

James Gosling OC (born 19 May 1955) is a Canadian computer scientist, best known as the founder and lead designer behind the Java programming language. Gosling was elected a member of the National Academy of Engineering in 2004 for the conception and development of the architecture for the Java programming language and for contributions to window systems.

James Gosling

ADVANCE JAVA

JDBC

- As if now it is known as trademark.
- But earlier it is also known as Java Database Connectivity.
- RAM is a temporary storage device or medium.
- During the program execution our data will store in a RAM and once the program execution is completed we will loss the data.
- To overcome this limitation we are making application writing the data into a file or database software.
- Files and database softwares act like a permanent storage device or medium.

Persistence

- The process of storing and managing the data for a long 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.

ex:

Files

Database softwares

2) Persistence data

- Data of a persistence store is called persistence data.

ex:

Records

tables

3) Persistence operation

- Insert,update,delete,create and etc are called persistence operations.
- In the realtime this operation is also known as CURD/CRUD/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
JPA code



5) Persistence technology

- A technology which is used to develop persistence logic is called persistence technology.

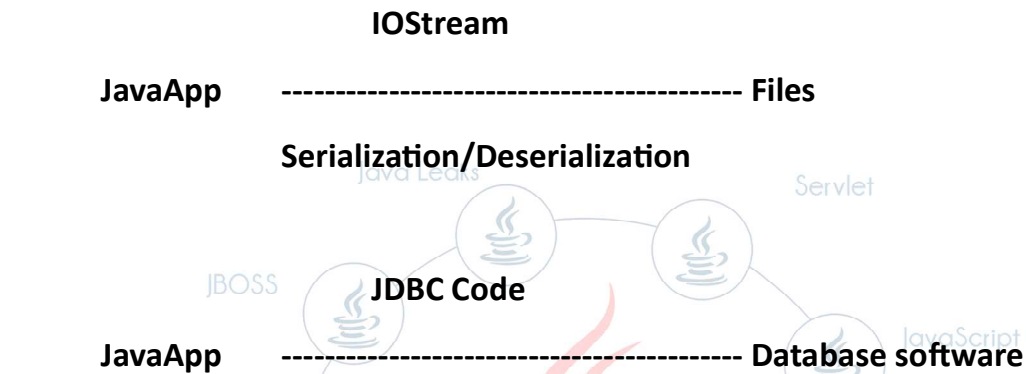
ex:

JDBC
Hibernate
EJB
and etc.

Q)What is JDBC ?

JDBC is a persistence technology which is used to develop persistence logic having the capability to perform persistence operations on persistence data of a persistence store.

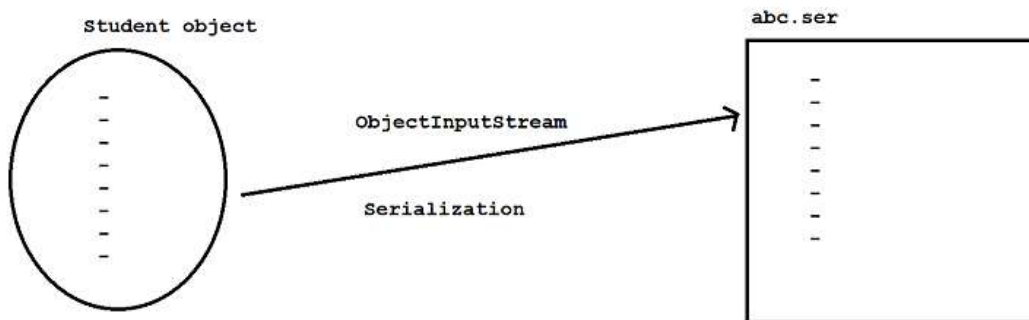
Note:



Serialization

- The process of taking object data and storing a file is called serialization.
- In general, it converts object state to file state.

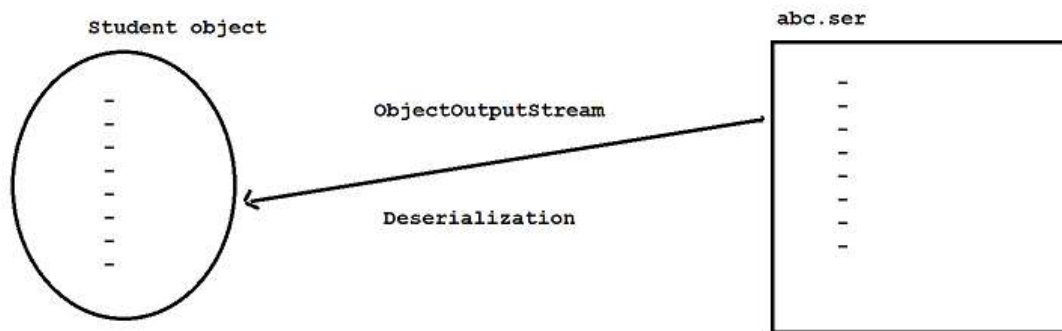
Diagram: jdbc1.1



Deserialization

- The process of taking the data from file and representing an object is called deserialization.
- In general, it converts file state to object state.

Diagram: jdbc1.2



Limitations with File as a persistence store

- It will store limited amount of data.
- There is no security.
- Fetch 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 not null, unique, primary key and etc.
- Updation and Deletion of data can't be done directly.
- Merging and comparison of data can't be done easily.

Advantages of database as a persistence store

- We can store unlimited amount of data.
- There is a security.
- It supports common query language.
- Fetching the data with multiple conditions is possible.
- It show an application with relationships.
- It allows us to apply constraints.
- Updation and Deletion of data can be done directly.
- Merging and comparison of data can be done easily.

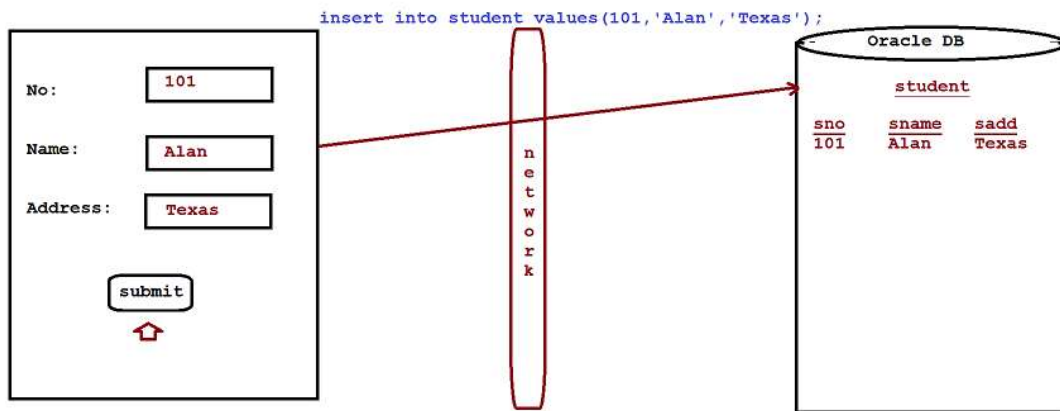
Every JDBC application is a two tier application where java with JDBC code acts like a frontend/tier1/layer1 and database software acts like a backend/tier2/layer2.

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 performs some operations based on the instructions given by frontend is called backend.

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

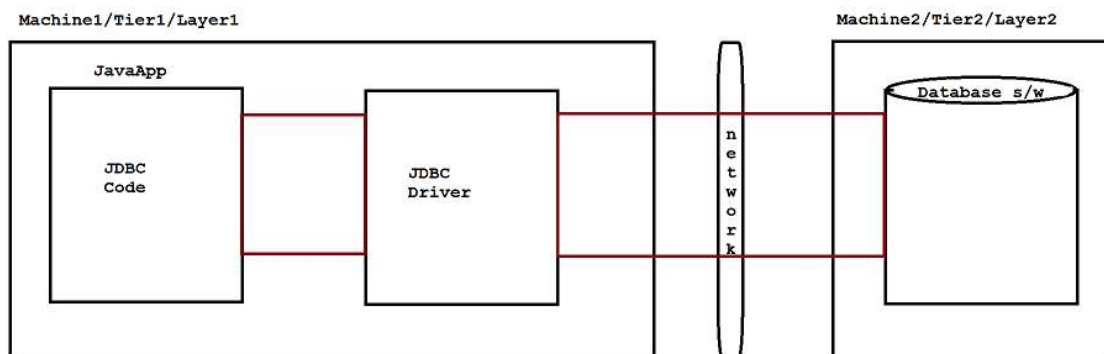
Diagram: jdbc1.3



JDBC Driver

- It acts like a bridge between java application and database software.
- It is used to convert java calls to database calls and vice versa.
- Here calls means instructions.

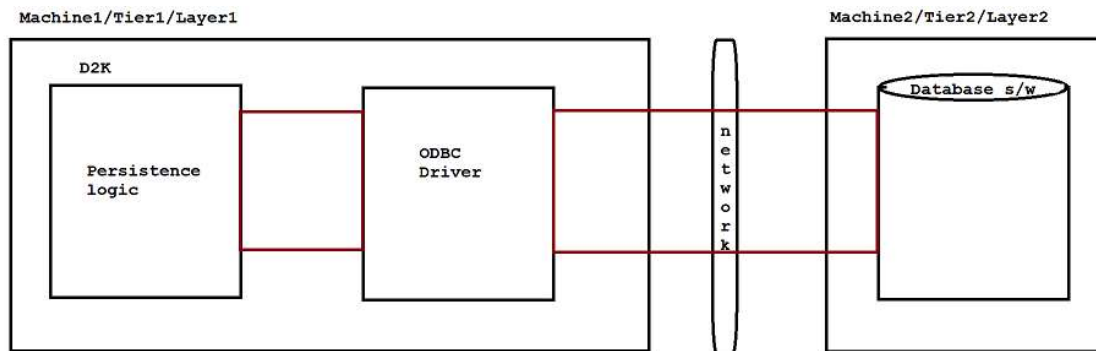
Diagram: jdbc1.4



ODBC Driver

Perl, D2K, VB script and etc uses ODBC driver to locate and communicate with database software.

Diagram: jdbc1.5



ODBC driver is developed in C language by taking the support of pointers.

But java does not supports pointers. Hence we can't take ODBC driver to interact with database software.

To overcome this limitation Sun micro system introduced JDBC driver exclusively.

We will get JDBC softwares from three parties.

- 1) Sun micro system (creator of JDBC driver)
- 2) Database vendor
- 3) Third party vendor

We will get ODBC softwares from three 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 which is having set of rules and guidelines to develop jdbc drivers for multiple database softwares.

Q)What is ODBC?

ODBC is a open technology given by Xopen company which is having set of rules and guidelines to develop ODBC drivers for multiple database softwares.

Q)How many drivers are there in JDBC?

We have four drivers in JDBC.

