JDBC

Whenever the java application in running mode, we can access the different type of data in different memory areas of JVM.

For example static data, non-static or instance data and objects and local data are available in method area, heap area and java stack area respectively.

Once the java program execution completed all the data which is available in different memory location will be destroy.

So if we want to store the data permanently, then we should go for persistence stores like File Management Systems (Flat-File Database), Relational Data Base Management Systems.

In Java Standard Edition we have one library like java.io, by using this library we can able to make a communication with files and doing some operations like writing data into file and reading the data from files.

File system data management (or flat-file databases) served as the only method of file storage and retrieval before the advent of database management systems (such as relational databases). While retaining some use, flat-file databases suffer from poor accessibility, data redundancy, lack of standard file access and the inability to organize data.

But we have lot of drawbacks in this procedure. Those are listed below.

A. Data Redundancy (or) Duplicate data:
The same type of data can be exists in different files.

B. Limited User Access:
Only one end user can access the data from file, the content of the file is not sharable between all the end users at a time or simultaneously.

C. Data Mapping and Access:
We are unable providing mapping between content between one file information to other file information.

D. **Data format and dependency:**

Based on the structure of data, which is existed in the files, we need to access it.

This concept we can understand through Buffered Reader and String Tokenizer classes.

E. **Security:**

Every file is having some password protection.

If one file having multiple information I want to provide the permissions on one particular info not on all info.

That type of flexibility not available in files.

F. **Unable to store huge amount of data.**

G. **No Sequel Query Language support.**

H. **Merging one file to another file is difficult.**

I. **No constraints concept unique key, null key etc…**

The drawbacks which we have related to file management system, we can avoids through Relational Data Base Management System.

To communicate with different types of RDBMS we required one specification.

That specification is given by the java in the form of JDBC.

Introduction to JDBC:

It is a Trademark.

We don't have any official name like Java Data Base Connectivity.

Persistence Store:

The area or place which is used to store the data permanently is called persistence store.

Persistence:

The procedure to store the data in persistence store permanently

Persistence Logic:

The logic which is used to store the data permanently is called Persistence logic.

Persistence Data:

The Data which is stored permanently is called persistence data.

Persistence Operations:

The operations, which are going to do on persistence data which is available in persistence store by using persistence logic through persistence technologies are called persistence operations.

Persistence Technologies or specifications or frameworks:

The specifications, which are used to develop persistence logic for doing persistence operations on persistence data which is available in persistence store is called persistence specifications.

Ex: JDBC and Hibernate.

What is JDBC?

JDBC is specification API.

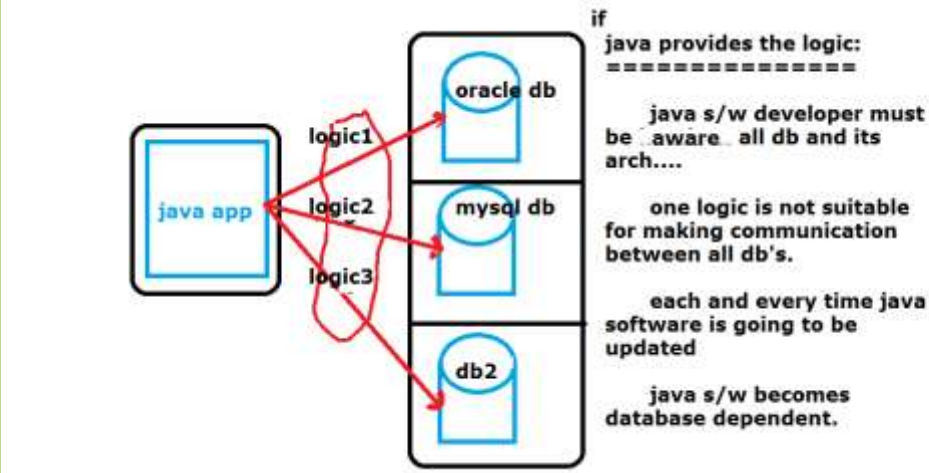
JDBC is a standard.

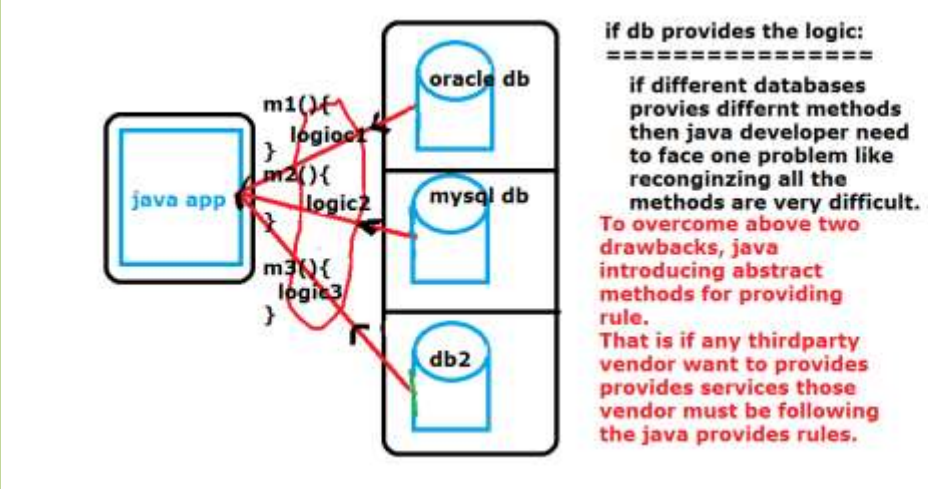
JDBC is a technology.

It is provides some set of rules and guidelines to communicate with persistence area's like relational database management system.

With the help of JDBC we can store and retrieving the data from RDBMS.

Ex: oracle, mysql, db2, sqlserver etc....





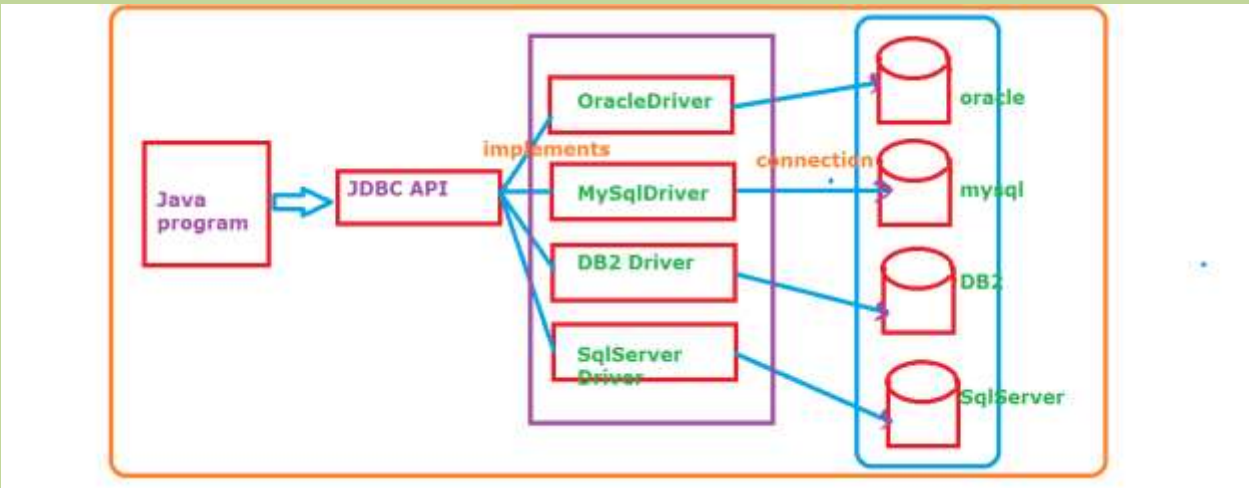
JDBC provides some abstraction layer in the form of interface and methods from java program to communicate with DataBase.

JDBC is not existed in java software programmer must be write the different code for connecting different database connections.

Here code may be like this, communicate with jdbc methods to C-language methods.

Note: Database software's are developing on top of C-language.

What is JDBC and Driver Software?



JDBC driver is software it is used to convert java calls or syntax to db calls or syntax to communicate with db for curd operations.

JDBC Driver is a JDBC specification implementing software, it provides the logic for interface methods which are given by the JDBC API.

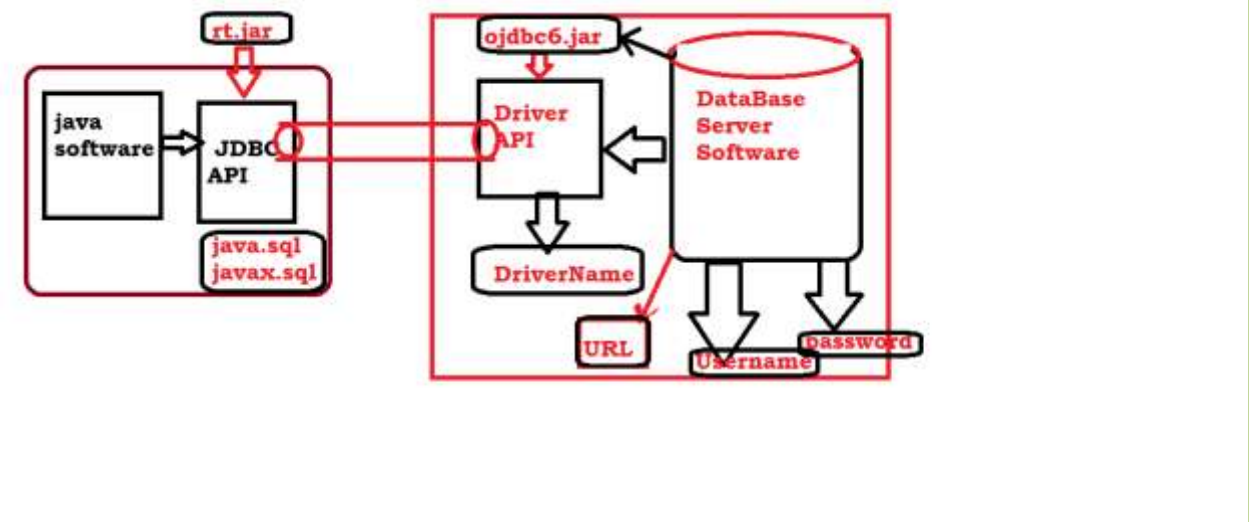
In these methods JDBC drivers software vendors writing the logic like calling database related functions to connect to db.

Role JDBC Driver software:

It is translator software, which is used to convert java calls to DB calls and establish the connection between java program and database.

Steps to connection with Data Base:

1. **Installation of Java Software.**
2. **JDBC API.(java.sql and javax.sql)**
3. **Installation of Database server software.**
4. **Driver API. (Driver interface implementation class details, nothing but Driver implementation software).**
   Ex: JdbcDriver, MysqlDriver
5. **Data Base URL.**
6. **Data Base Username.**
7. **Data Base Password.**
8. **Driver Name.**



Programmer no need to write connection logic to communicating with Data Base, this logic is already provided by Driver implementation classes.

Ex: oracle.jdbc.driver.OracleDriver

DB URL is used to make communication with specific database, this URL contains Driver Name, Data Base name, IP Address of Data Base installation system, port number, global service name.

Ex: jdbc:oracle:thin:@localhost:1521:XE

Data Base Username will provide data base username for provides security.

Data Base Password will provide data base password for provides security.

Prerequisites to develop JDBC Program:

To develop JDBC related programs we require two types of software's.

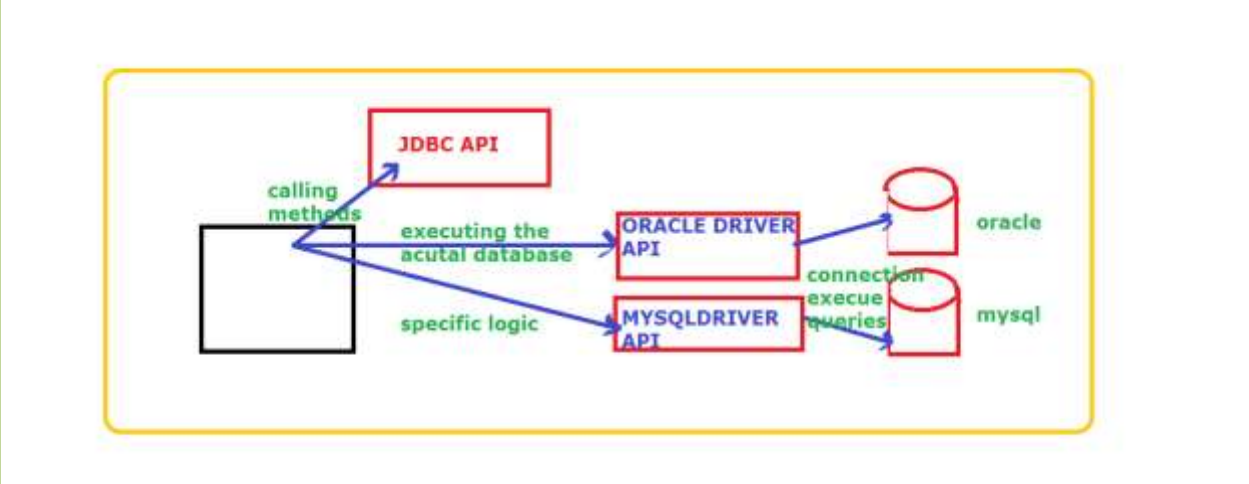
A. Java Standard Edition Development Kit (JDK).
B. Data Base software.

To develop JDBC related program, we require two types of API's.

1. **JDBC API.**
2. **Driver software API. (JDBC Technology Based Software).**

JDBC API is used to call the methods and making a communications and execute the queries.

Driver API is used to call actual logic. This logic is different from one database to other.



By seeing the above diagram, we came to one following conclusion.

That is java programmer calling methods by using JDBC API in the form of interface methods and calling actual logic from driver software API.

Interface always provides commonality to all implementation classes.

Calling interface methods by using interface reference variables and executing actual logic form their implementation class is part of object oriented programming principle, that runtime polymorphism.

If we follow the runtime polymorphism, we can able to communicate with different databases by using single java program.

For example if we want to communicate with oracle, we need to give oracle database related information, as well as mysql database related information.

So one db to other, only database related input data will be changes but not methods.

JDBC API is coming to our machine in the form of rt.jar file which is located in our java software installation directory. That is java\jre\lib\rt.jar file.

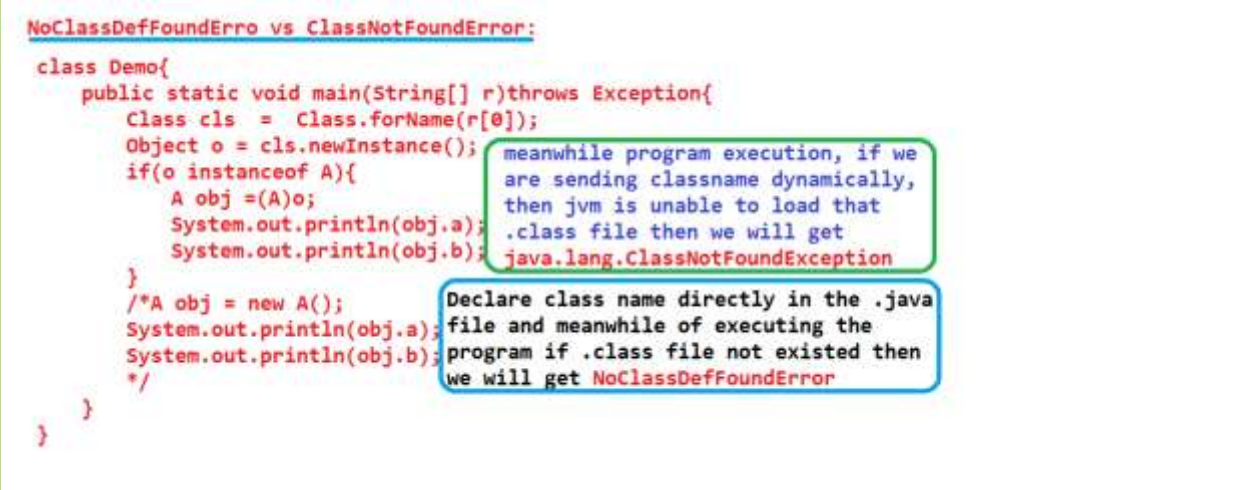
Driver software related API is coming to our machine in the form of .jar or zip file.

