC Application to interact with MySQL Database

==========

step1:

Download and Installed MySQL database software.

ex:

https://dev.mysql.com/downloads/installer/

step2:

Connect with mysql by using password.

ex:
username : root(default)
password: root

step3:
---
create a SCHEMA in MYSQL.
ex:
create schema IH_JAVA_036

step4:

To check list of databases /schemas present in mysql db.
ex:
show databases;

step5:

Use IH_JAVA_014 scheme/database.
ex:
use IH_JAVA_036;

step6:
---
create a student table and insert the records.
ex:
create table student(sno int(3),sname varchar(10),sadd varchar(10));
insert into student values(101,'raja','hyd');
insert into student values(102,'raju','delhi');
insert into student values(103,'ravi','pune');
commit;

step7:
---
create a JDBC Application to select student records.

MySQLApp.java

```
package com.ge.www;

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

public class MySQLApp {

	public static void main(String[] args) {

		final String DRIVER="com.mysql.jdbc.Driver";
		final String URL="jdbc:mysql://localhost:3306/IH_JAVA_036";
		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();
		}

	}

}
```

step8:

Add "mysql-connector.jar" file in project build path for mysql database.

right click to project --> built path --> configuration build path --> libraries
--> add external jars --> select mysql-connector.jar file --> open.

jar file download :

https://repo1.maven.org/maven2/com/mysql/mysql-connector-j/8.0.31/

Note:
ojdbc14.jar - - for oracle
mysql-connector.jar --> for mysql

step9:

Run the jdbc application.

JDBC Application to interact with MongoDB Database

==========

step1:

Download and install MongoDB Community Server.

ex:

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

step2:

Downoad and install MongoDB Shell.

ex:

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

step3:

Extract Mongodb shell inside "MongoDB" folder.

ex:

C:\Program Files\MongoDB

step4:

Copy "bin" directory of mongoshell.

ex:

C:\Program Files\MongoDB\mongosh-2.3.0-win32-x64\bin

step5:

Paste "bin" directory in environmental variables.

ex:

right click to This PC --> properties --> Advanced System Settings --> Environmental variables --> System variables --> click to path --> click to edit button --> New --> paste (C:\Program Files\MongoDB\mongosh-2.3.0-win32-x64\bin) -->ok

--->ok --->ok.

step6:

Launch Eclipse IDE by choosing workspace location.

step7:

Create MongoDBProj inside eclipse IDE.

ex:

File --> New --> Java project --> Name : MongoDBProj --> Next --> Finish.

step8:

Download and add mongodb jar file in project build path.

ex:

jar file link : https://repo1.maven.org/maven2/org/mongodb/mongo-java-driver/3.11.2/

right click to MongoDBProj --> Buildpath --> configure build path --> libraries --> click to classpath --> add external jars --> select mongo-java-driver.jar file --> Open --> Apply and close.

step9:

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

step10:

Create a InsertApp.java file inside "com.ihub.www" package.

SelectApp.java

```
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 InsertApp
{
	public static void main(String[] args)
	{
		try(MongoClient mongoClient=MongoClients.create("mongodb://localhost:27017");)
		{
			MongoDatabase mongoDatabase=mongoClient.getDatabase("myDatabase");

			MongoCollection<Document>
mongoCollection=mongoDatabase.getCollection("myCollection");

			Document doc=new Document("id", 101)
					.append("name","Alan")
					.append("add","Hyd");

			mongoCollection.insertOne(doc);

			System.out.println("Record Inserted in MongoDB");

		}
```

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

step11:

Run the jdbc application.

Q) Write a jdbc application to read the record from mongodb?

```
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 SelectApp
{
	public static void main(String[] args)
	{
		try(MongoClient mongoClient=MongoClients.create("mongodb://localhost:27017");)
		{
			MongoDatabase mongoDatabase=mongoClient.getDatabase("myDatabase");

			MongoCollection<Document>
mongoCollection=mongoDatabase.getCollection("myCollection");

			Document findDocument=mongoCollection.find(new
Document("id",101)).first();

			System.out.println(findDocument.toJson());

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