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

Oracle Database

version	:	10g or 11g
vendor	:	Oracle Corporation
Opensource	:	Open source
website	:	www.oracle.com
Default portno	:	1521
username	:	system (default)
password	:	admin (custom)

Download link

https://drive.google.com/file/d/0B9rC21sL6v0td1NDZXpkUy1oMm8/view?usp=drive_link&resourcekey=0-aKooR3NmAh eLo qGw inA

- 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 always recommended to use 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-classname");
```

Connection object

- A 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.

ex:

```
Connection con;
```

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

DriverManager.getConnection()

- A DriverManager is a class which is present in java.sql package.
- A getConnection() is a static method which returns a JDBC Connection object representing connectivity between java application and database software.

ex:

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

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

ex:

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

Statement object

- A Statement is an interface which is present in java.sql package.
- It acts like a vehical between java application and database software.
- It is used to sends and execute SQL query in database software.

We can create Statement object as follow.

ex:

Statement st=con.createStatement();

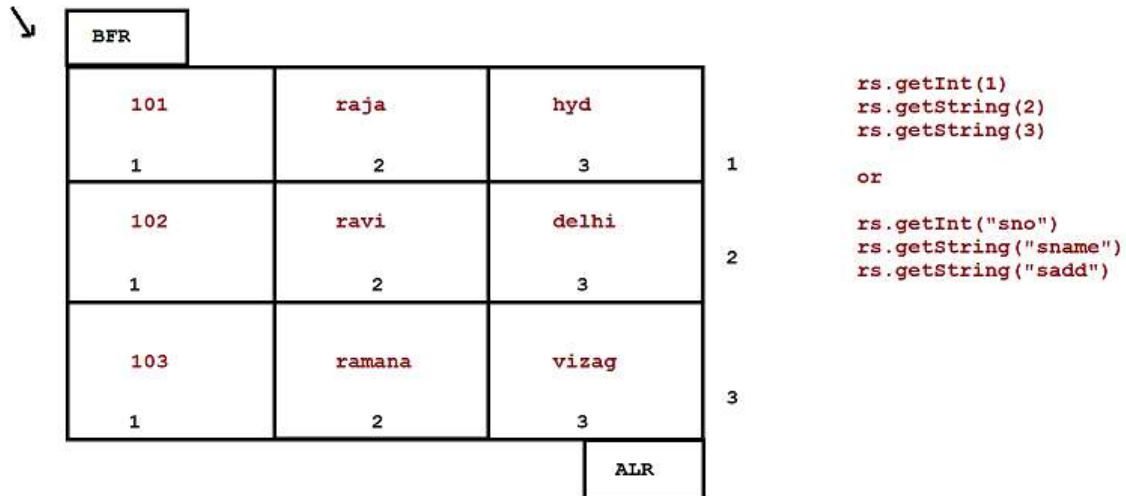
ResultSet object

- A ResultSet is an interface which is present in java.sql package.
- Every ResultSet object contains two positions.
- ✓ BFR (Before First Record/Row)
- ✓ ALR (After Last Record/Row)
- ✓
- By default every record pointer points to BFR position.
- Every record ResultSet contains 1 as base index and every record ResultSet column contains 1 as based index.

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 a ALR then it will return false.
- To read the values from record ResultSet we need to use getXxx() method with index number or column names.
- Here getXxx() method means getInt(),getString(),getFloat(),getDouble() and etc.

Diagram: jdbc2.1



Types of Queries

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

- Select Query
- Non-Select Query

Select Query

It will give bunch of records from database software.

ex:

```
select * from student;
```

To execute select query we need to use `executeQuery()` method of Statement object.

ex:

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

Non-Select Query

- It will give numeric value representing number of records effected in database table.

ex:

delete from student;

To execute non-select query we need to use executeUpdate() method of Statement object.

ex:

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

Type4 JDBC Driver Database Properties

Driver class : oracle.jdbc.driver.OracleDriver

pkg name driver class name

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

There are six steps to develop JDBC application.

- ✓ Register JDBC driver with DriverManager service.
- ✓ Establish the connection with database software.
- ✓ Create Statement object.
- ✓ Sends and Executes SQL query in database software.
- ✓ Gather the result from database software to process the result.
- ✓ Close all jdbc connection objects.

student table

drop table student;

create table student(sno number(3),sname varchar2(10),sadd varchar2(12));

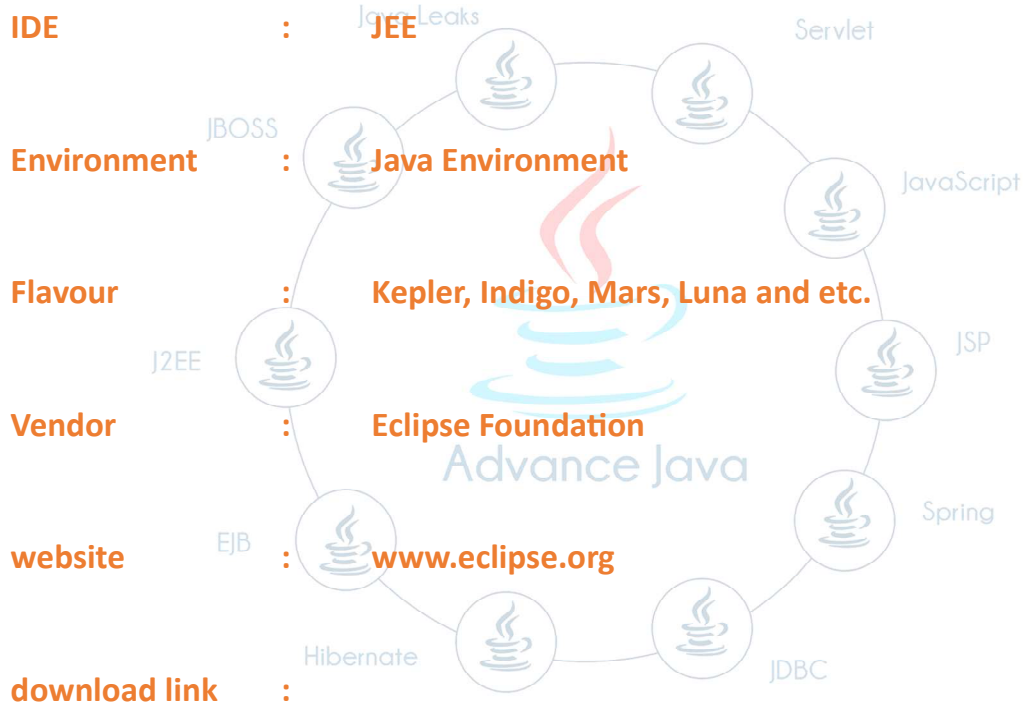
insert into student values(101,'raja','hyd');

insert into student values(102,'ravi','delhi');

insert into student values(103,'ramana','vizag');

commit;

Eclipse



https://drive.google.com/file/d/1p42SGx_hmm52Gw4WsuVvkrRNbxO-bjww/view?usp=drive_link

First JDBC application to select the record from student table

step1:

Launch eclipse IDE by choosing workspace location.

step2:

Create a java project i.e "IH-JAVA-018".

ex:

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

Name : IH-JAVA-018 --> next --> Finish.

step3:

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

ex:

Right click to IH-JAVA-018 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 -->.

package name : com.iHub.www --> finish.

step5:

Create a "SelectApp1.java" file inside "src/com.iHub.www" package.

ex:

right click to com.iHub.www package --> new --> class -->

Name : SelectApp1 --> 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 SelectApp1
```

```
{
```

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

```
    {
```

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

```
        Connection
```

```
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","s  
ystem","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.

ex:2

```
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 SelectApp1
```

```
{
```

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

```
    {
```

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

```
        Connection
```

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

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

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

```
        while(rs.next())
```

```
        {
```

```
            System.out.println(rs.getInt("sno")+ "  
"+rs.getString("sname")+ " "+rs.getString("sadd"));
```



```

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

```

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","s
ystem","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");
```

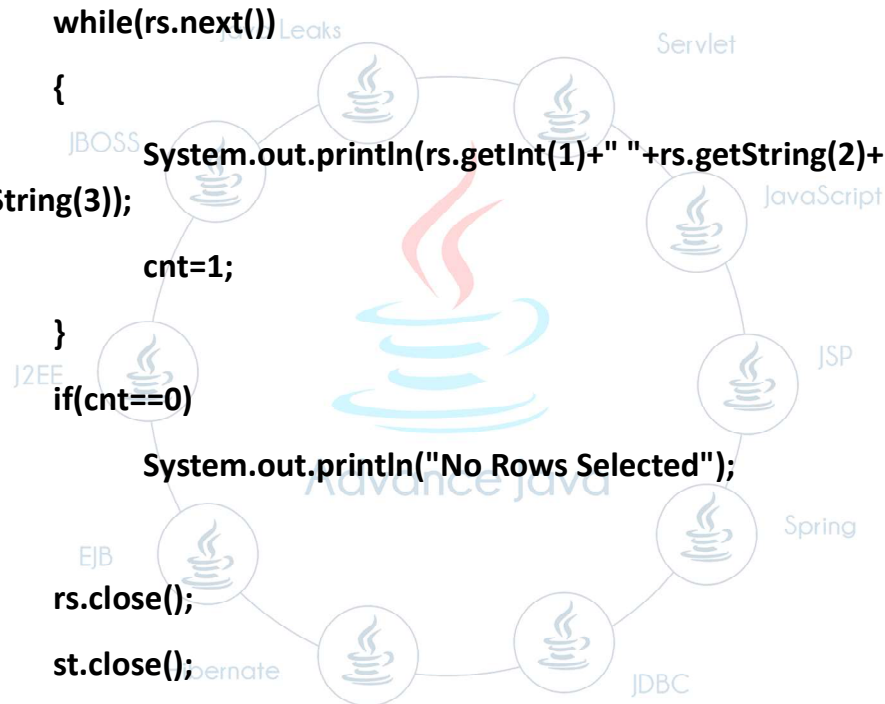
```
    rs.close();
```

```
    st.close();
```

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

```
}
```

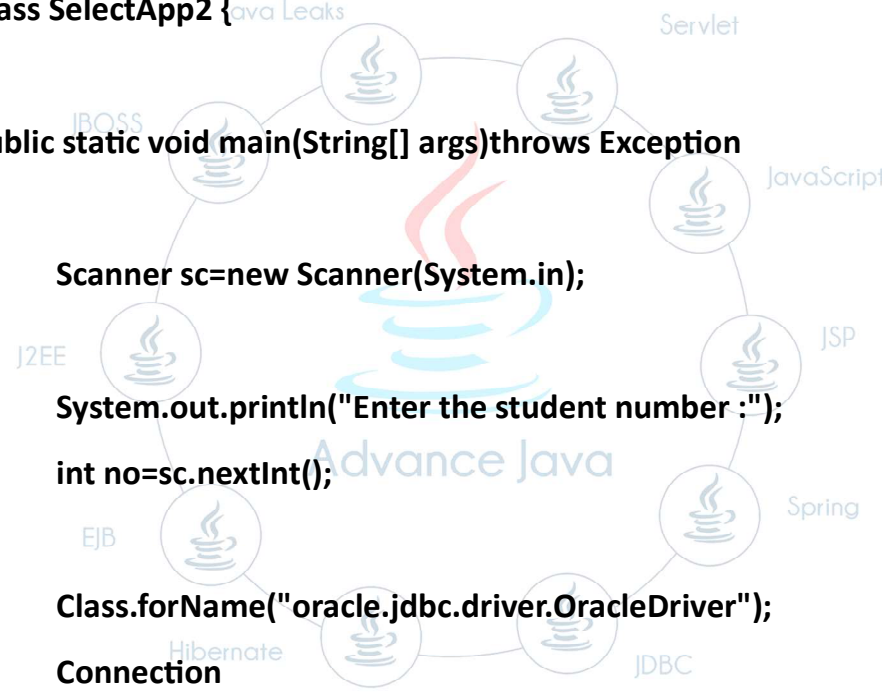
```
}
```



ex:2

```
package com.ihub.www;  
  
import java.sql.Connection;  
  
import java.sql.DriverManager;  
  
import java.sql.ResultSet;  
  
import java.sql.Statement;  
  
import java.util.Scanner;
```