Data Base	Driver Name
Oracle 8i	classes12.jar
Oracle 9i	classes12.jar and ojdbc14.jar
Oracle 10g	ojdbc14.jar and ojdbc5.jar
Oracle 11g	ojdbc5.jar and ojdbc6.jar
Oracle 10g XE	ojdbc14.jar
Oracle 11g XE	ojdbc6.jar

Ojdbc jar file is available in our data base installation directory that is

C:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar

```
new operator  vs  newInstance():
-------------------------------
new operator is used to create an object meanwhile developing program
for a particular that means in the static manner.
    Student s = new Student();
newInstance() is used to create an object dynamically.
That means meanwhile of developing program if we dont know about
particular class name then we should go for newInstance().
  class Demo{
     public static void main(String[] r)throws Exception{
         //A obj = new A();
         Class cls = Class.forName(r[0]);Object o = cls.newInstance();
     if(o instanceof A){A obj = (A)o;System.out.println( " A class");
           System.out.println(obj.a);System.out.println(obj.b);
         }else{B obj = (B)o;System.out.println( " B class");
           System.out.println(obj.c);    System.out.println(obj.d);
         }}
```



ODBC:

Stands for Open Database connectivity.

It is used to interact with database.

It has been developed on top of "C" language specific methods.

If we want to communicate with ODBC we need to execute native methods in our applications.

If we are using native methods implementation in our java application we can lose java features, like platform independency.

If the bug or error are raises in the native methods, there may be chance of destroy the JVM.

To overcome this problem Java community people introduce one technology or specification in the form of JDBC.

Understanding JDBC API:

JDBC API is used for accessing the JDBC Driver software, which can be drive (carry) our queries to data base and execute them in the data base server and finally that outputs will be carry to end user or java application, for this we need to establish the SESSION.

How can we develop the Session between Java Client to Database software?

By using the JDBC Driver software, it is the implementation class of java.sql.Driver interface.

After that we need to call the connect method.

Java.sql.Connection connect (String url, Properties props) throws SQLException{}.

```
import java.sql.Connection;
import java.sql.Driver;
import java.util.Properties;

public class JDBCEx1 {

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

		Driver d = new oracle.jdbc.driver.OracleDriver();
		String url = "jdbc:oracle:thin:@localhost:1521:xe";
		Properties p = new Properties();
		p.setProperty("user", "system");
		p.setProperty("password", "manage");
		Connection con = d.connect(url, p);
		System.out.println(con);

		if(con !=null){

			System.out.println("connection established");

		}
		else
			System.out.println("not established");
}

}
```

In the above approach, we need to face one problem that is assume we are developing one project with 10 classes. All the classes require data base connection then, above statements we need to write in all the classes, then automatically number of Driver objects are increases in JVM heap memory, to avoiding this process JDBC introduce one helper class that is java.sql.DriverManager.

This is one predefine class, which is used to register the multiple drivers with help of one method that is registerDriver (driverobject d).

More Information about Factory Methods:

```
import java.util.Scanner;

interface Animal{
	void eat();
	void sleep();
}
class AnimalFactory{
	static Animal factoryAnimal(String animalName){
		if(animalName.equalsIgnoreCase("lion")){
			return new Lion();
		}else if(animalName.equalsIgnoreCase("rabit")){
			return new Rabit();
		}return null;

	}
}
class Lion implements Animal{
	@Override
	public void eat() {
		System.out.println("eats non-veg");
	}
	@Override
	public void sleep() {
		System.out.println("sleeping in caves");
	}
}
class Rabit implements Animal{
	@Override
	public void eat() {
		System.out.println("eats veg");
	}
	@Override
	public void sleep() {
		System.out.println("sleeping in bushes");
	}
}
```

```
class Tiger implements Animal{
	@Override
	public void eat() {
		System.out.println("tiger eats non-veg");
	}
	@Override
	public void sleep() {
		System.out.println("tiger sleeping in caves");
	}
}
public class FactoryMethod {
	public static void main(String[] args) {
		System.out.println("enter which animal object you
want");
		Scanner scan = new Scanner(System.in);
		String animalName = scan.next();
		Animal obj = AnimalFactory.factoryAnimal(animalName);
		//Animal obj = new Lion();
		obj.eat();
		obj.sleep();
	}
}
```

In the above approach we are facing NullPointerException.

We can resolve above problem with factory method.

```
interface Animal{

	void eat();

	void sleep();

}

class AnimalFactory{

	static Animal factoryAnimal(String animalName) throws
ClassNotFoundException,


InstantiationException, IllegalAccessException{

		/*if(animalName.equalsIgnoreCase("lion")){
```

```
                return new Lion();
            }else if(animalName.equalsIgnoreCase("rabit")){
                return new Rabit();
            }return null;*/
            Class cls  = Class.forName(animalName);
            Object obj = cls.newInstance();
            return (Animal)obj;
        }
}
class Lion implements Animal{
        @Override
        public void eat() {
            System.out.println("eats non-veg");
        }
        @Override
        public void sleep() {
            System.out.println("sleeping in caves");
        }
}
class Rabit implements Animal{
        @Override
        public void eat() {
            System.out.println("eats veg");
        }
```

```
        @Override
        public void sleep() {
                System.out.println("sleeping in bushes");
        }
}
class Tiger implements Animal{
        @Override
        public void eat() {
                System.out.println("tiger eats non-veg");
        }
        @Override
        public void sleep() {
                System.out.println("tiger sleeping in caves");
        }
}
```

Save with Test.java

And compile like javac Test.java

```
import java.util.Scanner;


public class MethodDemo {
                public static void main(String[] args) throws
ClassNotFoundException,
InstantiationException, IllegalAccessException {
        System.out.println("enter which animal object you want");
                        Scanner scan = new Scanner(System.in);
```

```
                String animalName = scan.next();
                Animal obj =
AnimalFactory.factoryAnimal(animalName);
                //Animal obj = new Lion();
                obj.eat();
                obj.sleep();
            }
}
```

Save above file with FactoyMethod.java

Compile and execute.

Note: Before executing first we need to compile Test.java file otherwise we will get ClassNotFoundException.

The Same factory method approach we can use for Driver software loading process.

There are lots of ways to loading Driver Software.

forName() will load .class file from secondary memory to primary memory.

newInstance() will create object for .class file (byte code).

forName() will throws one exception like java.lang.ClassNotFoundExcepiton if the .class file is not existed.

newInstance() will throws two exceptions like java.lang.IllegalAccessException, if there is no zero argument constructor declared as private and throws java.lang.InstantiationExcepitn if there is no zero argument constructor.

We have lot of approaches to load Driver class.

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

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

import oracle.jdbc.driver.OracleDriver;

public class Demo1 {

		public static void main(String[] args) throws
SQLException, ClassNotFoundException{
			//step 1
			DriverManager.registerDriver(new OracleDriver());
			Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");

			if(con != null){
				System.out.println("connection is
established");
				System.out.println("connection object: "+con);
			}
			else
				System.out.println("connection is not
established");


	}

}
```

DriverManager.getConnection(-,-,-) will create object for implementation class (T4CConection) of Connection and placed object into Connection reference.

Simply DriverManager.getConnection(-,-,-) will create connection object for communicating with db.

If we are not providing DriverName then we will get java.sql.SQLException: No suitable driver found for jdbc:oracle:thing:@localhost:1521:xe

We will find portnumber, global service-name and computer name in the following location.

C:\oraclexe\app\oracle\product\10.2.0\server\NETWORK\ADMIN\ tnsnames file

→ojdbc14.jar file we can find in the following location:

C:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar

The url may existed in the following manner.

1. **"jdbc:oracle:thin:@localhost:1521:xe"**
2. **"jdbc:oracle:thin:@Ramchandar-PC:1521:xe"**
3. **"jdbc:oracle:thin:@ipaddress:1521:xe"**

localhost→ current system

Lenovo-pc→represents computer name

Ipaddress→system identification number

Approach-2:

```
OracleDriver od = new OracleDriver();
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");

if(con != null){
System.out.println("connection is established");
      System.out.println("con: "+con);
}
else
System.out.println("connection is not established");
```

Approach:3:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import oracle.jdbc.driver.OracleDriver;
class Demo extends OracleDriver{
	Connection connection() throws SQLException{
		Connection con = DriverManager.getConnection(
				"jdbc:oracle:thin:@lenovo-PC:1521:xe",
				"system",
				"manager");
		return con;
	}
}
public class JdbcDemoProgram2 {
	public static void main(String[] args) throws SQLException{
		Demo d = new Demo();
		Connection con = d.connection();
		if(con != null){
			System.out.println("con: "+con);
			System.out.println("connection established");
		}
		else
			System.out.println("connection not established");
	}
}
```

Approach:4:

```
	Class.forName("oracle.jdbc.driver.OracleDriver");//fully qualified name mandatory
		Connection con = DriverManager.getConnection("jdbc:oracle:thin:@lenovo-PC:1521:xe","system","manager");

		if(con != null){
		System.out.println("connection is established");
			System.out.println("con: "+con);
		}
		else
		System.out.println("connection is not established");
```

Approach:5:

```
Connection con = DriverManager.getConnection("jdbc:oracle:thin:@lenovo-PC:1521:xe","system","manager");
```

```
                if(con != null){
                System.out.println("connection is established");
                        System.out.println("con: "+con);
                }
                else
                System.out.println("connection is not established");
```

Appraoch:6:

```
                Properties p = new Properties();
                p.put("user","system");
                p.put("password", "manager");
                Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe", p);

                if(con != null){
                System.out.println("connection is established");
                        System.out.println("con: "+con);
                }
                else
                System.out.println("connection is not established");
```

In the above program p.setProperty(-,-) key value must be "user", don't write key like username or User.

Approach:7:

```
        FileInputStream fis = new FileInputStream("db.properties");
                Properties p = new Properties();
                p.load(fis);
                String driver = p.getProperty("driver");
                String url = p.getProperty("url");
                String user= p.getProperty("username");
                String password = p.getProperty("password");
                Class.forName(driver);
Connection con = DriverManager.getConnection(url,user,password);

                if(con != null){
                System.out.println("connection is established");
                        System.out.println("con: "+con);
                }
                else
```

System.out.println("connection is not established");

In the above program if write the bellow code like

```
Connection con =
           DriverManager.getConnection(url,p);
```

we get one exception that is java.sql.SQLException
invalid agrument in call

the we want to write this getConnection(url,p) for connecting the database
we must be set the username and password explicitly to properties object (p) by using setProperty(-,-).

like bellow

```
           p.setProperty("user", user);
           p.setProperty("password", password);
           Class.forName(driver);
           Connection con =
           DriverManager.getConnection(url,p);
           System.out.println(con);
```

AutoLoading:

From JDBC 4.0 no need to specify any details related DriverName.

Automatically Driver is loading.

Driver Class details automatically loading from bellow path

ojdbc6.jar\META-INF\services

```
import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class Demo1 {

      public static void main(String[] args) throws SQLException,
ClassNotFoundException{
```

```
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
            if(con != null){
                  System.out.println("connection is established");
                        System.out.println("connection: "+con);
            }
            else
                  System.out.println("connection is not established");
      }
}
```

Possible Exceptions in bellow program:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class Demo1{
public static void main(String[] args) throws SQLException,
ClassNotFoundException{
Class.forName("oracle.jdbc.driver.OracleDrivers");
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
                  if(con != null){
                  System.out.println("connection is established");
                        System.out.println("connection: "+con);
```

```
                }
                else
                System.out.println("connection is not established");
        }
}
```

1. If given class name not existed or any spelling mistake while type the program we will face one exception that is

   Java.lang.ClassNotFoundException.

2. In above program, if we are giving any invalid global service name we will face exception like

Exception in thread "main" java.sql.SQLException: Listener refused the connection with the following error:

ORA-12505, TNS:listener does not currently know of SID given in connect descriptor.The Connection descriptor used by the client was: lenovo-PC:1521:oracl

3. In above program if we are doing mistake in port number,localhost then we will face bellow exception

Exception in thread "main" java.sql.SQLException: Io exception: The Network Adapter could not establish the connection

4. In above program if we are doing any mistake related username and password we may be faced problem like bellow.

Java.sql.SQLException Inavalid username/password.

5. It the given table is not existed then we will face bellow exception that is

Java.sql.SqlSyntaxErrorException Table or View does not exist.

6. If data is not displayed on the console may related data is not available and records are inserted or after inserting the records may be records are not commit.

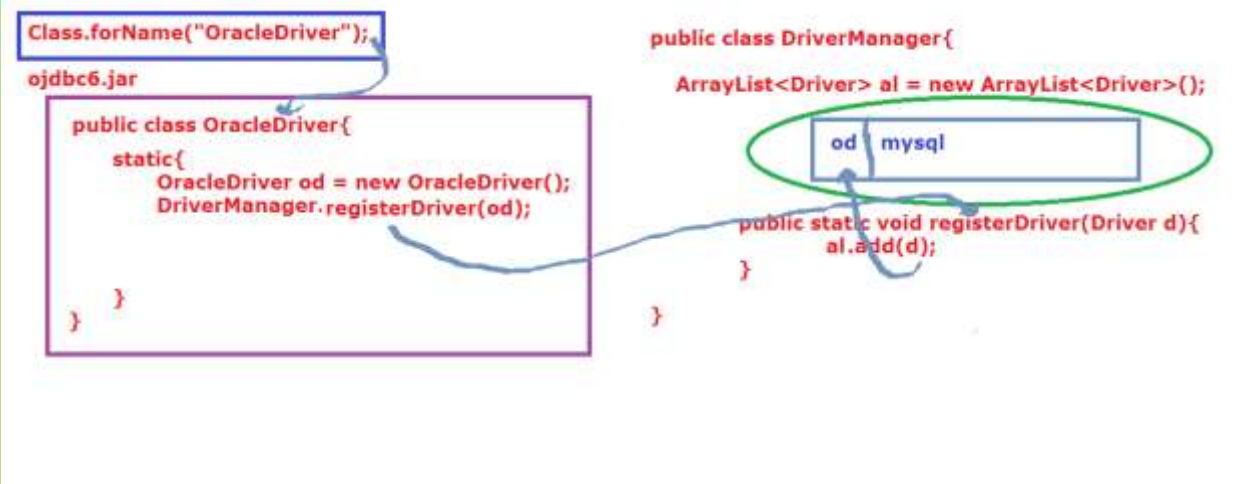
Explanation about First Program:

Class.forName(-) Method loads the OracleDriver class bytecode into JVM from ojdbc6.jar.

After that JVM executes static block from OracleDriver class. In this static block OracleDriver object is created later forward DriverManager class to register.

DriverManager establish the connection using this OracleDriver.

DriverManager class having one collection object to store or register different driver objects.



forName() is not register our driver object to DriverManager class.

OracleDriver class itself creates object and register with DriverManager object by calling registerDriver().

We can able to register to OracleDriver object to DriverManager, but no need already this will be done by the automatically.

Connection con = DriverManager.getConnection() ;

In the getConnection(), there is one more logic that is driver.connect() method is called for establish connection to the data base.

DriverManager is not create the connection it is just find out the url of given driver and handover these details to driver class. Driver classes itself create object and handover to DriverManager.

DriverManager will find out the suitable driver from registers drivers by using driver.acceptURL().

In the getConnection(), the DriverManager will fetch the drivers from collection object by calling acceptURL() by passing current url, if DriverManager get return value is true then it call connect() method.

Both acceptURL() and connect() are available in Driver implementation class like OracleDriver.