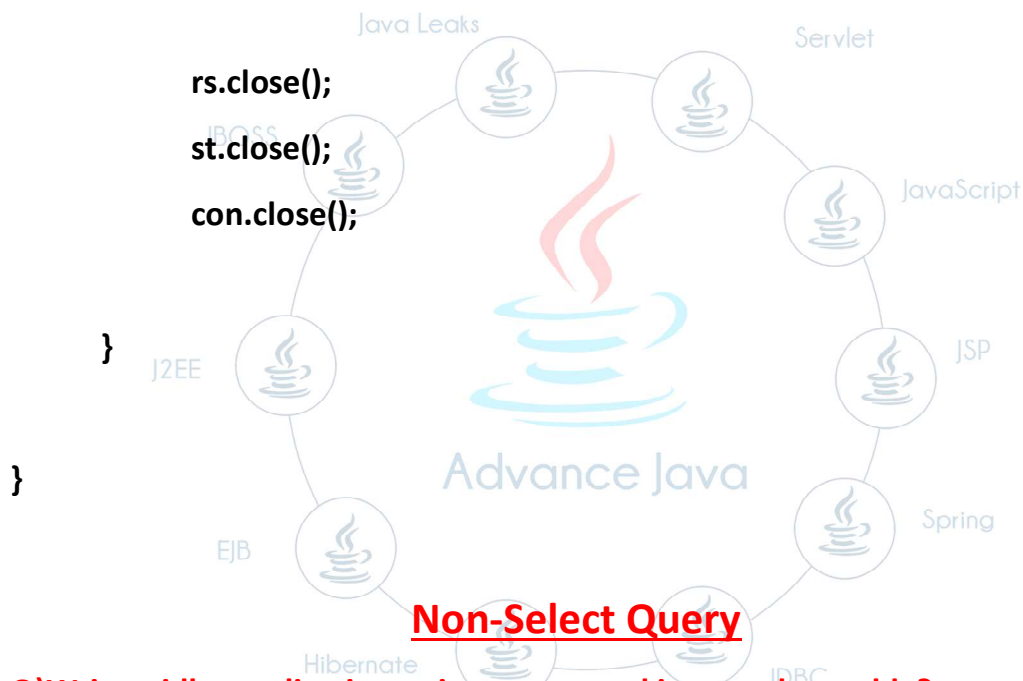
```
public class SelectApp2 {  
  
    public static void main(String[] args)throws Exception  
    {  
  
        Scanner sc=new Scanner(System.in);  
  
        System.out.println("Enter the student number :");  
        int no=sc.nextInt();  
  
        Class.forName("oracle.jdbc.driver.OracleDriver");  
        Connection  
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","s  
ystem","admin");  
  
        Statement st=con.createStatement();  
  
        String qry="select sname,sadd from student where sno="+no;  
  
        ResultSet rs=st.executeQuery(qry);  
  
        int cnt=0;
```



```

while(rs.next())
{
    System.out.println(rs.getString(1)+" "+rs.getString(2));
    cnt=1;
}
if(cnt==0)
    System.out.println("No Rows Selected");

```



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 out string inputs as per the query.
    name=""+name+"";
    add=""+add+"";

    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","s
ystem","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 name :");
    }
}

```

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

```
System.out.println("Enter the student no :");
```

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

```
//convert string 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 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");
```

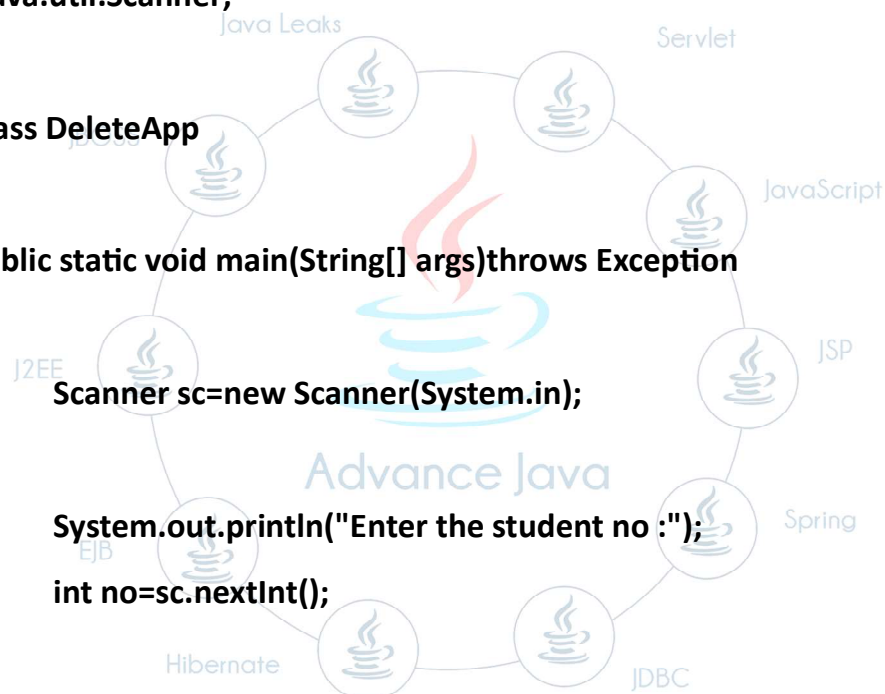
```
        Connection
```

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

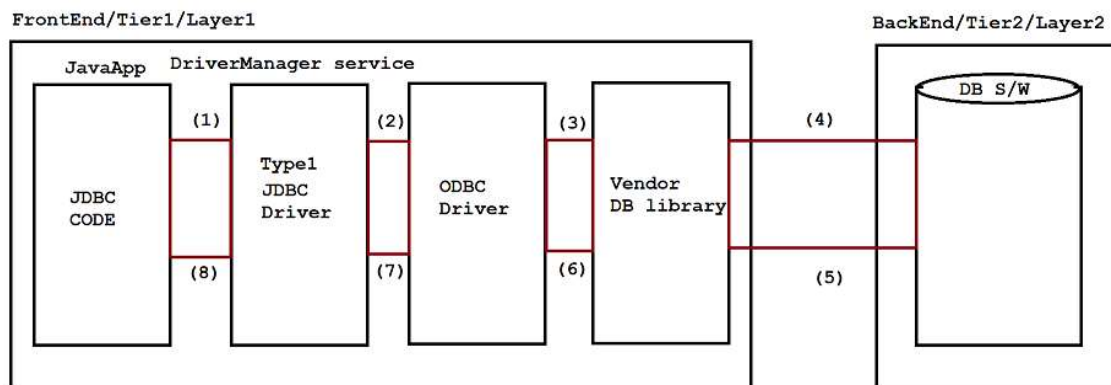
}
}

```

Type1 JDBC Driver Architecture / JDBC-ODBC Bridge Driver

- 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.

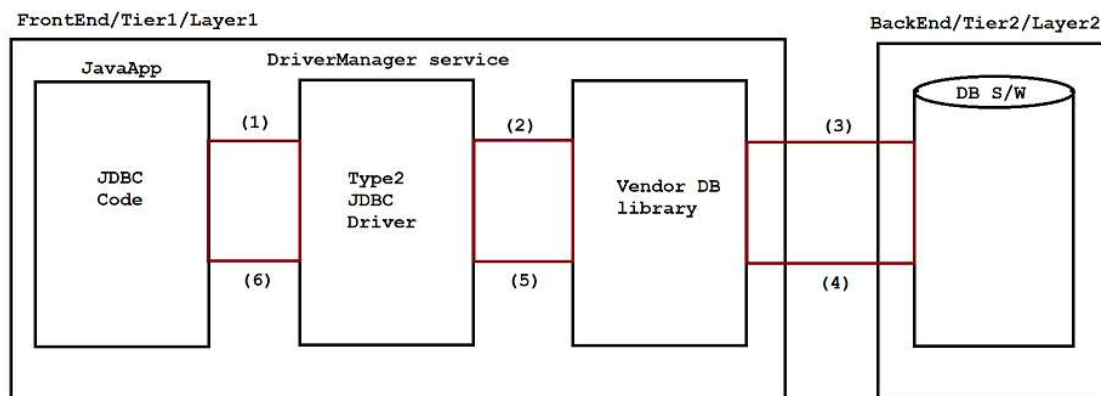
Disadvantages:

- Its performance is low. It is not suitable for medium and large scale projects. Hence it is not an industry standard driver.
- Since ODBC driver and vendor DB library present on the client side, so it is not suitable.
- To perform untrusted applets to database communication.
- To work with type1 jdbc driver we need to arrange ODBC driver and Vendor DB library separately.

Type2 JDBC Driver Architecture / Native API

- 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 multiple database softwares.

Diagram: jdbc3.2



Advantages:

- This driver will give better performance compared to type1 jdbc driver.
- Type2 jdbc driver will not take the support of ODBC driver.

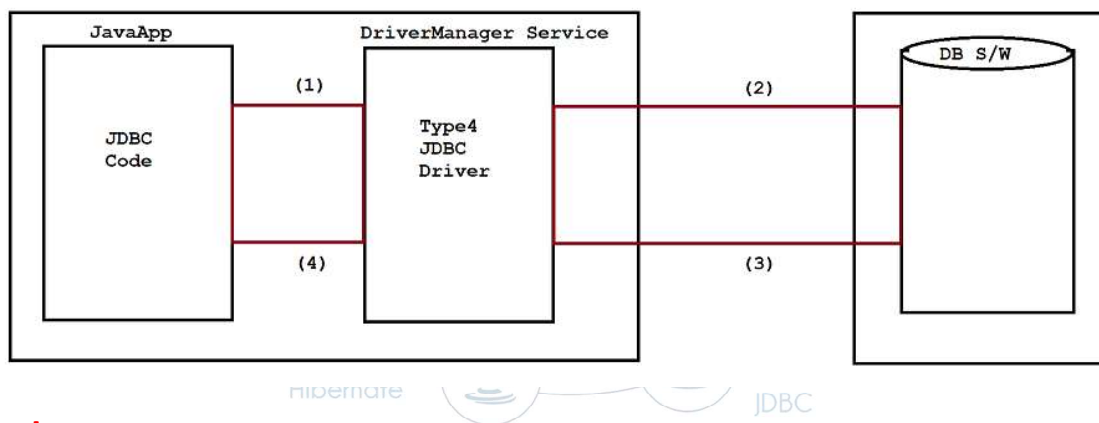
Disadvantages:

- Type2 driver performance is bit slow.It is not suitable for medium and large scale projects.Hence it is not a industry standard driver.
- Since vendor db library present at client side so it is not suitable to perform
- untrusted applets to database communication.
- To work with type2 jdbc driver we need to arrange vendor db library separately.
- For every database software we need to use type2 jdbc driver.

Type4 JDBC Driver Architecture / Native Protocol (Java Driver) / thin driver

- Type4 JDBC driver is not designed to take the support of ODBC driver or vendor db library.
- It is designed to locate and interact with database software directly.

Diagram: jdbc3.3



Advantages:

- This driver will give better performance when compare to type1 and type2 driver.
- This driver is completely developed in java so it will give platform independency.
- Since ODBC driver and vendor db library not present at client side so it is suitable for untrusted applets to database communication.
- It is suitable for medium and large scale projects.Hence it is a industry standard driver.
- It will not take the support of ODBC driver and vendor db library.