```
public class DriverManager{
    private static ArrayList<Driver> drivers = new ArrayList<Driver>();
    public static void registerDriver(Driver d){
        drivers.add(d);
    }
    public static Connection getConnection(String url,String userName,String password){
        for(int i=0;i<drivers.size();i++){
            Driver d = drivers.get(i);
            if(d.acceptURL(url)){
                Properties p = new Properties();
                p.setProperty("user",userName);
                p.setProperty("password",password);
                return d.connect(url,p);
            }

        }
        throw new java.sql.SQLException("No suitable driver found for "+url);
    }

}
```

DriverManager returns Connection interface implementation class object.

Statement stmt = con.createStatement();

Connection implementation class creates object for Statement interface implementation class object on loaded driver and finally handover to stmt variable.

Statement implementation class object created by the connection object because, to send and execute in the db, we need Connection object ID.

This connection id will copied from connection object to statement object meanwhile creating statement object.

Whenever sending the query to db, first statement object append that connection id to query.

Whenever query is handover to db, then db will reads the connection id and then finds its appropriate buffer and later execute the query and stores or gets result from main table according to that query.

If connection id related buffer not found or available, then simply database throw an exception that is connection closed.

If we are calling stmt.executeQuery() after connection closing (con.close()) , then we will face one exception like below.

That is java.sql.SQLException: Closed Connection.

ResultSet rs = st.executeQuery(-);

With the help of above line the given query is send and executed at db side by using appropriate buffer.

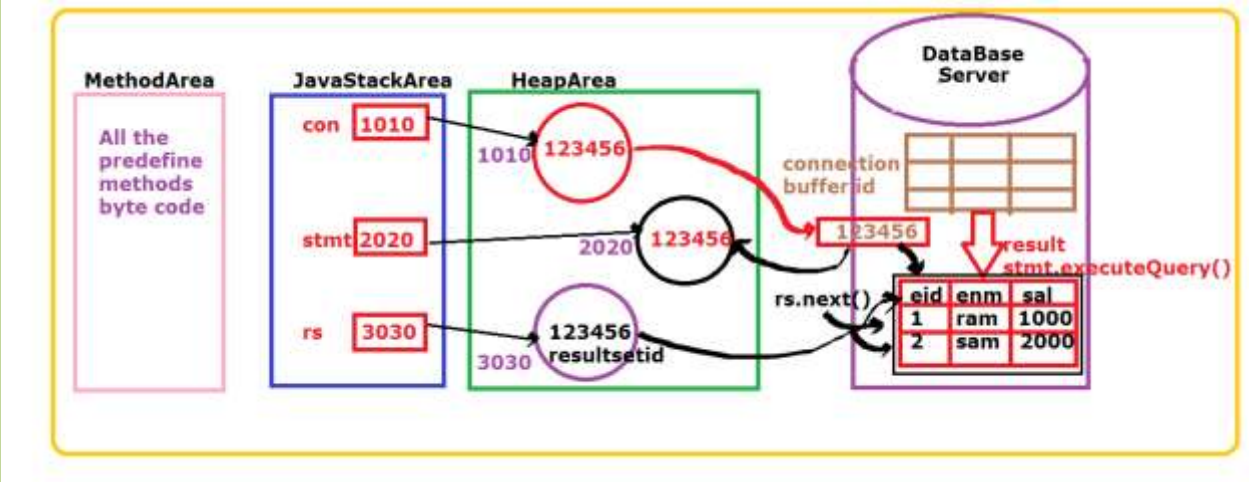
The results are stored in at database end in appropriate connection id buffer, and it reference id is given back to java app in the form ResultSet implementation class object and that will be stored in rs variable.

ResultSet is a cursor, it wills always pointing to non-record are that is before first record stored location.

If we want read the data from db we need to use rs.next() first, this will help us to move the cursor from non-record area to record area.

If the record is available that record will given back to java side and we should read that data by using rs.get() by passing index position or column name.

Once all records reading finally rs.next() will return false, then our reading operation will be closed.



API and API Documentation:

API means collection of classes and interfaces, these reusable components.

These are used to develop new classes, we can also called these new classes like user define classes.

Ex: String, Thread, Runnable, Connection, Driver etc...

API Documentation is theoretical information about API class and interfaces.

Like how to use the class, member variables and methods, their syntax....

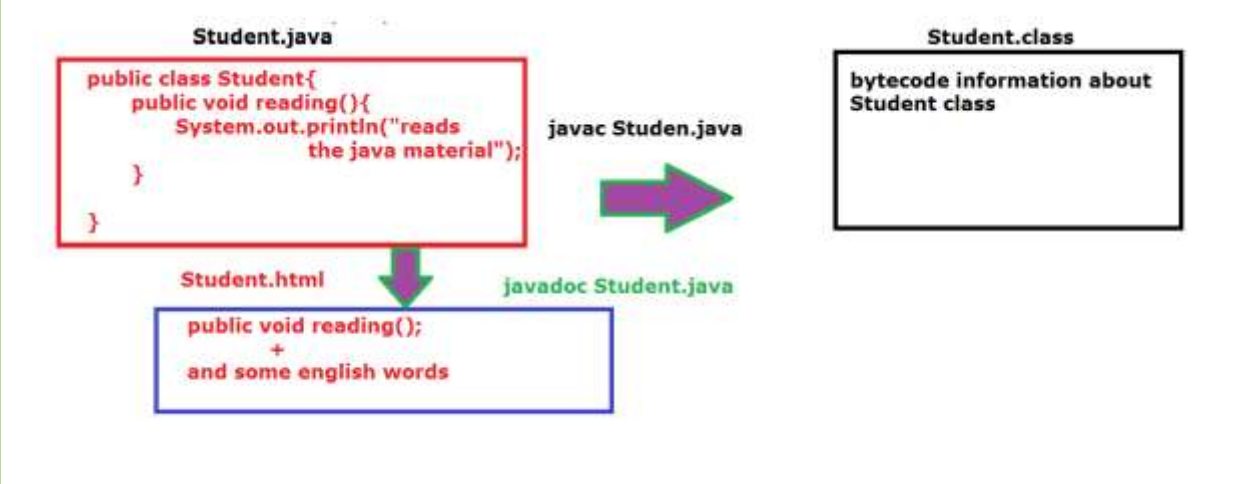
Java developer can develop their own classes by reading documentation and using API class and interfaces.

API classes and interfaces byte code are using by the compiler and JVM.

Every java class can be developing by using three extensions.

1. **.java (source code)**
2. **.class (byte code)**
3. **.html (theoretical information)**

.html file we can develop by using one command that is "javadoc", which is available in java\jdk\bin folder.



Java provides API documentation for both JSE and JEE.

ResultSetMeataData:

It is a JDBC object.

It is used retrieve the column information from given query.

Column information is changing from one query to another.

It will not give original table number of column information.

Select * from emp;

It will give all column information.

Select eno,ename from emp;

In the above RSMD will give only eno, ename column information.

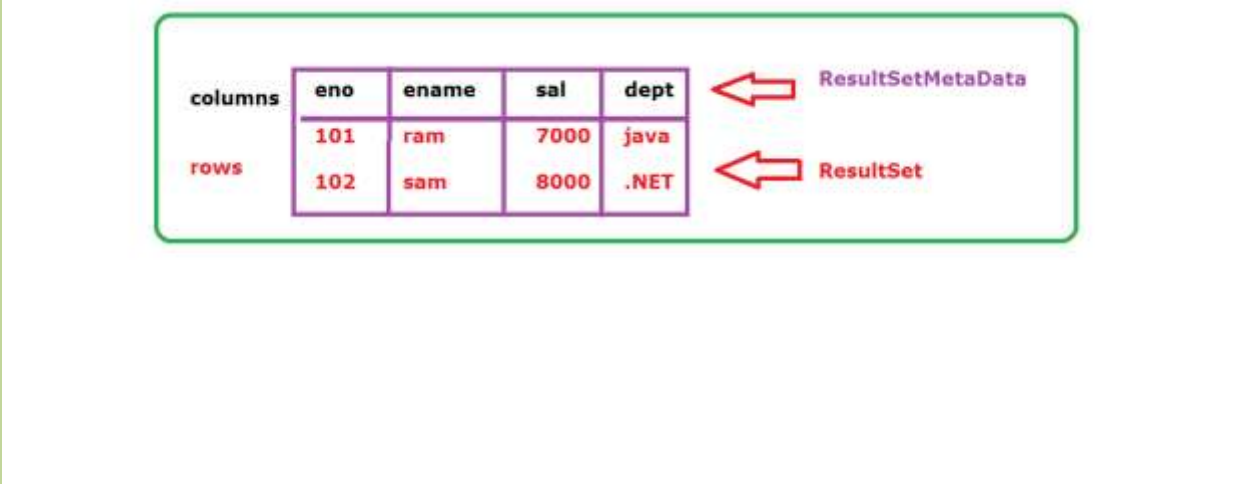
Difference between ResultSet and ResultSetMetaData:

ResultSet is mainly design for reading the data from the row actual data whereas RSMD will provide column information, which is quires from RS.

ResultSet contains some methods to read and move the cursor from one row to another row.

RSMD is also contains some methods column information like column name, column type, precision and scale and number of columns.

If we want to work with RSMD method, first we need to retrieve the RSMD object.



To obtain RSMD object, first we need obtain ResultSet object, in that object we have one factory method like getMetaData().

public ResultSetMetaData getMetaData() throws SQLException{

}

Syntax in java program:

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

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
public class Demo1 {
```

```
            public static void main(String[] args) throws
SQLException, ClassNotFoundException{
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
                    if(con != null){
                    System.out.println("connection is established");
                    System.out.println("connection: "+con);
                    }
                    else
                    System.out.println("connection is not established");
                    Statement st = con.createStatement();
              ResultSet rs = st.executeQuery("select * from emp");
                    System.out.println("rs: "+rs);
                    st.executeQuery("select * from emp");
                    ResultSetMetaData rsmd  = rs.getMetaData();
                    System.out.println("Number of   Columns:
"+rsmd.getColumnCount());
                    System.out.println("Name of the Column :
"+rsmd.getColumnName(1));
                    System.out.println("Type of the Column :
"+rsmd.getColumnTypeName(1));
                    System.out.println("precision of Column:
"+rsmd.getPrecision(4));
                    System.out.println("scale of the Column:
"+rsmd.getScale(4));
                    System.out.println("constraint of
Column:"+rsmd.isNullable(1));

      }

}
```

Note: We cannot execut "desc" command from java program, we will get one error. That is java.sql.SQLExcepiton: invalid sqlstatement.

Desc command is not a sql command, it is design for sql prompt for windows os.

ParameterMetadata:

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

```
import java.sql.ParameterMetaData;
import java.sql.PreparedStatement;

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

	DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe"
,
				"system","manager");
		System.out.println("con: "+con);
		PreparedStatement ps = con.prepareStatement("insert into
values sbaj4to6(?,?,?)");
		ParameterMetaData pmd = ps.getParameterMetaData();
		System.out.println(pmd.getParameterCount());
		//System.out.println(pmd.getParameterMode(1));
		//System.out.println(pmd.getPrecision(1));
		//System.out.println(pmd.getScale(1));

	}
}
```

Note: native,oci, type2 are not support this feature.

Query:

A Query is a database command, that is used for performing an operation on db. The operations are

Create,drop, delete, insert, select , update , delete etc options.

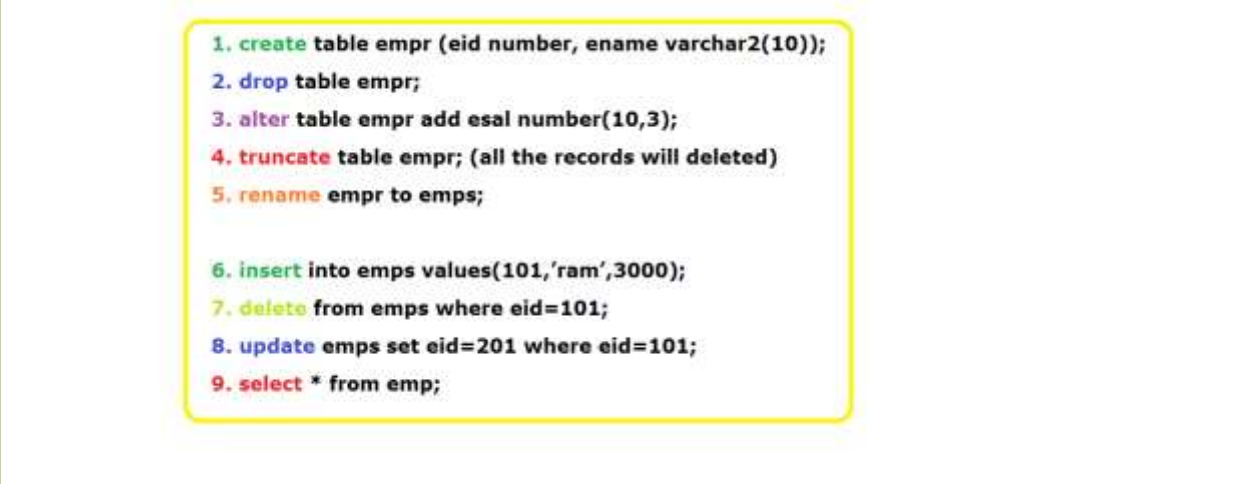
All queries are divided into 5 types.

1. **Data Definition Languages: (DDL): Used to define database objects.**
   a. **Create.**
   b. **Drop.**
   c. **Alter.**

   d. Truncate.
   e. Rename.
2. **Data Manipulate Languages: (DML):** It is used to manipulate the data, which is available in Objects.
   a. Insert.
   b. Delete.
   c. Update.
3. **Data Retrieval and query Language: (DRL/DQL):** It is used to read the information from database.
   a. Select.
4. **Transactional Control Language: (TCL):**
   a. Commit.
   b. Rollback.
   c. Savepoint.
5. **Data Control Language: (DCL):**
   a. Grant.
   b. Revoke.

Examples on Commands:

```
create table empr (eid number, ename varchar2(10));

drop table empr;

alter table empr add esal number(10,3);

truncate table empr; (all the records will deleted)

rename empr to emps;


insert into emps values(101,'ram',3000);

delete from emps where eid=101;

update emps set eid=201 where eid=101;

select * from emp;
```



create table empn(eid number(5) constraint eid_unq UNIQUE, ename varchar2(10);

Mainly queries are classified into two types.

Select (DQL)

Non-select

Again Non-select queries are two types.

Updatable (DML)

Queries that return nothing (DDL, TCL, DCL).

Statement Interface:

It is used to executing the queries in jdbc.

It will provide three methods.

1. **execute(String query).**
2. **executeUpdate(String query)**
3. **executeQuery(String query).**

execute() is used to run any type of query.

It will return Boolean value. The return value is true, if the query is select else returns false(insert query).

If the query execution failed it will returns SQL Exception.

Boolean b = st.execute("select * from emps where eid=101");

In above code assume, there is no record on 101 still it will return true.

executeUpdate() is used to run updatable queries and returns nothing.

It will return int value.

executeQuery() is used to execute given select query.

It will return set of results in the from ResultSet object.

Special Case:

We can get result, which generated by the execute() by using following methods.

a. getResultSet().
b. getUpdateCount().