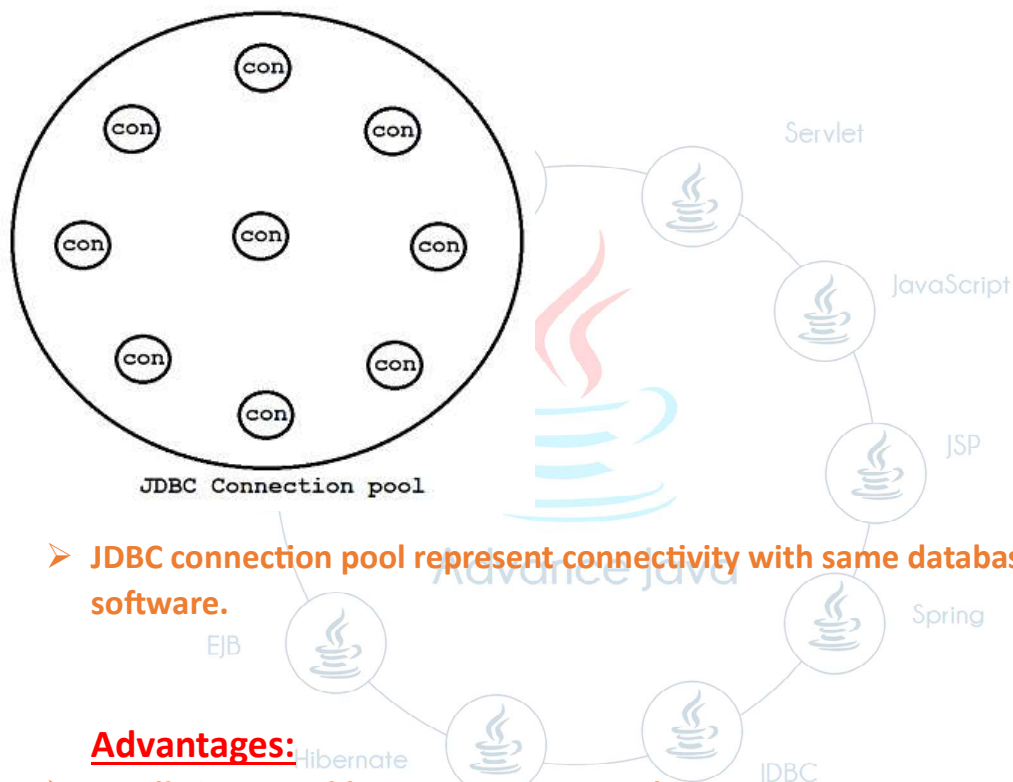
Disadvantages:

- It is not a built-in driver of JDK.
- For every database we need to arrange Type4 jdbc driver separately.

JDBC Connection pool

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

Diagram: jdbc3.4



- JDBC connection pool represent connectivity with same database software.

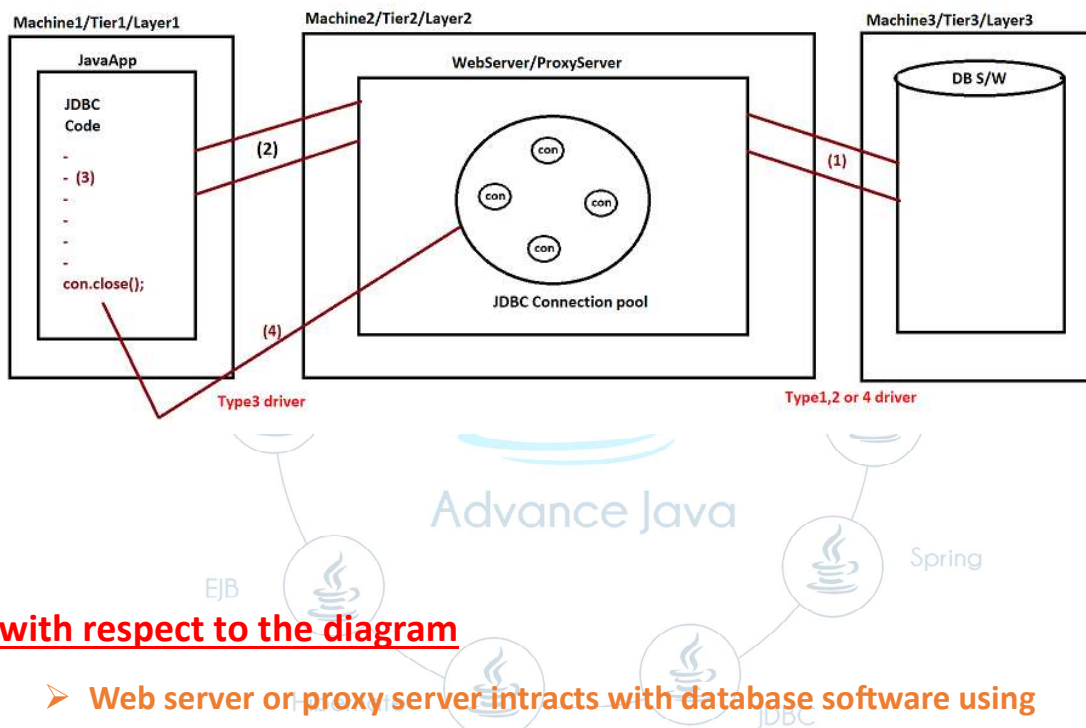
Advantages:

- It will give reusable JDBC Connection objects.
- With minimum number of jdbc Connection objects we can interact with multiple clients.
- A user is not responsible to create, manage or destroy jdbc Connection objects.
- A JDBC Connection pool is responsible to create, manage and destroy jdbc Connection objects.

Type3 JDBC Driver Architecture / Net Protocol

- Our web server or Proxy server or IDE's server contains JDBC Connection pool.
- Type3 JDBC driver is not designed to interact with database software directly.
- It is designed to interact with web server or proxy server to get one JDBC Connection object from JDBC connection pool.

Diagram: jdbc3.5



with respect to the diagram

- Web server or proxy server interacts with database software using type1,2 or 4 jdbc driver and gets JDBC Connections in JDBC connection pool.
- Our application uses type3 jdbc driver to interact with web server or proxy server to one reusable JDBC connection object from JDBC connection pool.
- Our application uses reusable connection object to create other JDBC Connection objects.
- Once if we call `con.close()` method then our connection object goes back to JDBC connection pool.

Q)Types of JDBC Connection objects ?

We have two types of JDBC Connection objects.

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 is called
- pooled jdbc connection object.

Q)How many Statement objects are there 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 or behaviour of an application is called SQL injection problem.

Here SQL instruction means comment in SQL i.e --.

ex:

username : raja'--

password : hyd

Valid Credentials

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

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 string as per inputs
```

```
        name="\""+name+"\"";
```

```
        pass="\""+pass+"\"";
```

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

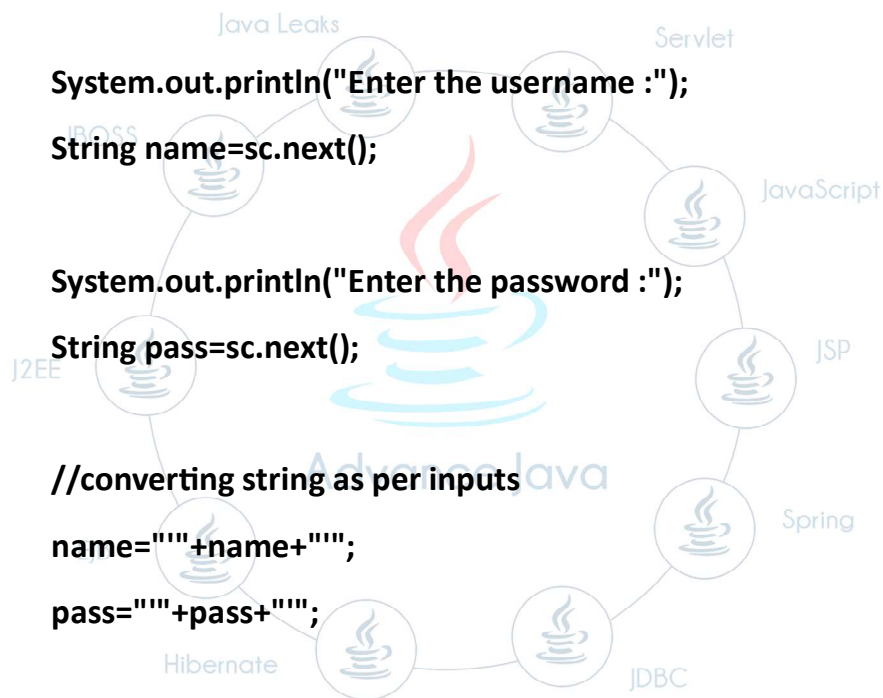
```
        Connection
```

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

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

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

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

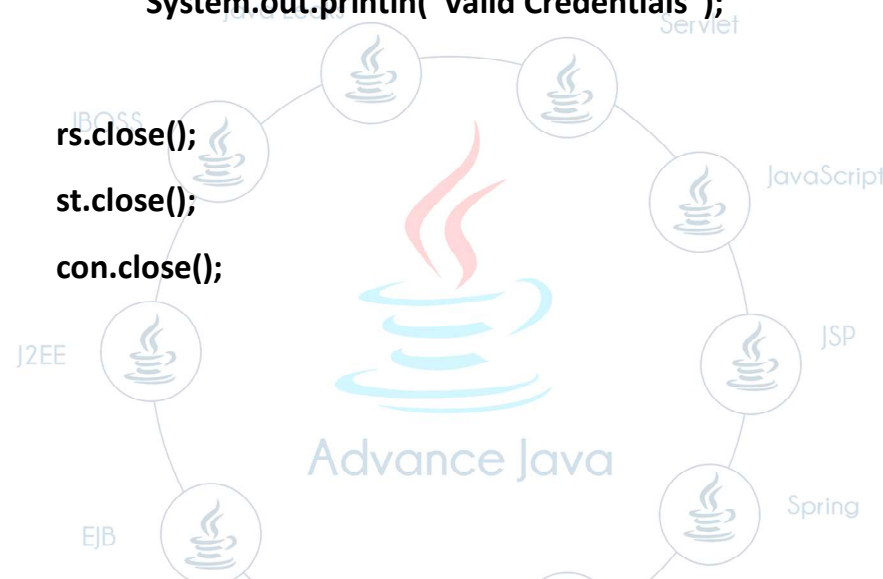



```

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

- It is not suitable to execute same query for multiple times with same values or different values.
- Framing query with variables is quite complex.
- We can't use string values directly to query without any conversion.
- It raises SQL injection problem.
- It does not allow us to insert date values in a database table column.
- It does not allow us to insert lob values in a database table column.
- To overcome these above limitations we need to use PreparedStatement object.

Pre-compiled SQL Query

- Our query goes to database software without input values and becomes parsed query either it is executed or not is called precompiled SQL query.
- PreparedStatement object deals with pre-compiled SQL query.

Working with PreparedStatement object

step1:

create a query with placeholder or parameters.

ex:

```
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 the pre-compiled SQL query.

ex:

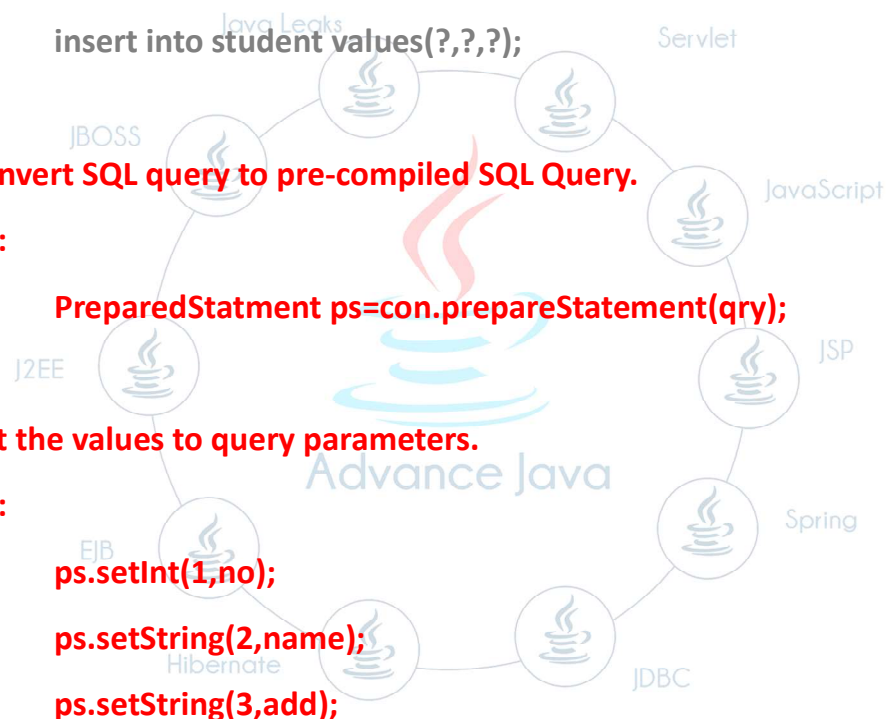
```
ps.executeUpdate();
```

step5:

Close PreparedStatement object.

ex:

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



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

```
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","s  
ystem","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
```

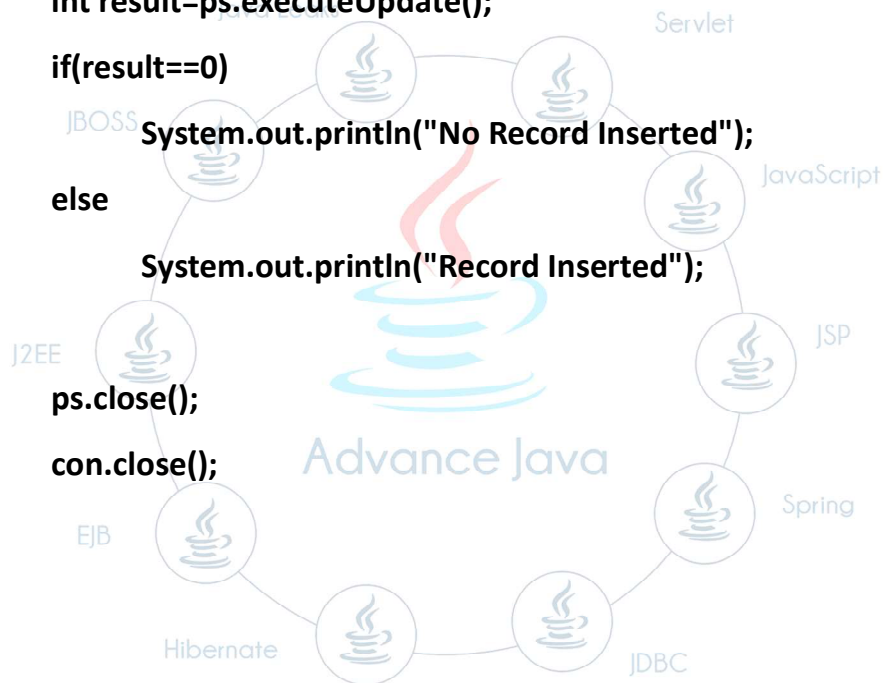
```
System.out.println("Record Inserted");
```

```
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","s
ystem","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 the student record 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 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 value
```

```
ps.setInt(1,no);
```

```
//execute
```

```
int result=ps.executeUpdate();
```

```
if(result==0)
```

```
System.out.println("No Record Deleted");
```

```
else
```

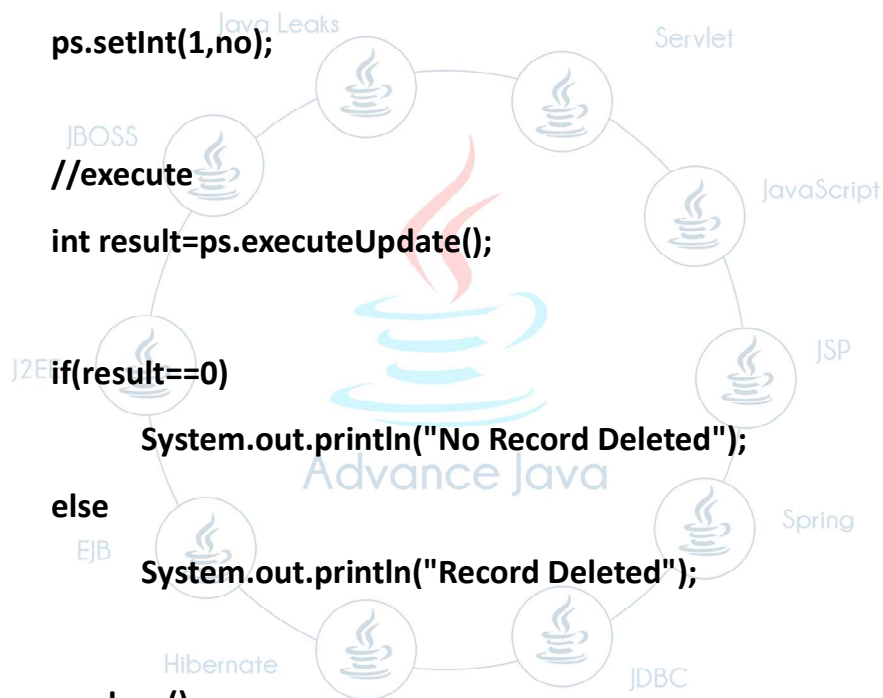
```
System.out.println("Record Deleted");
```

```
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 SolForSQLInjProb
```

```
{
```

```
    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
```

```
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 ");
```

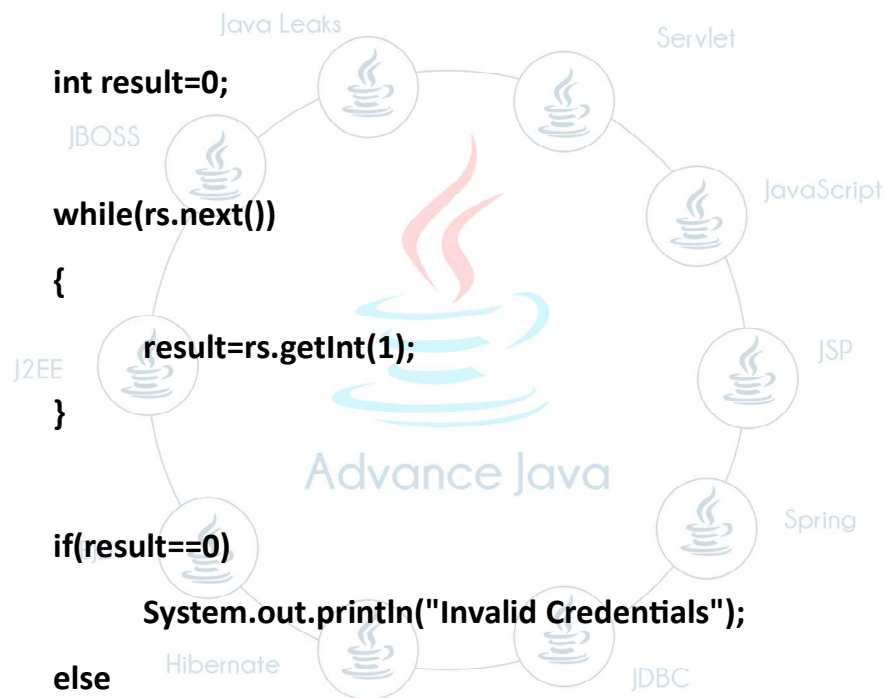
```
rs.close();
```

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

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

```
}
```

```
}
```



DatabaseMetaData

- A DatabaseMetaData is an interface which is present in java.sql package.
- A DatabaseMetaData provides metadata of a table.
- It gives information about database product name, database product version, driver name, driver versions, 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","s  
ystem","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());

    }
}

```

ResultSetMetaData

- A **ResultSetMetaData** is an interface which is present in **java.sql** package.
- A **ResultSetMetaData** provides metadata of a table.
- It gives information about number of columns, type of columns, datatype of a columns, size of a column 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","s
ystem","admin");

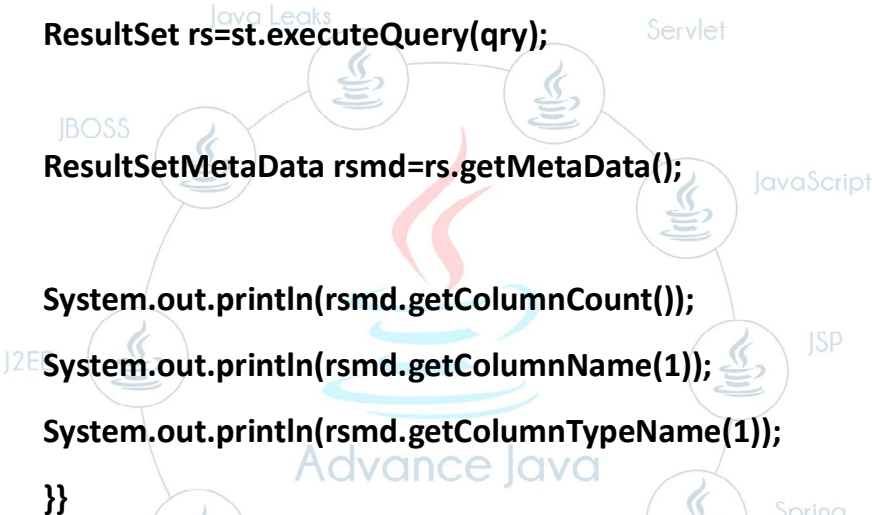
        Statement st=con.createStatement();