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

public class Demo1 {

	public static void main(String[] args) throws
SQLException, ClassNotFoundException{
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
		if(con != null){
		System.out.println("connection is established");
			System.out.println("connection: "+con);
```

```
                }
                else
                System.out.println("connection is not established");
                Statement st = con.createStatement();
            /*ResultSet rs = st.executeQuery("select * from emp");
                System.out.println("rs: "+rs);*/
boolean b = st.execute("insert into emps values(101,'sam',4000)");

                if(b){
                    ResultSet rs = st.getResultSet();
                    System.out.println("^^^^^^^^");
                    while(rs.next()){
            System.out.println(rs.getInt(1)+"..."+rs.getString(2));
                    }
                }
                else{
                    int updateCount = st.getUpdateCount();
                System.out.println("updateCount: "+updateCount);

                    }
        }

}



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

public class Demo1 {

        public static void main(String[] args) throws
SQLException, ClassNotFoundException{
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
                if(con != null){
                System.out.println("connection is established");
                    System.out.println("connection: "+con);
                }
                else
```

```
                System.out.println("connection is not established");
                Statement st = con.createStatement();
            /*ResultSet rs = st.executeQuery("select * from emp");
                System.out.println("rs: "+rs);*/
boolean b = st.execute("select * from emps");

                if(b){
                    ResultSet rs = st.getResultSet();
                    System.out.println("^^^^^^^^");
                    while(rs.next()){
            System.out.println(rs.getInt(1)+"..."+rs.getString(2));
                    }
                }
                else{
                    int updateCount = st.getUpdateCount();
                System.out.println("updateCount: "+updateCount);

                    }
        }

}
```

Execute() can run all type of queries. But we need to write more line of code to view the records.

Code understandability is difficult.

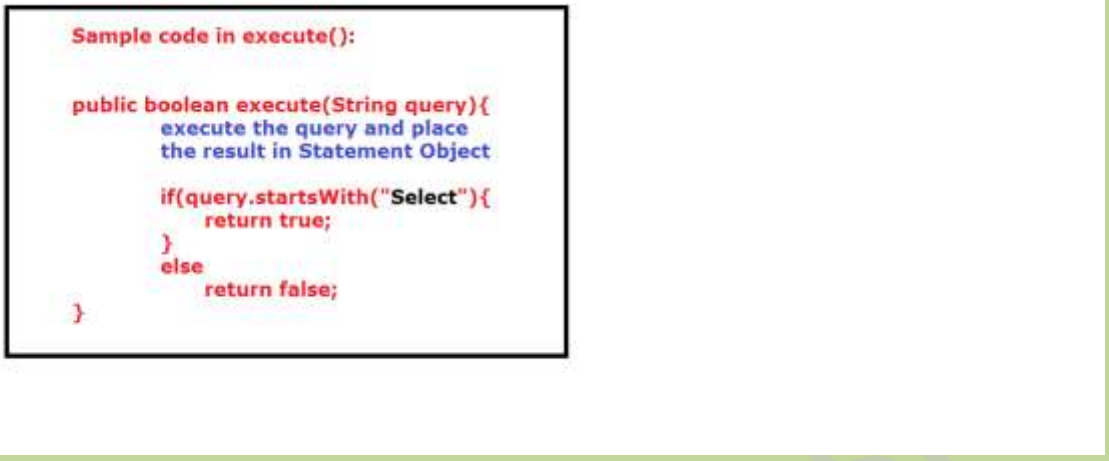
To run query and get the result within the single line we can go for two more methods.

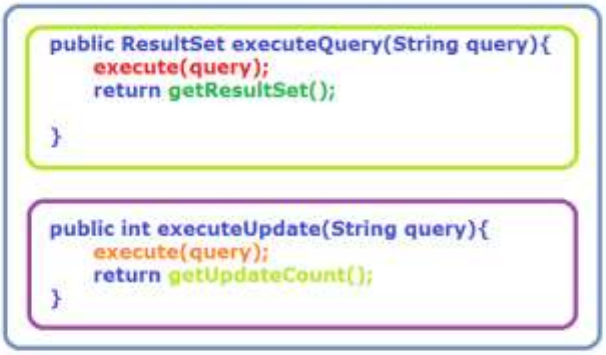
executeQuery()

executeUpdate()

If we are using executeUpdate(), that will be call execute() and that method will executeUpdateCount() for return the value.

If we are using executeQuery(), that will be call execute() and that method will call getResultSet() for return the results.






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

public class Demo1 {

        public static void main(String[] args) throws
SQLException, ClassNotFoundException{
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
                if(con != null){
                System.out.println("connection is established");
                        System.out.println("connection: "+con);
```

```
                    }
                    else
                    System.out.println("connection is not established");
                    Statement st = con.createStatement();
int count = st.executeUpdate("insert into emps
values(444,'kittu',5000)");
                        System.out.println("count: "+count);
            ResultSet rs = st.executeQuery("select * from emps");
                        while(rs.next()){

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

      }

}
```

Note: In the above program we using both executeUpdate() and executeQuery().

In which order using these method in that order only data will be stored in Statement object.

According above problem first executeUpdate() result will be placed into Statement object later executeQuery() result will be placed.

How to insert into a new employee data into a database:

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

public class Demo1 {

            public static void main(String[] args) throws
SQLException, ClassNotFoundException{
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
                    if(con != null){
```

```
                System.out.println("connection is established");
                        System.out.println("connection: "+con);
                }
Statement st = con.createStatement();
        int count = st.executeUpdate("insert into emps
        values(444,'kittu',5000)");
        System.out.println("count: "+count);
        }

}
```

If we execute above program multiple times, the same data will be inserting into emps table.

If the table having unique constraint then we need to face the following problem from second time onwards that is SQLException: unique constraint violated.

In data base side we need to execute this query:

```
create table empn(eid number(5) constraint eid_unq UNIQUE,
ename varchar2(10);
```

In java application we need to type following code.

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

public class Demo1 {

        public static void main(String[] args) throws
SQLException, ClassNotFoundException{
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
                if(con != null){
                System.out.println("connection is established");
                        System.out.println("connection: "+con);
                }
Statement st = con.createStatement();
        int count = st.executeUpdate("insert into emps
        values(444,'kittu')");
```

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

}
```

First executing the above program no problem data will be stored in db successfully. But from second time onwards we will face bellow Exception.

```
java.sql.SQLIntegrityConstraintViolationException: ORA-00001: unique constraint (SYSTEM.SID) violated
```

How to delete the data/record from Data Base:

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

public class Demo1 {

            public static void main(String[] args) throws
SQLException, ClassNotFoundException{
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
                    if(con != null){
                    System.out.println("connection is established");
                        System.out.println("connection: "+con);
                    }
Statement st = con.createStatement();
      int count = st.executeUpdate("delete from empn where
      eid=444”);
      System.out.println("count: "+count);
      }

}
```

How to update record in Database:

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

public class Demo1 {

		public static void main(String[] args) throws
SQLException, ClassNotFoundException{
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
			if(con != null){
			System.out.println("connection is established");
				System.out.println("connection: "+con);
			}
Statement st = con.createStatement();
	int count = st.executeUpdate("update empn set ename='kiran'
	where eid=101");
	System.out.println("count: "+count);
	}

}
```

How to create table in db from java appn:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class JdbcProgram5 {
	public static void main(String[] args)
	throws ClassNotFoundException, SQLException{
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection con = DriverManager.getConnection(
				"jdbc:oracle:thin:@localhost:1521:xe",
				"system",
				"manager");
		if(con!=null){
			System.out.println("con: "+con.hashCode());
			System.out.println("connection established");
		}
		else
			System.out.println("connection not established");
```

```
            Statement st = con.createStatement();
String query = "create table sbajram (eid number(5),ename
varchar2(20),esal number(10))";
            int count = st.executeUpdate(query);
            System.out.println("count: "+count);
      }
}
```

In the above program executeUpdate() will return value like zero(0).

The reason is executeUpdate() method always gives information about records/rows but not table.

How to drop the table in db from java appn:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class JdbcProgram5 {
      public static void main(String[] args)
      throws ClassNotFoundException, SQLException{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con = DriverManager.getConnection(
                        "jdbc:oracle:thin:@localhost:1521:xe",
                        "system",
                        "manager");
            if(con!=null){
                  System.out.println("con: "+con.hashCode());
                  System.out.println("connection established");
            }
            else
                  System.out.println("connection not established");
            Statement st = con.createStatement();
String query = "drop table sbajram";
            int count = st.executeUpdate(query);
            System.out.println("count: "+count);
      }
}
```

How to add new column to existing table from java appn:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class JdbcProgram5 {
```

```
	public static void main(String[] args)
	throws ClassNotFoundException, SQLException{
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection con = DriverManager.getConnection(
				"jdbc:oracle:thin:@localhost:1521:xe",
				"system",
				"manager");
		if(con!=null){
			System.out.println("con: "+con.hashCode());
			System.out.println("connection established");
		}
		else
			System.out.println("connection not established");
		Statement st = con.createStatement();
String query = "alter table sbaj add institute varchar2(20)";
		int count = st.executeUpdate(query);
		System.out.println("count: "+count);
	}
}
```

How to delete all the records the from table:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class JdbcProgram5 {
	public static void main(String[] args)
	throws ClassNotFoundException, SQLException{
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection con = DriverManager.getConnection(
				"jdbc:oracle:thin:@localhost:1521:xe",
				"system",
				"manager");
		if(con!=null){
			System.out.println("con: "+con.hashCode());
			System.out.println("connection established");
		}
		else
			System.out.println("connection not established");
		Statement st = con.createStatement();
		String query = "truncate table sbaj";
		int count = st.executeUpdate(query);
		System.out.println("count: "+count);
	}
```

```
}
```

How to rename the table in from java appn:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class JdbcProgram5 {
	public static void main(String[] args)
	throws ClassNotFoundException, SQLException{
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection con = DriverManager.getConnection(
				"jdbc:oracle:thin:@localhost:1521:xe",
				"system",
				"manager");
		if(con!=null){
			System.out.println("con: "+con.hashCode());
			System.out.println("connection established");
		}
		else
			System.out.println("connection not established");
		Statement st = con.createStatement();
		String query = "rename sbaj to sbajram";
		int count = st.executeUpdate(query);
		System.out.println("count: "+count);
	}
}
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Properties;
import java.util.Scanner;
public class JdbcProgram5 {
	public static void main(String[] args){
		Connection con=null;
		try{
			Properties p = new Properties();
			p.put("user", "system");
			p.put("password", "manager");
			String url = "jdbc:oracle:thin:@localhost:1521:xe";
			con = DriverManager.getConnection(url,p);
			System.out.println("con: "+con.hashCode());
			Statement st = con.createStatement();
```

```
                Scanner scan = new Scanner(System.in);
                System.out.println("enter student id number");
                int sid = scan.nextInt();
                System.out.println("etner student name ");
                String sname = scan.next();
                System.out.println("enter course fee");
                int sfee = scan.nextInt();
      //String query = "insert into sbaj values(sid,sname,sfee)";
      String     query = "insert into sbaj
values("+sid+",'"+sname+"',"+sfee+")";
                int count = st.executeUpdate(query);
                System.out.println("count: "+count);
           }catch(Exception e){
                e.printStackTrace();
           }
           finally{
                if(con!=null){
                     try {
                          con.close();
                     } catch (SQLException e) {
                          e.printStackTrace();
                     }
                }
           }
      }
}
```

Note: Whenever we create a table from java application by using executeUpdate (-) we will get result is zero. That means don't be think table is not create in the db. Basically executeUpdate (-) will provides number records are update or not in the table, but it will not giving any information about table creation.

If we want to know about whether the table is created or not go check in db.

Based on the values we are placing in query, queries are divided into three types.

a. **Static SQL query.**
b. **Dynamic SQL query.**

c. **Stored Procedures.**

Static SQL query:

A query, that has values directly is called static sql query.

Ex: select * from empn where eid = 101;

Insert into empn values (102,'varun');

Dynamic SQL query:

A query that will be getting parameter values at run time means meanwhile of executing the program is called Dynamic SQL Query.

Query parameter is represented by using '?'.

By default '?' is act as IN parameter.

In oracle we have three types of parameters.

a. **IN → Input**
b. **Out → Output**
c. **INOUT → Both Input and Output.**

All these 3 types of parameters are represented by '?'. It is also called as "place holders".

Insert into empn values(?,?);

We must set values for place holders (?) at run time meanwhile of query execution.

While inserting the values to into place holder (?) we must be represent place holder with index position.

In the above insert query first '?' index is 1 and second '?' index is 2.

Statement and Types of Statement:

Statement is one java object, which is used to send and execute the query sql and pl/sql queries.

Internals of Statement Object:

1. **Validating sql query (based on standard rules).**
2. **Compiling sql query (converting into internal db instructions).**
3. **Prepared a build plan for query using db own algorithms.**
4. **Process the build plan to perform operations on db.**
5. **Finally destroy the build plan and returns appropriate result to java application.**

We have three types of statement objects.

1. **Statement.**
2. **PreparedStatement.**
3. **CallableStatement**

Statement interface or object is root interface for all types of statement interface, is used for executing the static query.

PrepatedStatement is sub interface for Statement interface, is used for executing pre-compile static or dynamic query.

CallableStatement is sub interface for PreparedStatement interface, is used to executing the stored procedures.

In above I already noticed Statement interface internals. Here I will give full information about Statement interface internals.

These internals technically we can call as build plan.

Build plan/SQL contains seven steps.

1. **Verifying query syntax. If valid then**
2. **Verifying given table and columns existed or not. If available**
3. **Generating or creating C-language program for the Sql query.**
4. **Compiling C-Language program.**
5. **Executing the above compiled C-Language program.**
6. **Finally returning the result to java app.**
7. **After successfully return the result, destroy the above C-Language program.**

If we are using Statement interface every time these above seven steps are going to executing repeatedly.

From java side it is performance issue.

So we need create the build plan only one and reuse it then we should go for concept like PreparedStatement interface.

If we are using Statement interface for sending the dynamic values we need to face another program that is decrease query readability.

Example to insert two values into db we need to use following query like

Insert into empn values(105, 'swathi');

"Insert into empn values ("+eid+",' "+ename+" ')";

To overcome the above two problems like Query Syntax readability and avoiding Build plan recreation we should move to PreparedStatement interface.

```
package jdbc.aj.ram;

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

public class JdbcProgram {
	public static void main(String[] args)throws SQLException {
		Connection con = ConnectionFactory.getConnect();
		System.out.println("con: "+con);

		Scanner scan = new Scanner(System.in);
		Statement st = con.createStatement();
		System.out.println("enter student details");
		for(int i=1;i<=5;i++){
			System.out.println("enter student number");
			int sno = scan.nextInt();
			System.out.println("enter student name");
			String sname = scan.next();
			System.out.println("enter student fee");
			int sfee = scan.nextInt();
```

```
                String query = "insert into sbaj
values("+sno+",'"+sname+"',"+sfee+")";
                st.executeUpdate(query);

          }
     }
}
import java.sql.Connection;
import java.sql.Statement;
import java.sql.PreparedStatement;
import java.sql.DriverManager;
public class FirstJdbcProg {
     public static void main(String[] args) throws Exception{
          System.out.println("=============");
          Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection

("jdbc:oracle:thin:@localhost:1521:xe","system", "manager");
          System.out.println("con: "+con);
Statement st = con.createStatement();
          System.out.println("st: "+st);
PreparedStatement ps = con.prepareStatement("insert into
sbajemp(?,?,?,?)");
          System.out.println("ps: "+ps);
     }
}
```

Java always provides rules to db vendors in the form of interface like Connection, Statement, PreparedStatement.

The rules are implemented by the database vendors by using classes like
T4CConnection, T4CStatement, T4CPreparedStatement.

In these classes method database vendors provides logic like creating objects and placed into interface reference variables.

The implementation classes may be different from one database vendor to another database vendor.

PreparedStatement:

PreparedStatement provides bellow benefits.

Those are

a. **Build Plan or Sql plan created only one time meanwhile of preparedstatement object creation and plan placed into that object.**
b. **In place of variables we can use '?' so no '+' operators.**
c. **'?' by default provides varchar type column, so no need to write single quotes(`), these single quotes automatically added.**

```
package jdbc.aj.ram;

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

public class JdbcProgram {
	public static void main(String[] args)throws SQLException {
		Connection con = ConnectionFactory.getConnect();
		System.out.println("con: "+con);

		Scanner scan = new Scanner(System.in);
		PreparedStatement ps =
			con.prepareStatement("select * from sbaj where
sname=?");
		System.out.println("enter student name");
		String sname = scan.next();
		ps.setString(1, sname);

		ResultSet rs = ps.executeQuery();
		while(rs.next()){
			System.out.println(rs.getInt(1)+"\t"+rs.getString(2)
					+"\t"+rs.getInt(3));
		}
	}
```

```
}


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

public class Demo1 {

		public static void main(String[] args) throws
SQLException, ClassNotFoundException{
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
			if(con != null){
			System.out.println("connection is established");
				System.out.println("connection: "+con);
			}
			else
			System.out.println("connection is not established");
PreparedStatement ps = con.prepareStatement("insert into empn
values(?,?)");
				Scanner scan = new Scanner(System.in);
				System.out.println("enter id");
				int eid=scan.nextInt();
				System.out.println("enter ename");
				String ename = scan.next();
				ps.setInt(1, eid);
				ps.setString(2, ename);
				int x = ps.executeUpdate();
				System.out.println("x: "+x);

	}

}


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

```
public class Demo1 {

		public static void main(String[] args) throws
SQLException, ClassNotFoundException{
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
			if(con != null){
			System.out.println("connection is established");
				System.out.println("connection: "+con);
			}
			else
			System.out.println("connection is not established");
PreparedStatement ps = con.prepareStatement("insert into empn
values(?,?)");
				ps.setInt(1, 104);
				ps.setString(2, "ramesh");


				int x = ps.executeUpdate();
				System.out.println("x: "+x);

				ps.setInt(1, 105);
				ps.setString(2, "mahesh");
				int y = ps.executeUpdate();
				System.out.println("x: "+y);
	}
}
```

Note: To execute the query one time better prefer Statement object.

To execute the query multiple times better prefer PreparedStatement object.

Note: Whenever we work with PreparedStatement better execute the query multiple times, otherwise it will leads to performance issue.

Compare to Statement object, PreparedStatement object not destroy the sql plan within the less time after query execution, we must call close() explicitly.

```
package jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;
public class JdbcDemo {
      public static void main(String[] args)
      throws SQLException{
            Connection con = DriverManager.getConnection(

      "jdbc:oracle:thin:@localhost:1521:xe","system","manager");
            System.out.println("con: "+con);
            System.out.println("select your operation");
            System.out.println("1.INSERT\t2.DELETE");
            System.out.println("3.TRUNCATE\t4.UPDATE");
            System.out.println("5.SELECT");
            Scanner scan = new Scanner(System.in);
            int operation = scan.nextInt();
            switch(operation){
            case 1: System.out.println("YOUR ARE SELECTING INSERT
OPERATION");
PreparedStatement ps1 = con.prepareStatement("insert into sbaj2_4
values(?,?,?,?)");
                        System.out.println("enter student id");
                        int sid = scan.nextInt();
                        System.out.println("enter studnet name");
                        String sname = scan.next();
                        System.out.println("enter studnet fee");
                        int sfee = scan.nextInt();
                        System.out.println("enter studnet age");
                        int sage = scan.nextInt();
                        ps1.setInt(1, sid);
                        ps1.setString(2, sname);
                        ps1.setInt(3, sfee);
                        ps1.setInt(4, sage);
                        int count1 = ps1.executeUpdate();
                        System.out.println("recored inserted
successfully: "+count1);
                        break;
            case 2: System.out.println("YOU ARE SELECTING DELETE
OPERATION");
PreparedStatement ps2 = con.prepareStatement("delete from sbaj2_4
where sno=?");
                        System.out.println("enter studnet number");
```

```
                    int sid1 = scan.nextInt();
                    ps2.setInt(1, sid1);
                    int count2 = ps2.executeUpdate();
                    System.out.println("recored deleted: "+count2);
                    break;
          case 3: System.out.println("YOU ARE SELECTING TRUNCATE
OPERATION");
          PreparedStatement ps3 = con.prepareStatement("truncate table
sbaj2_4");
                    int count3 = ps3.executeUpdate();
                    System.out.println("all records are delected:
"+count3);
                    break;
          case 4: System.out.println("YOU ARE SELECTING UPDATE
OPERATION");
PreparedStatement ps4 =
     con.prepareStatement("update sbaj2_4 set sname=? where sno=?");
                    System.out.println("enter studnet name for
updation");
                    String sname1 = scan.next();
                    System.out.println("enter studnet number");
                    int sid2 = scan.nextInt();
                    ps4.setString(1, sname1);
                    ps4.setInt(2, sid2);
                    int count4 = ps4.executeUpdate();
                    System.out.println("records are updated:
"+count4);
                    break;
          case 5:   System.out.println("YOU ARE SELECTING SELECT
OPERATION");
          PreparedStatement ps5 = con.prepareStatement("select * from
sbaj2_4");
          ResultSet rs = ps5.executeQuery();
          while(rs.next()){

     System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getI
nt(3)
               +"\t"+rs.getInt(4));
          }

          }
     }
}
```

```java
package jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Scanner;
public class JdbcDemo {
	public static void main(String[] args)
	throws SQLException{
		Connection con = DriverManager.getConnection(

	"jdbc:oracle:thin:@localhost:1521:xe","system","manager");
		System.out.println("con: "+con);
		String query = "update sbaj2_4 set sfee=? where sno=?";
		PreparedStatement ps= con.prepareStatement(query);
		Scanner scan = new Scanner(System.in);
		System.out.println("Enter Student fee");
		int sfee = scan.nextInt();
		System.out.println("Enter Student number");
		int sno = scan.nextInt();
		ps.setInt(1, sfee);
		ps.setInt(2, sno);
		int count = ps.executeUpdate();
		System.out.println("count: "+count);
	}
}
package jdbc;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;
public class JdbcDemo {
	public static void main(String[] args)
	throws SQLException{
		Connection con = DriverManager.getConnection(


"jdbc:oracle:thin:@localhost:1521:xe","system","manager");
		System.out.println("con: "+con);
```

```
                Scanner scan = new Scanner

(System.in);
                System.out.println("do you want do the

operations if yes type yes if no type no");
                String flag = scan.next();
                while(flag.equalsIgnoreCase("yes")){


                System.out.println("select your

operation");
                System.out.println("1.INSERT

\t2.DELETE");
                System.out.println("3.TRUNCATE

\t4.UPDATE");
                System.out.println("5.SELECT");
                int operation = scan.nextInt();

           switch(operation){
           case 1: System.out.println("YOUR ARE

SELECTING INSERT OPERATION");
PreparedStatement ps1 = con.prepareStatement("insert into sbaj2_4

values(?,?,?,?)");
                     System.out.println("enter

student id");
                     int sid = scan.nextInt();
                     System.out.println("enter

studnet name");
                     String sname = scan.next();
                     System.out.println("enter

studnet fee");
                     int sfee = scan.nextInt();
                     System.out.println("enter

studnet age");
                     int sage = scan.nextInt();
                     ps1.setInt(1, sid);
```

```
                    ps1.setString(2, sname);
                    ps1.setInt(3, sfee);
                    ps1.setInt(4, sage);
                    int count1 =

ps1.executeUpdate();
                    System.out.println("recored

inserted successfully: "+count1);
                    break;
          case 2: System.out.println("YOU ARE SELECTING

DELETE OPERATION");
PreparedStatement ps2 = con.prepareStatement("delete from sbaj2_4

where sno=?");
                    System.out.println("enter

studnet number");
                    int sid1 = scan.nextInt();
                    ps2.setInt(1, sid1);
                    int count2 =

ps2.executeUpdate();
                    System.out.println("recored

deleted: "+count2);
                    break;
          case 3: System.out.println("YOU ARE SELECTING

TRUNCATE OPERATION");
          PreparedStatement ps3 = con.prepareStatement

("truncate table sbaj2_4");
                    int count3 =

ps3.executeUpdate();
                    System.out.println("all

records are delected: "+count3);
                    break;
          case 4: System.out.println("YOU ARE SELECTING

UPDATE OPERATION");
PreparedStatement ps4 =
```

```
     con.prepareStatement("update sbaj2_4 set sname=? where

sno=?");
                    System.out.println("enter

studnet name for updation");
                    String sname1 = scan.next();
                    System.out.println("enter

studnet number");
                    int sid2 = scan.nextInt();
                    ps4.setString(1, sname1);
                    ps4.setInt(2, sid2);
                    int count4 =

ps4.executeUpdate();
                    System.out.println("records

are updated: "+count4);
                    break;
          case 5:   System.out.println("YOU ARE

SELECTING SELECT OPERATION");
          PreparedStatement ps5 = con.prepareStatement

("select * from sbaj2_4");
          ResultSet rs = ps5.executeQuery();
          while(rs.next()){
               System.out.println(rs.getInt

(1)+"\t"+rs.getString(2)+"\t"+rs.getInt(3)
               +"\t"+rs.getInt(4));
          }//while

          }//switch
          System.out.println("do you want continue if no

type no");
          flag = scan.next();
          }//while
     }
}
```

CallableStatement:

CallableStatement is one of the jdbc statement object, it is used for executing the stored procedures and functions.

It is the sub interface of PreparedStatement interface.

Stored procedure is one pl/sql program.

It contains set of sql queries for execution with proper validation or conditions and calculations to perform crud operations on db table.

Advantages of Stored Procedure:

Separate sql queries from java program and stores them permanently in db server in compiled format.

So parallel development is possible, that means java program and stored procedure programmer can work at a time.

StoredProcedure is reusable.

So All the programming language can use the same stored procedure.

Procedure creation and calling same for all language.

No need to change the java program to change the one db to another db.

It is precompiled program, so execution is happen very quickly.

Db query statements are different from one db to another so better to avoid the writing queries in our app. So whenever we change db no need to disturb the java program.

PreparedStatement query precompiled from second

time onwards, SQL plan is temporary(upto program

execution). It is not reusable.

It is only suitable for current and one PreparedStatement object only.

PreparedStatement connection is closed SQL or build plan destroyed.

A procedure precompiled SQL plan is permanently stored in database and reusable through multiple application and languages also.

Differences between Procedure and Functions in DB:

StoredProcedure are pre-compile objects which are compiled for first time and its compiled format will be saved and executes whenever it is called where as functions are compiled and called every time when it is called.

Procedures may or may not return values where as functions must return value.

Procedures have input and output parameters where as functions have input parameters only.

Functions can calls from procedures where as procedures cannot called from functions.

Syntax to create Stored Procedure:

create or replace procedure procedure_name(parameter list with , separator)

is

variable declarations;

begin

logic with sql commands and pl/sql operators, conditions, loops and exception handling

end;

/

Create or replace function function_name(parameter list)

Return <type> here type is db datatype

Is

Variable declaration

Begin

logic

Return <result>;

End;

/

Types of Parameters:

To send input and to read output from stored procedure, every db will support three types of parameters.

1. **IN**
2. **OUT**
3. **INOUT**

IN parameter is used to send input values.

OUT parameter is used to return output values.

INOUT parameter is used to read and also return output value.

To create parameter as Input we need to write IN in parameter declaration.

By default parameters are comes under Input parameters only so no need to declare IN in parameter declaration.

To create parameter as output we need to write OUT in parameter declaration.

```
Create or replace procedure procedurename(
        Eno IN Number,
        Ename OUT varchar2,
        Esal OUT number,
        Empdept OUT varchar2)
```

To communicate with above procedure from java application, we need to send eno as input to procedure and get ename, esal, empdept as output.

Syntax to call procedure from java application:

"{call procedure_name (?,?)}"

Syntax to call function from java application:

"{?:=cal functionname(?)}"

CallableStatement object we can create by using following method. That is prepareCall(String call);

```
create table empram(eno number(5) Primary key, ename
varchar2(10),sal number(7), dept varchar2(10));
```

```
insert into empram values(101,'ram',5000,'software');

insert into empram values(102,'sam',6000,'testing');
```

```
create or replace procedure empramprocedure (empnum IN
number,incrsal IN number,empname OUT varchar2,empsal OUT
number, empdept out varchar2)

 is

   oldsal empram.sal%type;

  begin

    select sal into oldsal from empram where eno=empnum;

    update empram set sal=oldsal+incrsal;

    commit;

  select ename,sal,dept into empname,empsal,empdept from
empram where eno=empnum;

  end;

   /
```

Communicating with database through Procedure:

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

public class Demo1 {

            public static void main(String[] args) throws
SQLException, ClassNotFoundException{
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
```

```
                if(con != null){
                System.out.println("connection is established");
                        System.out.println("connection: "+con);
                }
                else
                System.out.println("connection is not established");
CallableStatement cs = con.prepareCall("{call
empramprocedure(?,?,?,?,?)}");

                cs.registerOutParameter(3,Types.VARCHAR);
                cs.registerOutParameter(4,Types.NUMERIC);
                cs.registerOutParameter(5,Types.VARCHAR);

                Scanner scan = new Scanner(System.in);
                System.out.println("enter eno");
                int eno = scan.nextInt();
                System.out.println("enter sal for increment");
                int incrsal = scan.nextInt();

                cs.setInt(1,eno);
                cs.setInt(2, incrsal);
                cs.execute();


                String ename = cs.getString(3);
                int sal = cs.getInt(4);
                String dept = cs.getString(5);

                System.out.println(ename+"..."+sal+"..."+dept);
                cs.close();
                        con.close();
        }
}
```

```
public class JdbcProgram11 {
        public static void main(String[] args)
        throws SQLException{
                Connection con = ConnectionFactory.getConnect();
                System.out.println("con: "+con);
                CallableStatement cs =
                        con.prepareCall("{call
sbajramprocedure(?,?,?,?,?,?)}");
                Scanner scan = new Scanner(System.in);
                System.out.println("enter employee id");
```

```
                int eno = scan.nextInt();
                System.out.println("enter increment salary");
                int incrsal = scan.nextInt();
                cs.setInt(1, eno);
                cs.setInt(2, incrsal);
                cs.registerOutParameter(3, Types.NUMERIC);
                cs.registerOutParameter(4, Types.VARCHAR);
                cs.registerOutParameter(5, Types.NUMERIC);
                cs.registerOutParameter(6, Types.VARCHAR);
                cs.execute();

                /*ResultSet rs = cs.executeQuery();
                System.out.println("rs: "+rs);
                rs.next();
                System.out.println(rs.getInt(1));
                System.out.println(rs.getString(2));
                System.out.println(rs.getInt(3));
                System.out.println(rs.getString(4));
                */
                int empnumber = cs.getInt(3);
                String empName = cs.getString(4);
                int empSal = cs.getInt(5);
                String empDept = cs.getString(6);

      System.out.println(empnumber+".."+empName+"..."+empSal
+"..."+empDept);


      }
}
```

Whenever we working with stored procedure through CallableStatement don't read data in the form of ResultSet,

we can't do fetch operations by using next().

How to communicate with static queries by using callable statemen object:

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

```
import java.sql.SQLException;
import java.sql.Types;
import java.util.Scanner;
import jdbc.aj.ram.ConnectionFactory;
public class JdbcProgram11 {
	public static void main(String[] args)
	throws SQLException{
		Connection con = ConnectionFactory.getConnect();
		System.out.println("con: "+con);
		CallableStatement cs = con.prepareCall("select * from
empram");
		ResultSet rs = cs.executeQuery();
		while(rs.next()){

	System.out.println(rs.getInt(1)+".."+rs.getString(2)+".."+
		rs.getInt(3)+"..."+rs.getString(4));
		}

	}
}
```

executeQuery() is not available in the CallableStatement interface, it is available in PreparedStatement interface.

How to communicate with dynamic queries by using callable statemen object:

```
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Types;
import java.util.Scanner;
import jdbc.aj.ram.ConnectionFactory;
public class JdbcProgram11 {
	public static void main(String[] args)
	throws SQLException{
		Connection con = ConnectionFactory.getConnect();
		System.out.println("con: "+con);
		CallableStatement cs = con.prepareCall("insert into
empram values(?,?,?,?)");
		Scanner scan = new Scanner(System.in);
```

```
            System.out.println("enter employee number");
            int eno = scan.nextInt();
            System.out.println("enter employee name");
            String ename = scan.next();
            System.out.println("enter employee sal");
            int esal = scan.nextInt();
            System.out.println("enter employee dept");
            String edept = scan.next();
            cs.setInt(1, eno);
            cs.setString(2, ename);
            cs.setInt(3, esal);
            cs.setString(4,edept);
            int count = cs.executeUpdate();
            System.out.println("count: "+count);

            }
}
```

How to function in database:

```
create or replace function empramfunction(empno number)
 return number as amount number;
 begin
 select sal into amount from empram where eno=empno;
 return amount;
end;
 /
```

Communicating with database through function:

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

```
public class Demo1 {

		public static void main(String[] args) throws
SQLException, ClassNotFoundException{
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
			if(con != null){
			System.out.println("connection is established");
				System.out.println("connection: "+con);
			}
			else
			System.out.println("connection is not established");
CallableStatement cs = con.prepareCall("{call
?:=empramfunction(?)}");

			Scanner scan = new Scanner(System.in);
			System.out.println("enter emp no: ");
			int eno = scan.nextInt();
			cs.setInt(2, eno);
			cs.registerOutParameter(1, Types.INTEGER);
			cs.execute();
			int sal = cs.getInt(1);
			System.out.println("sal: "+sal);
		con.close();

}

}
```

Transaction Management:

The process of taking database state from one state to another state completely or none is called Transaction Management.