        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(1));
    }
}

```



Working with Date values

- While dealing with DOB,DOD,DOR,DOA and etc we need to insert or retrieve date values.
- It is never recommended to store date values in the form of string because it will not give proper comparison between two dates.

Every database software supports different date patterns.

ex:

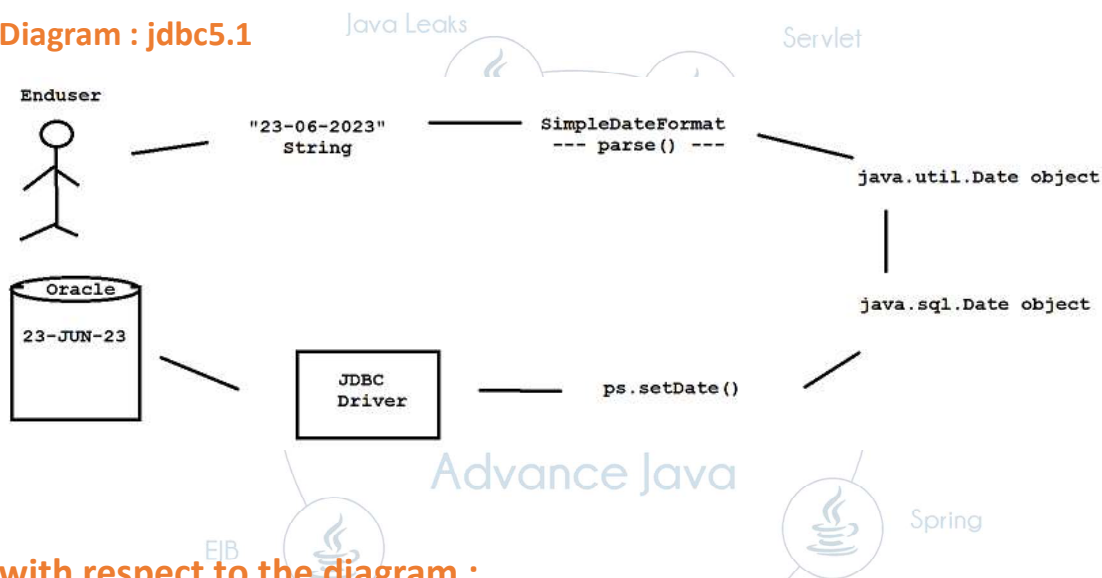
Oracle - dd-MMM-yy

MySQL - yyyy-MM-dd

- While working with simple Statement object we can't place date values directly to query parameter.
- To overcome this limitation we need to PreparedStatement object.
- A java.util.Date class is not suitable to perform database operations.
- A java.sql.Date class is suitable to perform database operations.
- Once jdbc driver gets the date value then it will insert in the pattern which is supported by underlying database software.

Standard procedure to insert Date values

Diagram : jdbc5.1



with respect to the diagram :

- Enduser will give date value in the form of string.
- A parse() method of java.text.SimpleDateFormat class converts string date to util date.
- Our application converts util Date class object to sql Date class object.
- A ps.setDate(-,-) method is used to set the date value to query parameter.
- Once the JDBC driver gets the date value then it will insert the pattern which is supported by underlying database software.

ex:

emp1 table

drop table emp1;

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

program

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","system","admin");

Statement st=con.createStatement();

String qry="select * from emp1";

ResultSet rs=st.executeQuery(qry);

while(rs.next())

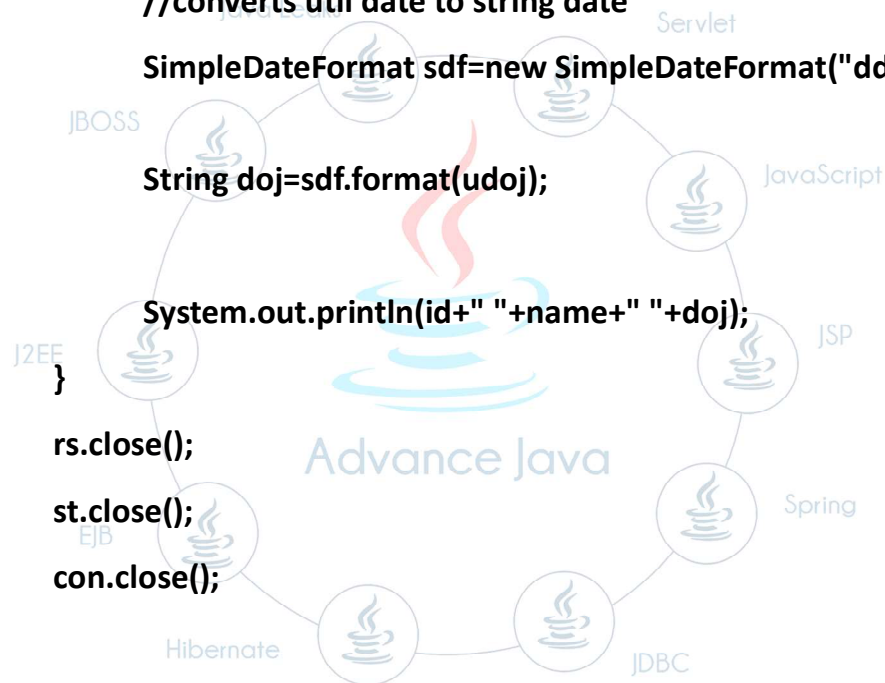
```

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

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

    //converts util date to string date
    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, audio, video, avi file and etc.

2)CLOB (Character Large Object)

ex:

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

- While dealing with matrimonial applications , job portal applications, profile management applications we need to insert and retrieve LOB values.
- Using simple Statment object we can't insert LOB values to query parametes.
- To overcome this limitation we need to use PreparedStatement object.

We can set LOB values to query parameters as follow.

ex:

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

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


ex:

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","s  
ystem","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");
```

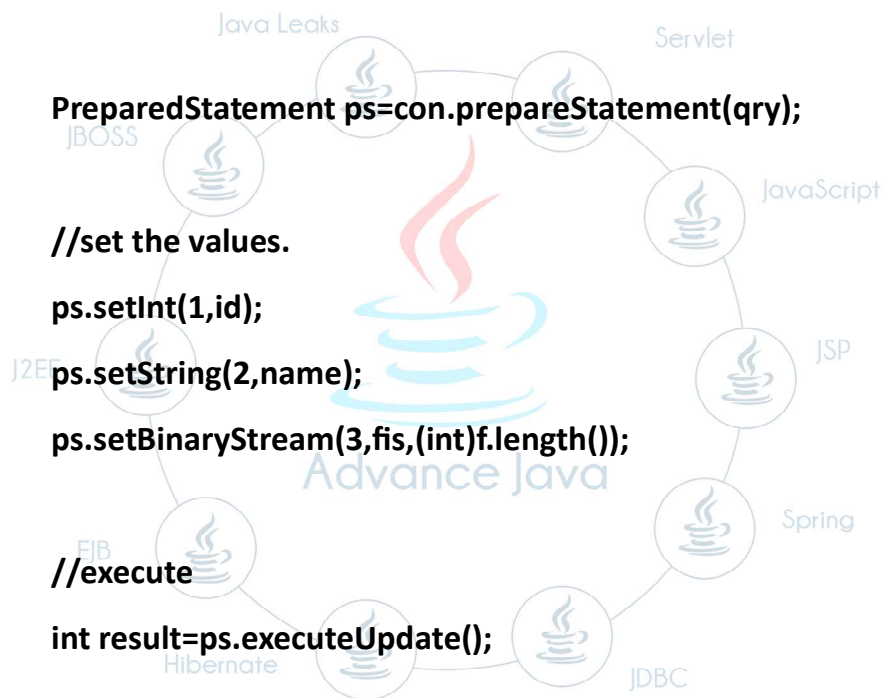
```
else
```

```
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","s  
ystem","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-
018\\JDBC\\sandeep.png");

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

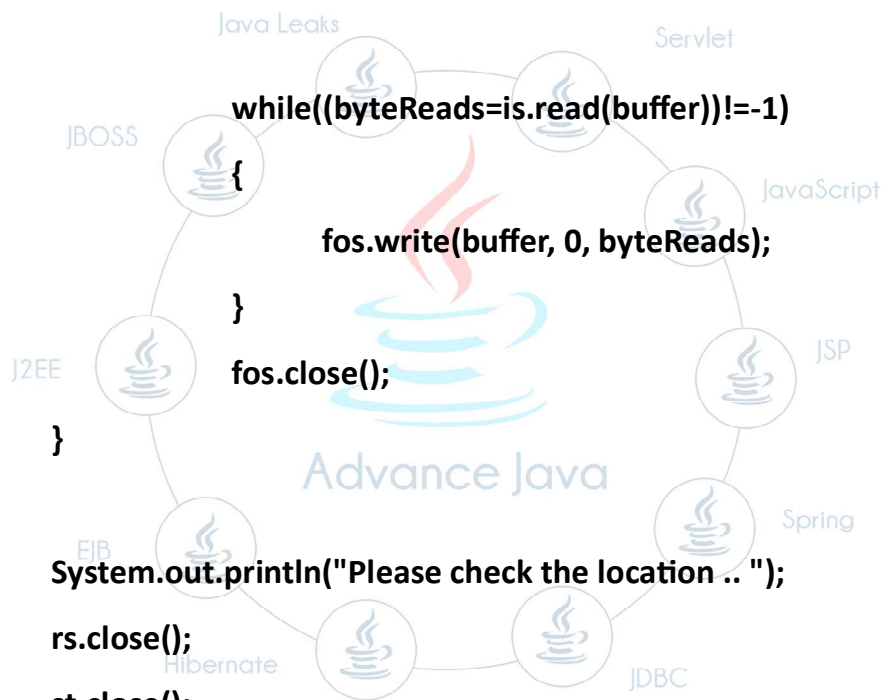
    while((byteReads=is.read(buffer))!=-1)
    {
        fos.write(buffer, 0, byteReads);
    }
    fos.close();

    System.out.println("Please check the location .. ");
    rs.close();
    st.close();
    con.close();

}

}

```



JDBC Flexible Application

- In jdbc , Connection object consider as heavy weight object.
- It is never recommended to create Connection object in every jdbc application.
- It is always recommended to create a separate class which returns JDBC Connection object.

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");
```

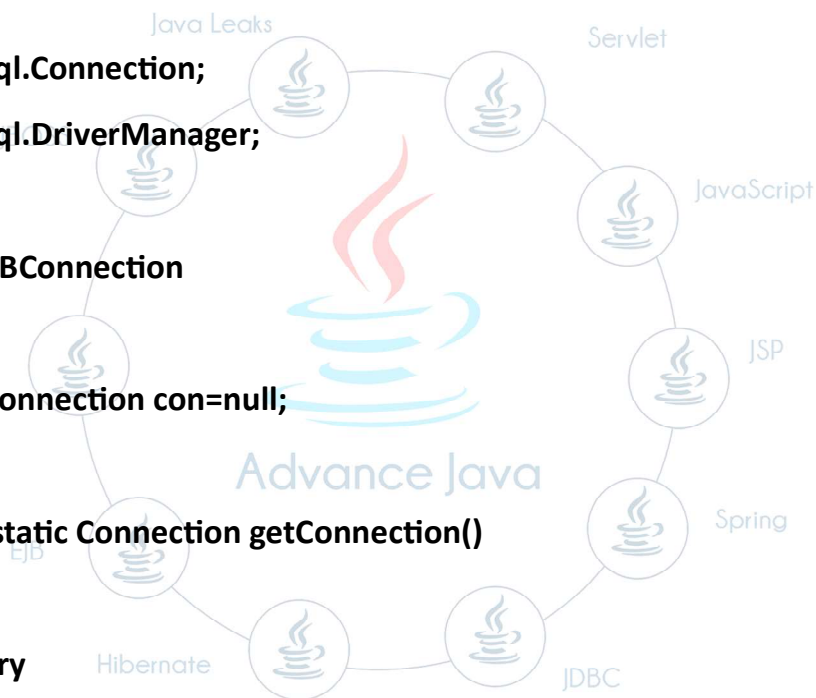
```
            con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","system","admin");
```

```
        }
```

```
        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";
```