In transaction management more than one updatable query executed.

We need to commit results of all updatable queries only if all updatable queries successfully executed.

If anyone updatable query is failed, we need to rollback all queries result and should bring back to database state to its original state before our transaction is started.

To commit and rollback transaction results, we must use methods from connection interface.

They are:

void setAutoCommit(Boolean autocommit)throws SQLException

By using above method, we can change auto commit mode true or false.

If we pass true, each updatable statement (insert, delete, update) result will be commit as individual transactions.

If we pass false, all updatable statements are grouped in a single transaction, then all SQL Updatable queries results is committed or rollback at a time by the method calls con.commit() or con.rollback().

By default all new connections are opened in auto commit mode true only.

Public void commit() throws SQLException

This method stores or saves all changes made in database since the previous commit or rollback and releases any database currently held by this connection object.

If we are insert new record that new record is locked until it is committed. Other database clients with new connection cannot access this row.

Whenever we run update or delete query on a particular row that row is locked, that will not access by the other connection client.

Public void rollback() throws SQLException:

This method reverts the changes done in db in current transaction and releases the locks.

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

public class Demo1 {

		public static void main(String[] args){
			try{
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
			if(con != null){
				System.out.println("connection is
established");
				System.out.println("connection: "+con);
			}
			else
				System.out.println("connection is not
established");


			con.setAutoCommit(false);
			Statement st = con.createStatement();
			st.execute("insert into  emps
values(110,'a',4444)");
			st.execute("update emps set eid=301 where
ename='kam'");
			st.execute("delete from emps where eid=444");
			con.commit();
			System.out.println("*****************8");
			}catch(SQLException e){
				System.out.println("some problem");
			}
}

}
```

IN the above program if we are not write con.commit() records are not updated into main table. The reason is setAutoCommit(false) so records are not updated by default. We need use commit().

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

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

public class Demo1 {

		public static void main(String[] args) throws
SQLException, ClassNotFoundException{
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
			if(con != null){
				System.out.println("connection is
established");
				System.out.println("connection: "+con);
			}
			else
				System.out.println("connection is not
established");



			Statement st = con.createStatement();
			st.execute("insert into  emps
values(110,'a',4444)");
			st.execute("update emps set eid=301 where
ename='kam'");
			st.execute("delete from emps where eid=444");
}

}
```

In the above program records directly committed into main table the reason the setAutoCommit(true) by default.

The bellow program will the complete information about Transaction management and commit() and rollback().

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

```
public class Demo1 {

		public static void main(String[] args){
			Connection con =null;
			Statement st = null;
			try{
				con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
			if(con != null){
				System.out.println("connection is
established");
				System.out.println("connection: "+con);
			}
			else
				System.out.println("connection is not
established");
			String query1 = "insert into empn values(666,'fff')";
			String query2 = "insert into empn
values(555,'ggg')";
			con.setAutoCommit(false);
			st = con.createStatement();

			st.executeUpdate(query1);
			System.out.println("1st query inserted");
			Thread.sleep(20000);
			st.executeUpdate(query2);
			System.out.println("2nd query inserted");
			Thread.sleep(20000);

			con.commit();
			System.out.println("records are commited");
			Thread.sleep(20000);
			}catch(Exception e){
				System.out.println("problem comes here");
				e.printStackTrace();
				if(con!=null){
					try{
						con.rollback();
					}catch(Exception e1){
						e1.printStackTrace();
					}
				}
```

```
            }
      }
}
```

In the above both query1 and query2 are valid then both the insert statement will updated to database main table with support of commit(), otherwise no insert query will be updatable into database main table.

Program on savePoint():

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Savepoint;
import java.sql.Statement;
public class TransactionManagementDemo {
      public static void main(String[] args) {
            try{
                  Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con = DriverManager.getConnection(
      "jdbc:oracle:thin:@localhost:1521:xe","system","manager");
            System.out.println("con: "+con);
            con.setAutoCommit(false);
            Statement st = con.createStatement();
            st.executeUpdate(
                  "insert into sbajemp values
(204,'ram4',99991,'hr4')");
            st.executeUpdate(
                  "insert into sbajemp values
(205,'ram5',99992,'hr5')");
            Savepoint sv = con.setSavepoint();

            st.executeUpdate(
                  "insert into sbajemp values
(201,'ram1',99993,'hr1')");
            st.executeUpdate(
                  "insert into sbajemp values
(202,'ram2',99994,'hr2')");
            st.executeUpdate(
                  "insert into sbajemp values
(203,'ram3',99995,'hr3')");
```

```
            con.rollback(sv);
            con.commit();
            }catch(ClassNotFoundException e){
                  System.out.println("class is not existed");
                  e.printStackTrace();
            }
            catch(SQLException e){
                  System.out.println("problem database connection");
                  e.printStackTrace();
            }
      }
}
```

Batch Updations:

Batch updates is the process of sending and executing multiple updatable queries (Insert, Update, Delete) is called batch updation.

Whenever we execute multiple updatable queries as group or execute multiple updatable queries with dependency we must choose batch updations.

The advantages of the batch updations is performance is improved in executing multiple updatable queries, because number of hits on the database are reduced and moreover number of transformation control from java app to db will be reduced.

If we want to send 1000 employee details into db, we need to write 1000 insert queries and 1000 times we need to hit db and 1000 times transfer control from java app to db.

If we want to use batch updations, we can send 1000 insert queries only one time.

Other advantage is if first query is successfully executed then only second query will be executed successfully. This is called dependency of queries.

To develop batch updations in java, we can take support of Statement and PreparedStatement interface and their methods.

Void addBatch(String query) throws SQLException

The above method adds given sql command to this Statement object batch.

Int[] executeBatch() throws SQLException

The above method is used to handover the all sql commands stored in statement object batch to the database.

If all the sql statements executed successfully, it returns an int[].

If any exception is raised then we will get BatchUpdateException. It is a subclass of SQLException.

Public void clearBatch()SQLException

The above method is used to removes all queries from the batch and make this statement object batch as empty.

PrepatedStatement interface methods for batch updataions:

void addBatch() throws SQLException

the above is used to adds the same sql query, which is stored in PreparedStatement object, multiple times to PreparedStatement object batch with different values.

What is difference between in addBatch() in Statement and PreparedStatment objects?

addBatch(-) of Statement is parameterized method, query passed at adding time only.

addBatch() of PreparedStatement is non-parameterized method.

Program on Batch Updation:

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

public class Demo1 {

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

```
		Connection con =null;
		Statement st = null;
		try{
			con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
		if(con != null){
		System.out.println("connection is established");
			System.out.println("connection: "+con);
		}
		else
		System.out.println("connection is not established");
			con.setAutoCommit(false);
			st=con.createStatement();
	String iquery= "insert into empram values(103, 'varun', 7000,
'hr')";
	String uquery = "update empram set dept='developer' where
eno=101";
	String dquery = "delete from empram where eno=102";
			st.addBatch(iquery);
			st.addBatch(uquery);
			st.addBatch(dquery);

			int[] ia = st.executeBatch();
			System.out.println(ia[0]+" rows inserted");
			System.out.println(ia[1]+" rows updated");
			System.out.println(ia[2]+" rows delete");
			con.commit();
		System.out.println("recoreds commit completed");
		}catch(Exception e){
			e.printStackTrace();
			if(con!=null){
				try{
					con.rollback();

				}catch(Exception e1){
					e1.printStackTrace();
				}
			}


		 }

}
```

```
}
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Statement;

public class Jdbcprogram15 {
public static void main(String[] args){
Connection con=null;                    try{
Class.forName("oracle.jdbc.driver.OracleDriver");
con = DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe",
“system",
"manager");
System.out.println("con: "+con);
con.setAutoCommit(false);
PreparedStatement ps = con.prepareStatement(
"insert into empram values(?,?,?,?) ");
ps.setInt(1, 205);
ps.setString(2, "megha");
ps.setInt(3, 999999);
ps.setString(4,"faculty");
ps.addBatch();
```

```
int a[]=ps.executeBatch();

System.out.println(a);

con.commit();

}catch(Exception e){

System.out.println("controle comes to here");

e.printStackTrace();

if(con!=null){

try {

con.rollback();

} catch (SQLException e1) {

e1.printStackTrace();

}

}

}

}
```

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Scanner;
public class Jdbcprogram15 {
	public static void main(String[] args){
		Connection con=null;
	try{
		Class.forName("oracle.jdbc.driver.OracleDriver");
		con = DriverManager.getConnection(
				"jdbc:oracle:thin:@localhost:1521:xe",
				"system",
				"manager");
		System.out.println("con: "+con);
```

```
            con.setAutoCommit(false);

            PreparedStatement ps =
                        con.prepareStatement("insert into empram
values(?,?,?,?)");
            Scanner scan = new Scanner(System.in);
            for(int i =1; i<=3;i++){
                  System.out.println("enter employee number");
                  int eno = scan.nextInt();
                  System.out.println("enter employee name");
                  String ename= scan.next();
                  System.out.println("enter employee salaray");
                  int esal = scan.nextInt();
                  System.out.println("enter employee department");
                  String edept = scan.next();
                  ps.setInt(1,eno);
                  ps.setString(2, ename);
                  ps.setInt(3,esal);
                  ps.setString(4, edept);
                  ps.addBatch();
            }
      int a[]=ps.executeBatch();
      System.out.println(a[0]);
      System.out.println(a[1]);
      System.out.println(a[2]);
            con.commit();
      }catch(Exception e){
            System.out.println("controle comes to here");
            e.printStackTrace();
            if(con!=null){
                  try {
                        con.rollback();
                  } catch (SQLException e1) {
                        e1.printStackTrace();
                  }
            }
      }
   }
}
```

How to insert the image into database table:

```
import java.io.File;
import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.Statement;
import java.util.Scanner;
public class Demo1 {

		public static void main(String[] args){
			Connection con =null;
			Statement st = null;
			try{
				con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
			if(con != null){
			System.out.println("connection is established");
				System.out.println("connection: "+con);
			}
			else
			System.out.println("connection is not established");
			PreparedStatement ps =
con.prepareStatement("insert into empblob values(?,?)");
			Scanner scan = new Scanner(System.in);
			System.out.println("enter some emp id:");
			int eno = scan.nextInt();
			ps.setInt(1, eno);
			File f = new File("anni 097.jpg");
			FileInputStream fis = new FileInputStream(f);
			ps.setBinaryStream(2, fis, (int)f.length());
			int i = ps.executeUpdate();
			System.out.println(i+".is inserted");
			}catch(Exception e){
				e.printStackTrace();
			}
}
}
```

Uploading and downloading video file from db:

```
import java.io.ByteArrayInputStream;
```

```
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.sql.*;
public class VideoASInput {
	public static void main(String[] args) throws SQLException,
IOException{
		Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","sys
tem","manager");
		System.out.println(con);
		Statement st = con.createStatement();
	int count = st.executeUpdate("create table video(video blob)");
	PreparedStatement ps = con.prepareStatement("insert into video
values(?)");
	File f = new File("C:\\Users\\Public\\Videos\\Sample
Videos\\Wildlife.wmv");
	FileInputStream fis = new FileInputStream(f);
	ByteArrayOutputStream byins = new ByteArrayOutputStream();
	while (fis.available()>0) {
		byins.write(fis.read());
		}
		byte[] videobytes = byins.toByteArray();
	ps.setBytes(1, videobytes);
	int count1=ps.executeUpdate();
	System.out.println(count1);
		Statement st1 = con.createStatement();
		ResultSet rs =st1.executeQuery("select * from video");
		while(rs.next())
		{
			File file1 = new File("E://ram.mp4");
			FileOutputStream foStream = new
FileOutputStream(file1);
			InputStream is2 = rs.getBinaryStream(1);
			int i=0;
			while((i=is2.read())!=-1){
				foStream.write(i);
			}
			}
		}

}
```

Reading the image from data base:

```
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 Demo1 {

	public static void main(String[] args){
		Connection con =null;
		Statement st = null;
		try{
			con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
		if(con != null){
		System.out.println("connection is established");
			System.out.println("connection: "+con);
		}
		else
		System.out.println("connection is not established");
		st = con.createStatement();
	ResultSet rs = st.executeQuery("select * from empblob");
		while(rs.next()){
			int eno = rs.getInt(1);
			System.out.println("eno: "+eno);
			InputStream is = rs.getBinaryStream(2);
FileOutputStream fos = new FileOutputStream("myphoto.jpg");
			int i = is.read();
			while(i !=-1){
				fos.write(i);
				i = is.read();
			}
		}

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

```
}
```

How to work with multiple image files (BLOB) :

```
package jdbc;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;
public class JdbcDemo {
      public static void main(String[] args)
      throws ClassNotFoundException,SQLException,
      IOException{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con = DriverManager.getConnection(
                        "jdbc:oracle:thin:@localhost:1521:xe", "system",
                        "manager");
            System.out.println("con: "+con);
            PreparedStatement ps =
            con.prepareStatement("insert into studentsbaj
values(?,?,?)");
            Scanner scan = new Scanner(System.in);
            System.out.println("enter student number");
            int sid = scan.nextInt();
            System.out.println("enter student name");
            String sname = scan.next();
            System.out.println("enter student image path");
            String sphoto = scan.next();

            File f = new File(sphoto);
            FileInputStream fis = new FileInputStream(f);
            ps.setInt(1, sid);
            ps.setString(2, sname);
            ps.setBinaryStream(3, fis, (int)f.length());
            int count = ps.executeUpdate();
            System.out.println(count+".record inserted");
```

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

              while(rs.next()){
                  int sid1 = rs.getInt(1);
                  String sname1 = rs.getString(2);
                  System.out.println(sid1+"\t"+sname1);
                  InputStream is = rs.getBinaryStream(3);
                  FileOutputStream fos =
      new
FileOutputStream("C:\\Users\\Ramchandar\\Desktop\\"+sname1+".jpg");
                  int i = 0;
                  while((i= is.read())!=-1){
                        fos.write(i);
                  }
            }
      }
}
```


How to convert image format from one type to another type:

Create blob table in db:

Create table empblob (eno number(5), ephoto blob);

Commit;

```
import java.awt.image.BufferedImage;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.imageio.ImageIO;

import jdbc.aj.ram.ConnectionFactory;
public class JdbcProgram9{
      public static void main(String[] args)
                  throws SQLException, IOException{
            Connection con = ConnectionFactory.getConnect();
            System.out.println(con);
            /*PreparedStatement ps =
               con.prepareCall("insert into empblob values(?,?)");
            Scanner scan = new Scanner(System.in);
            System.out.println("enter employee number");
            int eno = scan.nextInt();
            File f = new File("mypic.gif");
            FileInputStream fis = new FileInputStream(f);
            ps.setInt(1, eno);
            ps.setBinaryStream(2, fis,(int)f.length());
            int count = ps.executeUpdate();
            System.out.println(count);
            */
            Statement st = con.createStatement();
            ResultSet rs = st.executeQuery("select * from empblob");
            while(rs.next()){
                  System.out.println(rs.getInt(1));
                  InputStream is = rs.getBinaryStream(2);
                  BufferedImage img =ImageIO.read(is);
                  FileOutputStream fos =
                              new FileOutputStream("emp-
"+rs.getInt(1)+".png");
                  ImageIO.write(img, "png", fos);
                  System.out.println("image created");
            }
      }
}
```

```
import java.awt.image.BufferedImage;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
```

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import javax.imageio.ImageIO;
import jdbc.aj.ram.ConnectionFactory;
public class JdbcProgram9{
	public static void main(String[] args) throws SQLException,
IOException{
		Connection con = ConnectionFactory.getConnect();
		System.out.println(con);
	/*PreparedStatement ps = con.prepareCall("insert into
empblob values(?,?)");
		Scanner scan = new Scanner(System.in);
		System.out.println("enter employee number");
		int eno = scan.nextInt();
		File f = new File("anni 012.jpg");
		FileInputStream fis = new FileInputStream(f);
		ps.setInt(1, eno);
		ps.setBinaryStream(2, fis,(int)f.length());


		int count = ps.executeUpdate();
		System.out.println(count);*/




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

		while(rs.next()){
			System.out.println(rs.getInt(1));

			InputStream is = rs.getBinaryStream(2);
			BufferedImage img =ImageIO.read(is);
			FileOutputStream fos = new
FileOutputStream("mypic15.bmp");

			ImageIO.write(img, "bmp", fos);

			System.out.println("image created");

		}
```

```
	}
}
```

ImageIO is one final class is used to read and write image information within the less time.
BufferedImage is used to store the image information.

How to insert text file into DataBase:

```
import java.io.File;
import java.io.FileReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.Statement;
import java.util.Scanner;

public class Demo1 {

		public static void main(String[] args){
			Connection con =null;
			Statement st = null;
			try{
				con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
			if(con != null){
				System.out.println("connection is
established");
				System.out.println("connection: "+con);
			}
			else
				System.out.println("connection is not
established");

				PreparedStatement ps =
con.prepareStatement("insert into empclob values(?,?)");
				File f = new File("MyFile");
				FileReader fr = new FileReader(f);
				Scanner scan = new Scanner(System.in);
				System.out.println("enter eno ");
				int eno = scan.nextInt();
				ps.setInt(1, eno);
				ps.setCharacterStream(2, fr, (int)f.length());
```

```
                        int i = ps.executeUpdate();
                        System.out.println(i +" inserted");
                    }
                        catch(Exception e){
                            e.printStackTrace();
                        }

            }
}
```

How read the text file From Database:

```
import java.io.FileWriter;
import java.io.Reader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class Demo1 {

            public static void main(String[] args){
                Connection con =null;
                Statement st = null;
                try{
                con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
                if(con != null){
                System.out.println("connection is established");
                    System.out.println("connection: "+con);
                }
                else
                System.out.println("connection is not established");
                    st = con.createStatement();
            ResultSet rs = st.executeQuery("select * from empclob");
                    while(rs.next()){
                        int eno = rs.getInt(1);
                        System.out.println("eno: "+eno);
                        Reader r = rs.getCharacterStream(2);
                    FileWriter fw = new FileWriter("ramfile.doc");
                        int i=0;
                        i=r.read();
```

```
                    while(i!=-1){
                        fw.write(i);
                        i=r.read();
                    }

                fw.flush();

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

    }
}
```

How to insert multiple file into database in jdbc:

```
package jdbc;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Scanner;
public class JdbcDemo {
    public static void main(String[] args)
    throws ClassNotFoundException,SQLException,
    IOException{
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection con = DriverManager.getConnection(
                "jdbc:oracle:thin:@localhost:1521:xe", "system",
                "manager");
        System.out.println("con: "+con);
        PreparedStatement ps =
        con.prepareStatement("insert into course values(?,?,?)");
        Scanner scan = new Scanner(System.in);
        System.out.println("How many records do you want to
insert");
        int records = scan.nextInt();
        int count = 0;
        for(int i=1;i<=records;i++){
            System.out.println("ENTER COURSE NAME");
            String cname = scan.next();
```

```
                System.out.println("ENTER FACULTY NAME");
                String fname = scan.next();
                System.out.println("ENTER COURSE DETAILS FILE PATH");
                String cfpath = scan.next();
                ps.setString(1,cname);
                ps.setString(2, fname);
                File f = new File(cfpath);
                FileReader fr = new FileReader(f);

                ps.setCharacterStream(3, fr, (int)f.length());
                count = count+ps.executeUpdate();

            }
            System.out.println(count+".records inserted");
        }
}
```

How to work with Array type of data in JDBC:

create type mytype1 as VARRAY(3) of NUMBER;
/
Type created.

SQL> create table mytype1tab (eno number,marks mytype1);
Table created.

SQL> commit;
Commit complete.

SQL> insert into mytype1tab values(101,mytype1(11,22,33));
1 row created.

SQL> select * from mytype1tab;

ENO	MARKS
101	MYTYPE1(11, 22, 33)

SQL> select * from mytype1tab;

ENO	MARKS
101	MYTYPE1(11, 22, 33)
102	MYTYPE1(55, 66, 77)

Inserting array type of data in db in jdbc:

```
import java.io.IOException;
import java.sql.BatchUpdateException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

import ram.factory.ConnectionFactory;
public class Demo {
	public static void main(String[] args) throws Exception {
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe"
,"system","manager");
		System.out.println("con: "+con);
		Statement st  = con.createStatement();
		int count = st.executeUpdate("insert into mytype1tab
values(102,mytype1(55,66,77))");
		System.out.println("count: "+count);
	}
}
```

Reading the Array type of data from jdbc:

```
import java.sql.Array;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
public class Demo {
	public static void main(String[] args) throws Exception {
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","sys
tem","manager");
		System.out.println("con: "+con);
		Statement st  = con.createStatement();
		ResultSet rs = st.executeQuery("select * from mytype1tab");
		while(rs.next()){

			int eno = rs.getInt(1);
			System.out.println("eno: "+eno);
			Array arr = rs.getArray(2);
```

```
                System.out.println("arr: "+arr);
                ResultSet rs1 = arr.getResultSet();
                while(rs1.next()){
                     System.out.println("arrr: "+rs1.getInt(2));
                }
           }

     }
}
```

How to insert ARRAY type data db by using preaparedstatement:

```
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class Demo {
     public static void main(String[] args) throws
ClassNotFoundException, SQLException, IOException {
           Class.forName("oracle.jdbc.driver.OracleDriver");
           Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","sys
tem","manager");
           System.out.println("con: "+con);
           PreparedStatement ps = con.prepareStatement("insert into rt
values(?,?)");
           ps.setInt(1, 222);
           oracle.sql.ArrayDescriptor ad =
oracle.sql.ArrayDescriptor.createDescriptor("MYTYPE", con);
     //MYTYPE must be in capital letters.
           oracle.sql.ARRAY a = new oracle.sql.ARRAY(ad, con, new
Integer[]{11,22,33});
           ps.setArray(2, a);
           int count = ps.executeUpdate();
           System.out.println("count: "+count);
           }


}
```

```
package jdbc;

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

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

public class JdbcDemo {
	public static void main(String[] args)
			throws SQLException{
		Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","sys
tem","manager");
		System.out.println(con);

		PreparedStatement ps = con.prepareStatement(
			"insert into sajtype values(?,?)");
		Scanner scan = new Scanner(System.in);
		System.out.println("Ener student identification number");
		int sid = scan.nextInt();
		ps.setInt(1, sid);

		oracle.sql.ArrayDescriptor ad =

	oracle.sql.ArrayDescriptor.createDescriptor("AJTYPE",con);

		int ia[] = new int[3];
		System.out.println("enter student marks");

		for(int i=0;i<3;i++){
			ia[i] = scan.nextInt();
		}
		oracle.sql.ARRAY a = new oracle.sql.ARRAY(ad, con, ia);

		ps.setArray(2, a);

		int count= ps.executeUpdate();
		System.out.println("count: "+count);

		}
}
```

How to create object type data in oracle:

```
SQL> create type myobj as OBJECT (cityname varchar2(10),
statename varchar2(10));
  /
Type created.
SQL> create table myobjtab (eno number,addr myobj);
Table created.
SQL> commit;
Commit complete.
SQL> insert into myobjtab values(101,myobj('hyd','tel'));
1 row created.
SQL> select * from myobjtab;

    ENO   ADDR(CITYNAME, STATENAME)
    101   MYOBJ('hyd', 'tel')
```

How to insert object type of data in jdbc:

```
import java.sql.Array;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
public class Demo {
     public static void main(String[] args) throws Exception {
          Class.forName("oracle.jdbc.driver.OracleDriver");
          Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","sys
tem","manager");
          System.out.println("con: "+con);
          Statement st  = con.createStatement();
          int count = st.executeUpdate("insert into myobjtab
values(101,myobj('vizag','ap'))");
          System.out.println("count: "+count);

     }
}
```

How to read object type of data from jdbc:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.sql.Struct;
public class Demo {
	public static void main(String[] args) throws Exception {
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","sys
tem","manager");
		System.out.println("con: "+con);
		Statement st  = con.createStatement();
		ResultSet rs = st.executeQuery("select * from myobjtab");
		while(rs.next()){
			int eno= rs.getInt(1);
			System.out.println("eno: "+eno);
			Struct addr = (Struct) rs.getObject(2);
			Object[] obj = addr.getAttributes();
			for(Object obj1: obj){
				System.out.println("obj1: "+obj1);
			}
		}

	}
}
```

How to Create Connection Factory in JDBC:

```
import java.io.FileInputStream;
import java.sql.Connection;
import java.sql.Driver;
import java.sql.SQLException;
import java.util.Properties;
public class ConnectionFactory {
	private Properties myProperties;
	private  Driver driver;
	private String driverName;
	private String url;
	private String userName;
	private String password;
	private FileInputStream fis;
	private static ConnectionFactory connectionFactory=null;
	ConnectionFactory(){
		try{
```

```
            fis = new FileInputStream("MyProperties.properties");
                myProperties = new Properties();
                myProperties.load(fis);
                driverName = myProperties.getProperty("driver");
        driver = (Driver)Class.forName(driverName).newInstance();
                url = myProperties.getProperty("url");
                userName = myProperties.getProperty("user");
                password = myProperties.getProperty("password");
            }catch(Exception e){
                e.printStackTrace();
throw new RuntimeException("problem in connection
development");

            }
        }
        public static Connection getConnection() throws
SQLException{
            if(connectionFactory == null)
                connectionFactory = new ConnectionFactory();
return connectionFactory.driver.connect(connectionFactory.url,
connectionFactory.myProperties);
        }
}
```

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class TestDemo {
        public static void main(String[] s) throws SQLException{
            Connection con = ConnectionFactory.getConnection();
            if(con!=null){
                System.out.println("con: "+con);
                System.out.println("connection established");
            }
            else
                System.out.println("connection not established");
            Statement st = con.createStatement();
            ResultSet rs = st.executeQuery("select * from empram");
            while(rs.next()){
        System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename
")+"\t"+rs.getInt("sal")+"\t"+rs.getString("dept"));
            }
        }
}
```

Types Of ResultSet:

1. Forward ResultSet(TYPES_FORWARD_ONLY)
2. Scrollable ResultSet(
   TYPES_SCROLL_SENSITIVE
   TYPES_SCROLL_INSENSITIVE
)
3. Updateable ResultSet.

TYPES_FORWARD_ONLY ResultSet is suitable for reading the data from the table from top to bottom, but not suitable for reading the data from the table from bottom to top.

If we want to read the from top to bottom as well as bottom to top, we should go for either TYPES_SCROLL_SENSITIVE or TYPES_SCROLL_INSENSITIVE result set.

TYPES_SCROLL_SENSITIVE ResultSet can have the capability to know whether the database is updated or not where as TYPES_SCROLL_INSENSITIVE ResultSet can't have that capability.

How to create SCROLL_SENSITIVE ResultSet:

```
Statement st =
 con.createStatement(ResulSet.TYPE_SCROLL_SENSITIVE)
```

How to create SCROLL_INSENSITIVE ResultSet:

```
Statement st =
con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE)
```

How to create Read only ResultSet:

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

How to create updateable ResultSet:

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

How to update row in database by Updtable ResultSet object:

There are two ways to update the data in the table.

a. By using query.
   Update sbaj2_4 set sname='sailu' where sno=102;

b. By using updateRow().

ResultSet object must be ResultSet.Concur.UPDATABLE

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


public class TestDemo {
	public static void main(String[] s) throws SQLException{
		Connection con = ConnectionFactory.getConnection();
		if(con!=null){
			System.out.println("con: "+con);
			System.out.println("connection established");
		}
		else
			System.out.println("connection not established");
		Statement st =
con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.
CONCUR_UPDATABLE);
		ResultSet rs = st.executeQuery("select
eno,ename,sal,dept from empram");
		while(rs.next()){

	System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename
")+"\t"+rs.getInt("sal")+"\t"+rs.getString("dept"));
		}
		rs.first();
		rs.updateInt(3, 2500);
		rs.updateRow();

	System.out.println("========================");
		rs.beforeFirst();
		while(rs.next()){

	System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename
")+"\t"+rs.getInt("sal")+"\t"+rs.getString("dept"));
		}
```

```
	}
}
```

From the java application if we want to update the data which is available in database in two ways.

a. by writing bellow query

```
update sbajemp set esal=5000 where eid=104;
```

b. by using updateInt(-,-).

updateInt(-,-) logic itself generate query for updating the data in db.

Whenever we use updateInt(-,-) for updating done use * symbol for reading the data

Like select * from sbajemp (invalid)

We need to write the following manner

Select eid,ename,esal,dept from sbajemp;

How to print the data from database in both direction by using ResultSet Object:

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


public class TestDemo {
	public static void main(String[] s) throws SQLException{
		Connection con = ConnectionFactory.getConnection();
		if(con!=null){
			System.out.println("con: "+con);
			System.out.println("connection established");
		}
		else
			System.out.println("connection not established");
		Statement st =
con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.
CONCUR_READ_ONLY);
```

```
            ResultSet rs = st.executeQuery("select
eno,ename,sal,dept from empram");
            while(rs.next()){

      System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename
")+"\t"+rs.getInt("sal")+"\t"+rs.getString("dept"));
            }

      System.out.println("==========================");
            while(rs.previous()){

      System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename
")+"\t"+rs.getInt("sal")+"\t"+rs.getString("dept"));
            }
      }
}
```

Whenever we use relative(), first we need to check whether ResultSet object cursor position, the cursor shouldn't in beforeFirst and afterLast position.

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

public class TestDemo {
      public static void main(String[] s) throws SQLException{
            Connection con = ConnectionFactory.getConnection();
            if(con!=null){
                  System.out.println("con: "+con);
                  System.out.println("connection established");
            }
            else
                  System.out.println("connection not established");
            Statement st =
con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.
CONCUR_READ_ONLY);
            ResultSet rs = st.executeQuery("select
eno,ename,sal,dept from empram");
            while(rs.next()){

      System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename
")+"\t"+rs.getInt("sal")+"\t"+rs.getString("dept"));
```

```
            }

        System.out.println("============================");
            while(rs.previous()){

        System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename
")+"\t"+rs.getInt("sal")+"\t"+rs.getString("dept"));
            }
            rs.first();
            rs.relative(2);

        System.out.println("============================");

        System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename
")+"\t"+rs.getInt("sal")+"\t"+rs.getString("dept"));
            rs.last();
            rs.relative(-1);

        System.out.println("============================");

        System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename
")+"\t"+rs.getInt("sal")+"\t"+rs.getString("dept"));
        }
}
```

How to delete the record from database by using Updateable ResultSet object:

We can delete the data from database from our java application in two ways.

a. By using following query.
   Delete from sbajemp where eid = 107;

b. By using deleteRow().
   deleteRow() itself writing the query for delete record from db.

   For reading purpose don't we use * symbol, we need to write column names.