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

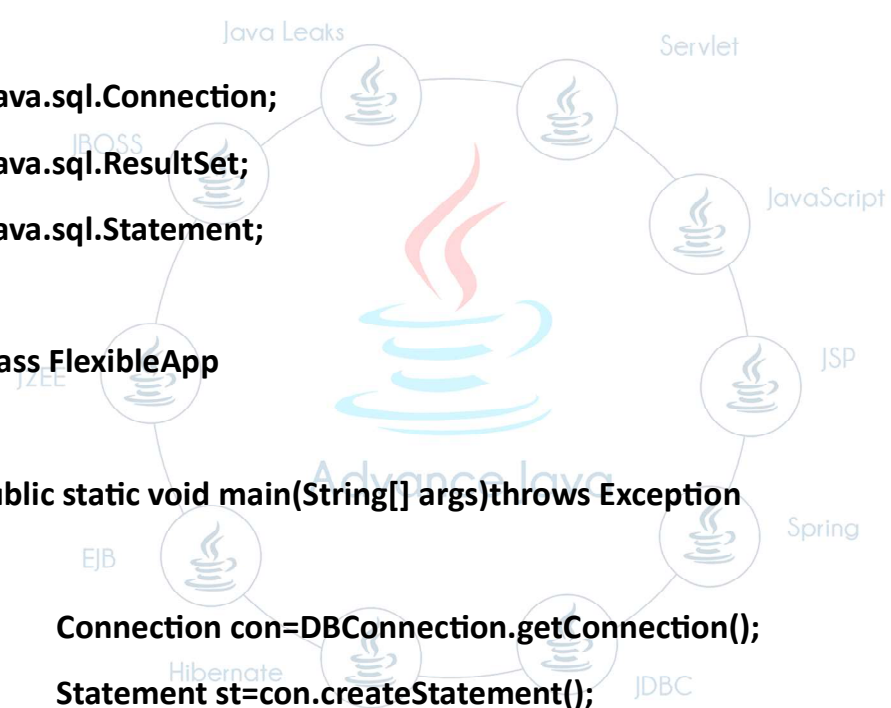
```
        while(rs.next())
```

```
        {
```

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

```
        }
```

```
        rs.close();
```



```

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

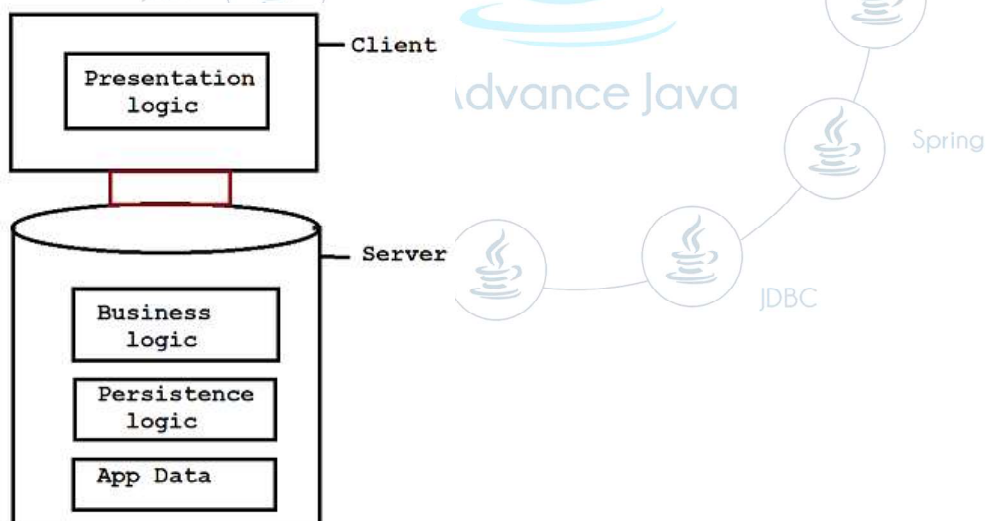
    }
}

```

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 database software in the form pl/sql procedures and functions.
- To deal with PL/SQL procedures and functions we need to use CallableStatement object.

Diagram: jdbc6.1



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

{

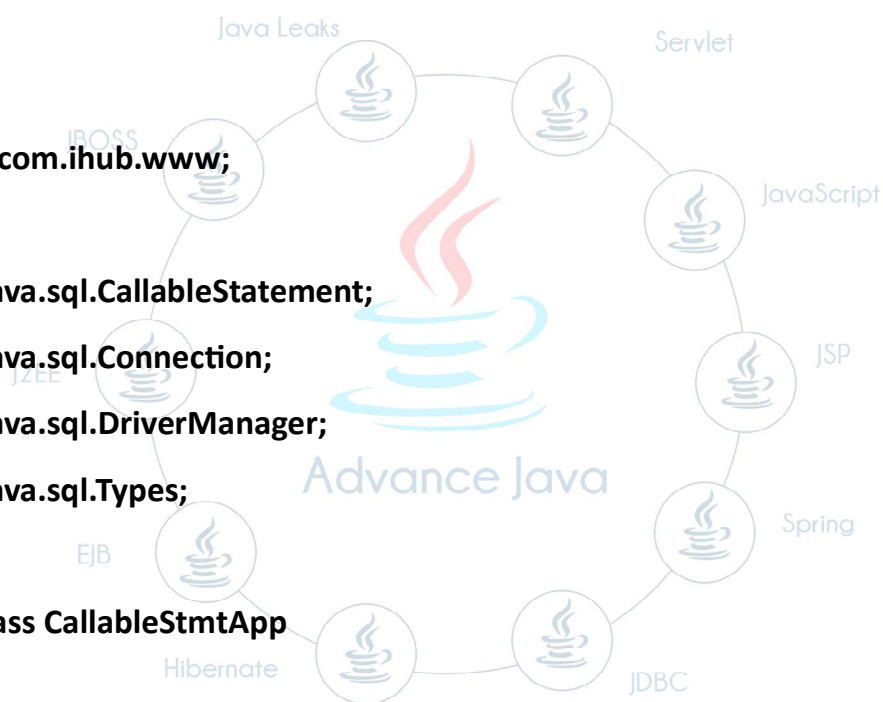
public static void main(String[] args)throws Exception

{

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 values to In parameter
```

```
cst.setInt(1, 10);
```

```
cst.setInt(2, 20);
```

```
//execute
```

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

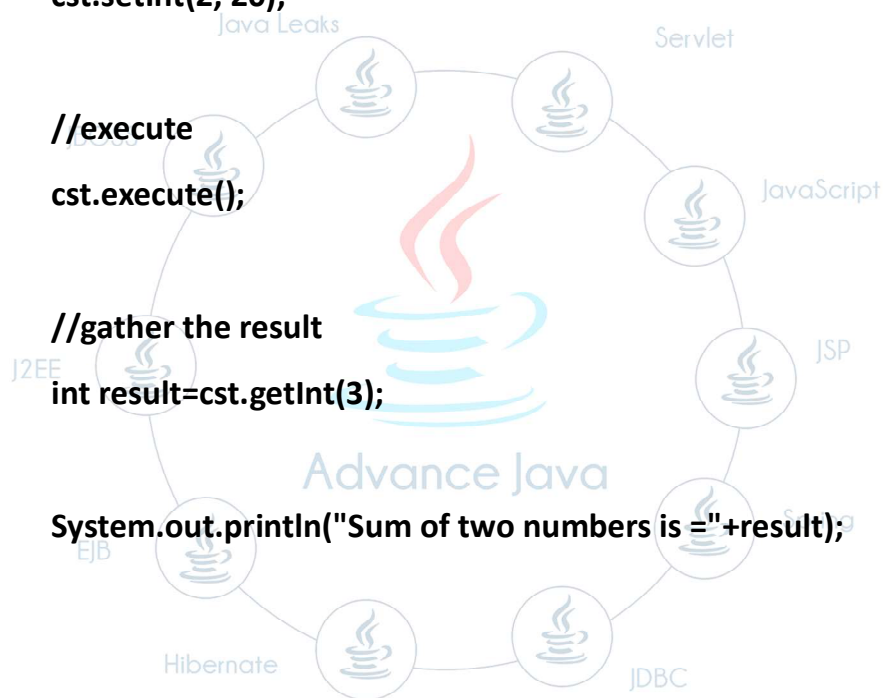
```
//gather the result
```

```
int result=cst.getInt(3);
```

```
System.out.println("Sum of two numbers is "+result);
```

```
}
```

```
}
```



Types of ResultSet objects

We have two types of ResultSet objects.

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, 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 ResultSet object is called 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, bi-directionally or randomly is called scrollable ResultSet object.
- If JDBC Statement object is created with type,mode value then 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 type values

ex:

`ResultSet.TYPE_SCROLL_SENSITIVE`

`ResultSet.TYPE_SCROLL_INSENSITIVE`

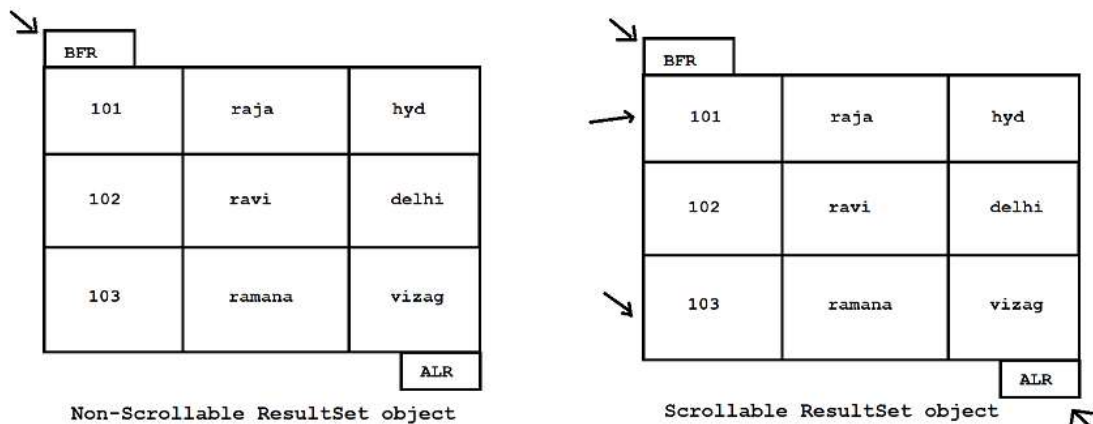
We have following mode values

ex:

`ResultSet.CONCUR_UPDATABLE`

`ResultSet.CONCUR_READ_ONLY`

Diagram: jdbc6.2



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","s  
ystem","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));
```

```
        }
```

```
        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());//true

```

```

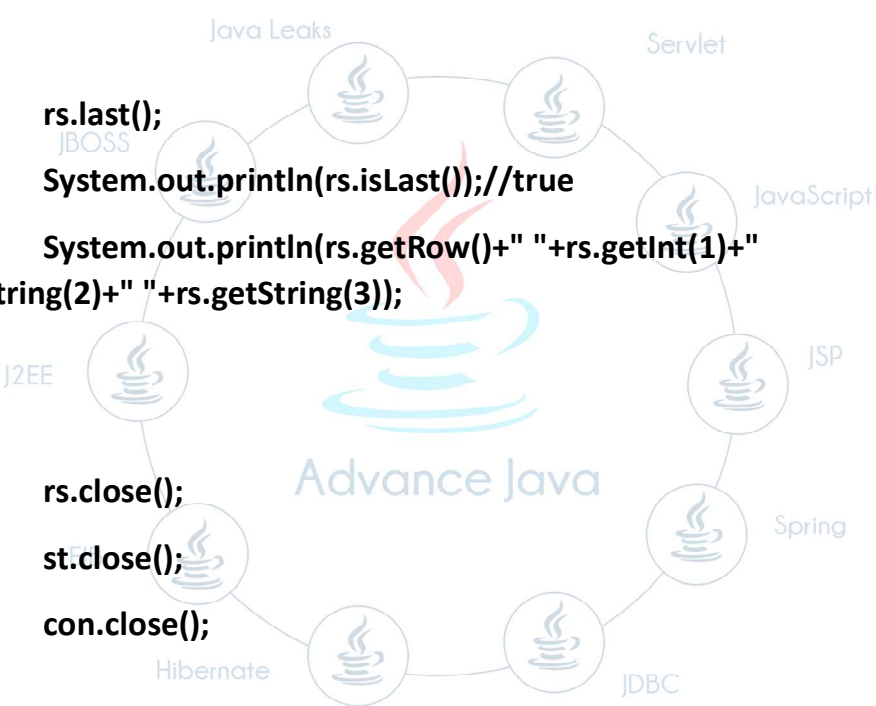
        System.out.println(rs.getRow()+" "+rs.getInt(1)+"
"+rs.getString(2)+" "+rs.getString(3));

```

```

rs.last();
System.out.println(rs.isLast());//true
System.out.println(rs.getRow()+" "+rs.getInt(1)+"
"+rs.getString(2)+" "+rs.getString(3));
rs.close();
st.close();
con.close();
    }
}

```



**“We are what our thoughts have made us,
so take care about what you think.”**

-Swami Vivekananda-

Batch Processing

- It is used to declare multiple queries in a batch and makes a single call to database.
- To add the queries in a batch we need to use `addBatch()` method of `Statement` object.
- To execute the batch we need to execute `executeBatch()` method of `Statement` object.

ex:

```
package com.ihub.www;  
  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.ResultSet;  
import java.sql.Statement;  
  
public class BatchProcessing  
{  
  
    public static void main(String[] args)throws Exception  
    {  
  
        Class.forName("oracle.jdbc.driver.OracleDriver");  
  
        Connection  
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","s  
ystem","admin");  
  
        Statement st=con.createStatement();  
  
        String qry1="insert into student values(105,'ramulu','pune')";
```

```
String qry2="update student set sname='rani' where sno=101";
```

```
String qry3="delete from student where sno=103";
```

```
//add query to batch
```

```
st.addBatch(qry1);
```

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

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

```
//execute the batch
```

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

```
int result=0;
```

```
for(int i:arr)
```

```
{  
    result+=i;  
}
```

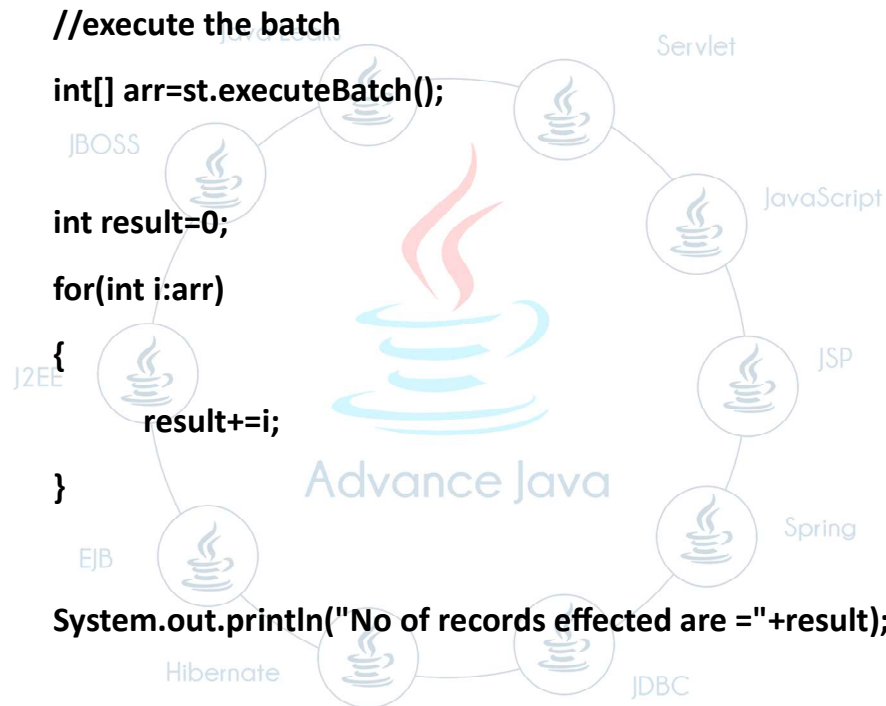
```
System.out.println("No of records effected are =" +result);
```

```
st.close();
```

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

```
}
```

```
}
```



**"Don't fear the darkness of the night,
wait for the dawn, one day your day will shine."**

-Sindhutai Sapkal-

Transaction Management

sbi table

```
drop table sbi;  
create table sbi(accno number(6),accholder varchar2(10),accbal number(10));  
insert into sbi values(111111,'rahul',80000);  
insert into sbi values(222222,'gopi',90000);  
commit;
```

kotak table

```
drop table kotak;  
create table kotak(accno number(6),accholder varchar2(10),accbal  
number(10));  
insert into kotak values(100001,'pavan',20000);  
insert into kotak values(100002,'srikanth',15000);  
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 sacno=sc.nextInt();

        System.out.println("Enter the destination account no :");
        int dacno=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","s
ystem","admin");

        con.setAutoCommit(false);

        Statement st=con.createStatement();

        //declare the queries

```

```
String qry1="update sbi set accbal=accbal-"+amt+" where  
accno="+sacno;
```

```
String qry2="update kotak set accbal=accbal-"+amt+" where  
accno="+dacno;
```

```
//add the queries to batch
```

```
st.addBatch(qry1);
```

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

```
//execute the batch
```

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

```
boolean flag=true;
```

```
for(int i:arr)
```

```
J2EE{
```

```
if(i==0)
```

```
{
```

```
EJB
```

```
flag=false;
```

```
break;
```

```
}  
Hibernate
```

```
}
```

```
if(flag)
```

```
{
```

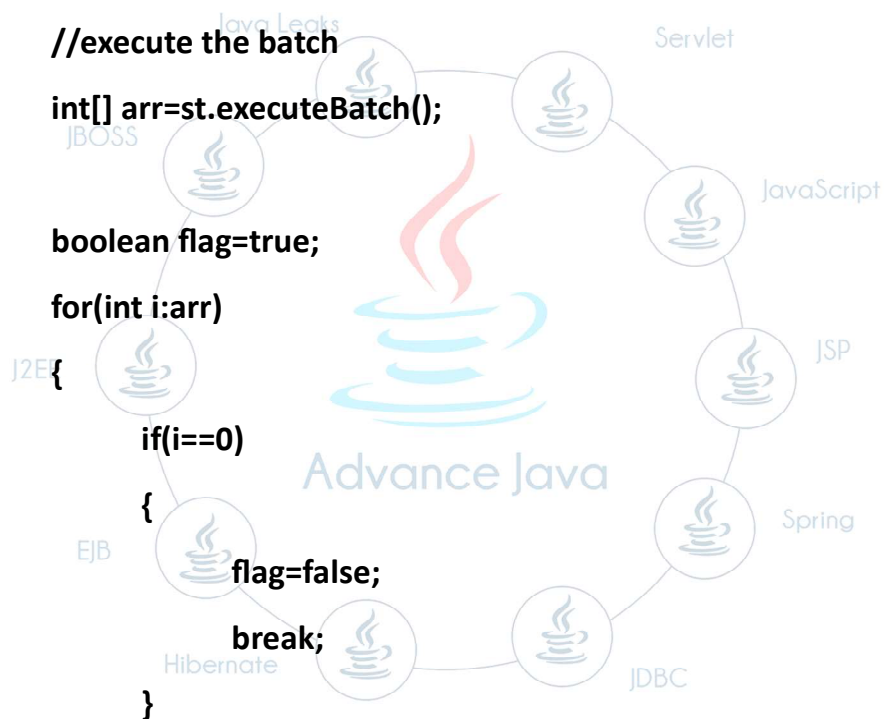
```
System.out.println("Transaction Done Successfully!!!");
```

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

```
}
```

```
else
```

```
{
```



```
        System.out.println("Transaction Failed ");
        con.rollback();
    }
    st.close();
    con.close();
}
}
```

Steps to interact with MYSQL Database

step1:

Download and Install MY/SQL Database successfully.

ex:

https://drive.google.com/file/d/1QQjWTJ9v8xz0nfuSGva1_QQwO6KDf9_c/view?usp=sharing

step2:

Connect with mysql by using password.

ex:

username : root(default)

password: root

step3:

create a SCHEMA in MYSQL.

ex:

create schema IH_JAVA_018

step4:

To check list of databases /schemas present in mysql db.

ex:

show databases;

step5:

Use IH_JAVA_018 scheme/database.

ex:

```
use IH_JAVA_018;
```

“The deeper the truth in a creative work, the longer it will live.”

-Charlie Chaplin-

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.ihub.www;
```

```
import java.sql.Connection;
```

```
import java.sql.DriverManager;
```

```
import java.sql.ResultSet;
```

```
import java.sql.Statement;
```

```
public class MySQLApp
```

```
{
```

```
    final static String DRIVER="com.mysql.jdbc.Driver";
```

```
    final static String URL="jdbc:mysql://localhost:3306/IH_JAVA_018";
```

```
    final static String USERNAME="root";
```

```
    final static String PASSWORD="root";
```

```
    final static String QUERY="select * from student";
```

```
    public static void main(String[] args)
```

```
    {
```

```
        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);
```

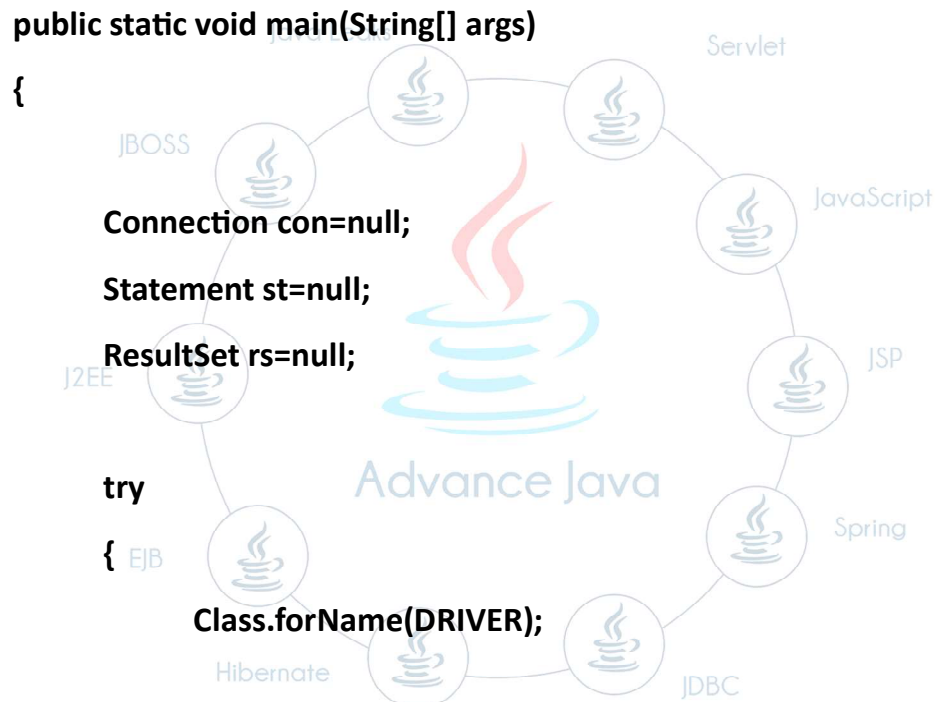
```
            while(rs.next())
```

```
            {
```

```
                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 :

<http://www.java2s.com/Code/Jar/m/Downloadmysqlconnectorjar.htm>

Note:

ojdbc14.jar -- for oracle

mysql-connector.jar --> for mysql

step9:

Run the jdbc application.

**"If you're really truthful with yourself,
it's a wonderful guidance".**

-Charlie Chaplin-