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

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;


public class TestDemo {
	public static void main(String[] s) throws SQLException{
		Connection con = ConnectionFactory.getConnection();
		if(con!=null){
			System.out.println("con: "+con);
			System.out.println("connection established");
		}
		else
			System.out.println("connection not established");
		Statement st =
con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.
CONCUR_UPDATABLE);
		ResultSet rs = st.executeQuery("select
eno,ename,sal,dept from empram");
		while(rs.next()){

	System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename
")+"\t"+rs.getInt("sal")+"\t"+rs.getString("dept"));
		}

	System.out.println("============================");
		while(rs.previous()){

	System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename
")+"\t"+rs.getInt("sal")+"\t"+rs.getString("dept"));
		}
		rs.first();
		rs.relative(2);

	System.out.println("============================");

	System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename
")+"\t"+rs.getInt("sal")+"\t"+rs.getString("dept"));
		rs.last();
		rs.relative(-1);

	System.out.println("============================");
```

```
		System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename
")+"\t"+rs.getInt("sal")+"\t"+rs.getString("dept"));
			rs.last();
			rs.deleteRow();
			rs.beforeFirst();

		System.out.println("============================");
			while(rs.next()){

		System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename
")+"\t"+rs.getInt("sal")+"\t"+rs.getString("dept"));
			}
		}
}
```

How to insert the record into database by using ResultSet Object:

we can insert record into database in two ways from our java applicaton.

a. By using the following query
   Insert into sbajemp values(103,'uma',9500,'hr');

b. By using insertRow().

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


public class TestDemo {
	public static void main(String[] s) throws SQLException{
		Connection con = ConnectionFactory.getConnection();
		if(con!=null){
			System.out.println("con: "+con);
			System.out.println("connection established");
		}
		else
			System.out.println("connection not established");
		Statement st =
con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.
CONCUR_UPDATABLE);
```

```
            ResultSet rs = st.executeQuery("select
eno,ename,sal,dept from empram");
            while(rs.next()){

      System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename
")+"\t"+rs.getInt("sal")+"\t"+rs.getString("dept"));
            }

      System.out.println("============================");
            while(rs.previous()){

      System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename
")+"\t"+rs.getInt("sal")+"\t"+rs.getString("dept"));
            }
            rs.first();
            rs.relative(2);

      System.out.println("============================");

      System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename
")+"\t"+rs.getInt("sal")+"\t"+rs.getString("dept"));
            rs.last();
            rs.relative(-1);

      System.out.println("============================");

      System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename
")+"\t"+rs.getInt("sal")+"\t"+rs.getString("dept"));
            rs.last();
            rs.deleteRow();
            rs.beforeFirst();

      System.out.println("============================");
            while(rs.next()){

      System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename
")+"\t"+rs.getInt("sal")+"\t"+rs.getString("dept"));
            }
            rs.moveToInsertRow();
            rs.updateInt(1, 104);
            rs.updateString(2, "pritam");
            rs.updateInt(3, 5000);
            rs.updateString(4, "faculty");
            rs.insertRow();
```

```
            rs.beforeFirst();

        System.out.println("============================");
            while(rs.next()){

        System.out.println(rs.getInt("eno")+"\t"+rs.getString("ename
")+"\t"+rs.getInt("sal")+"\t"+rs.getString("dept"));
            }
        }
}
```

Methods in ResultSet (updateable):

```
Statement st =
con.createStatement(
ResultSet.TYPE_SCROLL_SENSITIVE,
ResutlSet.CONCUR_UPDATABLE);
ResultSet rs = st.executeQuery
        ("select eid,ename,esal,dept from sbajemp");
```

rs.last() : cursor pointing to last record of the table.
rs.first() : cursor pointing to first record of the table.
rs.next() : forward the controle from non-record area to record area.
rs.beforeFirst(): cursor is pointing to non-record area which is avilable before first record.
rs.afterLast() : cursor is pointing to non-record area which is avilable after last record.
rs.moveToInsertRow(): it is used for move our cursor for inserting new record.
rs.insertRow(): used to place the record into table.
rs.updateXxx(): used to update old data with new data.
rs.deleteRow(): used to delete the record from the table.

DataBaseMetaData:

It is one JDBC API object, used for getting the details about database name, version, driver name, version, database username etc.....

```
public class Demo1 {

		public static void main(String[] args){
			Connection con =null;
			Statement st = null;
			try{
				con =
DriverManager.getConnection("jdbc:oracle:thin:@lenovo-
PC:1521:xe","system","manager");
			if(con != null){
				System.out.println("connection is
established");
				System.out.println("connection: "+con);
			}
			else
				System.out.println("connection is not
established");
				DatabaseMetaData dbmd = con.getMetaData();


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

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


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

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

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

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

Connection Pooling:

As we know that opening and closing connection to data base is very costlier job and it requires more resources and time consume.

This is will effect to performance of the system.

As a programmer, we want to reduce the number of hits to data base for opening and closing the connection.

<u>Connection Pool:</u>

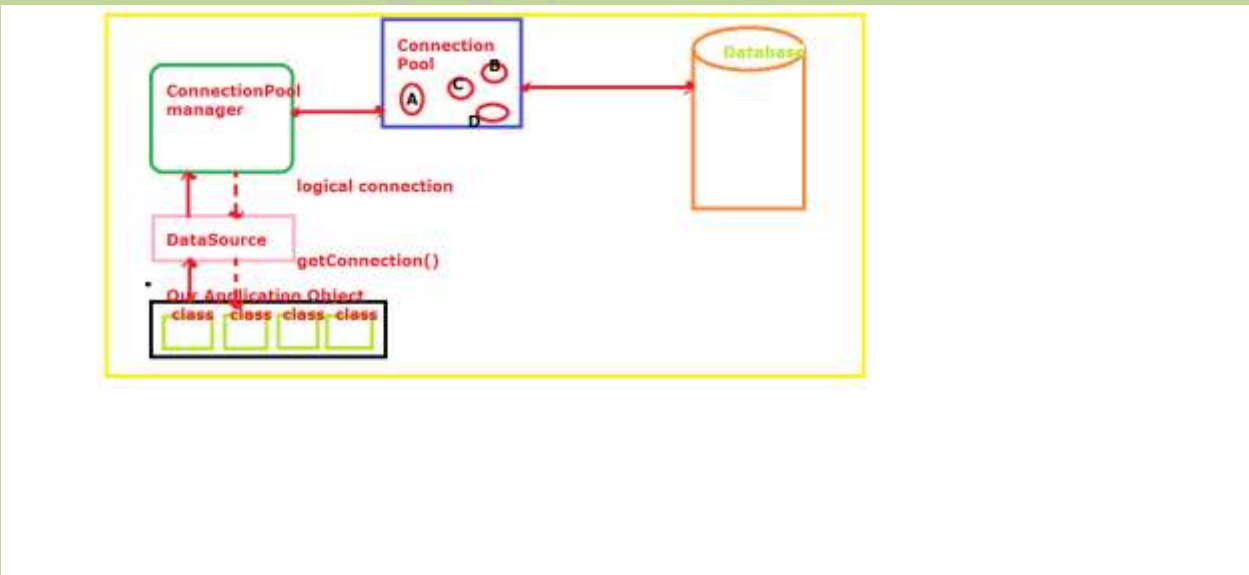
It is a collection of database connections managed to be available for application objects.

The system, which manages this set of connections objects is known as Connection pool Manager.

To develop connection pooling we required jdbc 2.0 api.
It includes on abstraction, in the form of one abstraction in the form javax.sql.DataSource to

getConnection() of Database allows us to collect the Connection from connection pool through connection pool manager.

This connection object obtained by the application in the case is a logical connection.




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

```
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import javax.sql.PooledConnection;

import oracle.jdbc.pool.OracleConnectionPoolDataSource;
public class Demo1 {

		public static void main(String[] args) throws
SQLException{
			//Creating Connection Pool Manager
			OracleConnectionPoolDataSource ocp = new
OracleConnectionPoolDataSource();
	//Setting properties to pool manager to create db connections
	//Driver name not required the reason is
//OracleConnectionPoolDataSource is designed for OracleDriver
			ocp.setURL("jdbc:oracle:thin:@localhost:1521:xe");
			ocp.setUser("system");
			ocp.setPassword("manager");

			//getting the pooled connection
			PooledConnection pc = ocp.getPooledConnection();

			//get the logical connection
			Connection con = pc.getConnection();

			Statement st = con.createStatement();

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

			while(rs.next()){
		System.out.println(rs.getInt(1)+"...."+rs.getString(2));
			}
	}
}
```

How to communicate with mysql database:

If we want to communicate with different databases from our application we required multiple logics.
These logics are different from one database to another database.
So java is not providing the logics to communicating with databases.

Database vendors itself provides logic to communicating or provides services to java.
But database vendors not using their own specification, all database vendors must be followed by the specifications which are given by the java.

Following the specification is nothing but provides the logic or implementation.
Database vendors provides the logic or implementation in the form of jar files like bellow

Oracle Vendors provides the following driver like:
ojdbc6.jar:

Driver→racle.jdbc.driver.OracleDriver
url: → jdbc:oracle:thin:@localhost:1521:xe
username→ system
password → manger

mysql-connector.jar

Driver→com.mysql.jdbc.Driver
url→ jdbc:mysql://localhost:3306/ram
username:→root
password: →root

To work with mysql database we need to the following steps:

1. **First install mysql database in our machine.**
2. **Developing program jdbc program.**
3. **Set classpath with mysql-connector.jar file**
4. **Create our own database in mysql.**
5. **Use our own database**
6. **Create table to do the curd operations.**

How create our own database in mysql:

Syntax: Create database database_name;

Create database ram;

How to use our own database:

Syntax: use database_name;

```
use ram;
```

How to create table in mysql:

```
Create table mytab(eno int(5) , ename varchar(10));
Commit;
```

How to insert the data into our own table:

```
Insert into mytab values(101, 'ram');
```

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

public class JDBCDemoProg1 {
	public static void main(String[] args)throws
ClassNotFoundException,SQLException{
		Class.forName("com.mysql.jdbc.Driver");
		Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:3306/ram1",
"root","root");
		if(con != null){
			System.out.println("con: "+con);
			System.out.println("connection established");
		}
		else
			System.out.println("connection not established");
		Statement st = con.createStatement();
		ResultSet rs = st.executeQuery("select * from mytab");
		while(rs.next()){

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

		int i = st.executeUpdate("insert into mytab
values(102,'varun')");
		System.out.println("i: "+i);
		con.close();
	}
```

```
}
```

How work with both the database from the same program:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Scanner;
public class Demo {
	public static void main(String[] s)
	throws ClassNotFoundException,SQLException{
		Scanner scan = new Scanner(System.in);
		System.out.println("enter driver name");
		String driver = scan.next();
		System.out.println("enter url");
		String url = scan.next();
		System.out.println("enter username");
		String user = scan.next();
		System.out.println("enter password");
		String password = scan.next();
		Class.forName(driver);
	Connection con = DriverManager.getConnection(url,user,password);
		System.out.println("con: "+con);
		Statement st = con.createStatement();
		ResultSet rs = st.executeQuery("select * from sbajtab");
		System.out.println("SID\tSNAME\tSAGE");
		while(rs.next()){
			System.out.println(rs.getInt(1)+"\t"+rs.getString(2)
			+"\t"+rs.getInt(3));
		}

	}
}
```

How insert Date related information int database:

Create table in database like bellow.

create table sbajkit(sno number(5), sname varchar2(15),sage number(3),jd Date);

QueryDemo.java:

```
package kit.aj.date;

public class QueryDemo {
     static final String INSERTQUERY = "insert into sbajkit
values(?,?,?,?)";
}
```

In the bellow program we create Date object of util package and collect the current system date and timings by using following code
Java.util.Date d1 = new java.util.Date();
Later we convert this information into one long value like d1.getTime();

Later we are creating Date object of sql package and sending this long value to constructor of Date class of sql package.

This constructor will convert that entire value into the following format
"year-month-date".

JdbcDateDemo:

```
package kit.aj.date;

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

import jdbc.aj.ram.ConnectionFactory;

public class JdbcDateDemo {
     public static void main(String[] args)
     throws SQLException{
          Connection con = ConnectionFactory.getConnect();
          System.out.println("con: "+con);
          PreparedStatement ps =
con.prepareStatement(QueryDemo.INSERTQUERY);
          Scanner scan = new Scanner(System.in);
          System.out.println("enter student number");
          int sno = scan.nextInt();
```

```
            System.out.println("enter student Name");
            String sname = scan.next();

            System.out.println("enter student age:");
            int sage = scan.nextInt();

            java.util.Date d1 = new java.util.Date();
            System.out.println(d1);

            java.sql.Date d2 = new java.sql.Date(d1.getTime());

            ps.setInt(1,sno);
            ps.setString(2, sname);
            ps.setInt(3, sage);
            ps.setDate(4, d2);
            int updatecount = ps.executeUpdate();
            System.out.println("updatecount: "+updatecount);
      }
}
package jdbc;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Date;
import java.util.Scanner;

public class JdbcDemo {
      public static void main(String[] args)
      throws ClassNotFoundException,SQLException{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            Connection con = DriverManager.getConnection(

      "jdbc:oracle:thin:@localhost:1521:xe","system","manager");
            System.out.println("con: "+con);
            Statement st = con.createStatement();
            boolean b = st.execute(
            "create table emptable (eid number, ename varchar2(15),ejd
date)");
            System.out.println("table created: "+b);
            Date d = new Date();

            PreparedStatement ps = con.prepareStatement(
```

```
                    "insert into emptable values(?,?,?)");
            Scanner scan = new Scanner(System.in);
            System.out.println("enter employee number");
            int eid = scan.nextInt();
            System.out.println("enter employee name");
            String ename = scan.next();
            ps.setInt(1, eid);
            ps.setString(2, ename);
            java.sql.Date sd = new java.sql.Date(d.getTime());
            ps.setDate(3, sd);
            int count = ps.executeUpdate();
            System.out.println(count+".record inserted");
      }
}
```

```
package kit.aj.date;

import java.sql.Connection;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Scanner;
import jdbc.aj.ram.ConnectionFactory;
public class JdbcDateDemo {
      public static void main(String[] args)
      throws SQLException{
            Connection con = ConnectionFactory.getConnect();
            System.out.println("con: "+con);
            PreparedStatement ps =
                        con.prepareStatement(QueryDemo.INSERTQUERY);

            Scanner scan = new Scanner(System.in);

            System.out.println("enter student number");
            int sno = scan.nextInt();

            System.out.println("enter student Name");
            String sname = scan.next();

            System.out.println("enter student age:");
            int sage = scan.nextInt();


            System.out.println("enter year");
            int year = scan.nextInt();
```

```
            System.out.println("enter month");
            byte month = scan.nextByte();
            System.out.println("enter day");
            byte day = scan.nextByte();
            Date d2 = new Date(year,month,day);
            ps.setInt(1,sno);
            ps.setString(2, sname);
            ps.setInt(3, sage);
            ps.setDate(4, d2);
            int updatecount = ps.executeUpdate();
            System.out.println("updatecount: "+updatecount);
      }
}
```

Date d2 = new Date(year,month,day);
Whenever we with above statement we need follow this rule
that is

actual year - 1900.

for example if we want enter 1997, we need to enter 97.
the reason whenever we enter year internally the above constructor add 1900 for the given year.

RowSet:

JdbcRowSet:

```
package jdbc;

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

import javax.sql.rowset.JdbcRowSet;
import javax.sql.rowset.RowSetProvider;

import oracle.jdbc.rowset.OracleJDBCRowSet;

public class SetDemo {
      public static void main(String[] args) throws Exception{
            Class.forName("oracle.jdbc.driver.OracleDriver");
            //JdbcRowSet js = new OracleJDBCRowSet();
            JdbcRowSet js =
RowSetProvider.newFactory().createJdbcRowSet();
            js.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
            js.setUsername("system");
            js.setPassword("manager");
            js.setCommand("insert into sbaj4to6
values(102,'sam',2000)");
            js.execute();
```

```
	}
}
```

CachedRowSet:

```
package jdbc;

import javax.sql.rowset.CachedRowSet;
import javax.sql.rowset.RowSetProvider;

import oracle.jdbc.rowset.OracleCachedRowSet;

public class SetDemo {
	public static void main(String[] args) throws Exception{
		Class.forName("oracle.jdbc.driver.OracleDriver");
		//JdbcRowSet js = new OracleJDBCRowSet();
		CachedRowSet js =
RowSetProvider.newFactory().createCachedRowSet();
		js.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
		js.setUsername("system");
		js.setPassword("manager");
		js.setCommand("insert into sbaj4to6
values(103,'mani',3000)");
		js.execute();

	}
}
```

WebRowSet:

```
package jdbc;

import java.io.File;
import java.io.FileWriter;

import oracle.jdbc.rowset.OracleWebRowSet;

public class SetDemo {
	public static void main(String[] args) throws Exception{
		Class.forName("oracle.jdbc.driver.OracleDriver");
		OracleWebRowSet js = new OracleWebRowSet();

		js.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
		js.setUsername("system");
```

```
            js.setPassword("manager");
            js.setCommand("select * from sbaj4to6");
            js.execute();

            File f = new File("D:\\jsp\\jdbc\\ors.xml");
            FileWriter fw = new FileWriter(f);
            js.writeXml(fw);
        }
    }
```