Standard Steps followed for developing JDBC(JDBC4.X) Application
=======================================================

1. Load and register the Driver
2. Establish the Connection b/w java application and database
3. Create a Statement Object
4. Send and execute the Query
5. Process the result from ResultSet
6. Close the Connection

Step1:
1. Load and register the Driver

A third party db vendor class which implements java.sql.Driver(I) is called as "Driver".

This class Object we need to create and register it with JRE to set up JDBC environment to run jdbc applications.

Note:

```
public class com.mysql.cj.jdbc.Driver extends
com.mysql.cj.jdbc.NonRegisteringDriver implements java.sql.Driver {
  public com.mysql.cj.jdbc.Driver() throws java.sql.SQLException;
  static {};
}
```

In MySQL Jar, Driver class is implementing java.sql.Driver, so Driver class Object should be created and it should be registered
to set up the JDBC environment inside JRE.

2. Establish the Connection b/w java application and database

```
   public static Connection getConnection(String url, String username,String password) throws SQLException;
   public static Connection getConnection(String url, Properties) throws SQLException;
   public static Connection getConnection(String url) throws SQLException;
```

The below creates the Object of Connection interface.

```
      Connection connection = DriverManager.getConnection(url,username,password);
                             |
                        getConnection(url,username,password) created an object of class which implements Connection(I)
                        that class object is collected by Connection(I).
                        This feature in java refers to
                                    a. Abstraction(hiding internal services)
                                    b. polymorphism(making code run in 1:M forms)
```

Can we create an Object for Interface?
 Answer. no
Can we create an Object for a class which implements interface?
Answer : yes

3. Create a Statement Object

```
    public abstract Statement createStatement() throws SQLException;
    public abstract Statement createStatement(int,int) throws SQLException;
    public abstract Statement createStatement(int,int,int) throws SQLException;

      Statement statement = connection.createStatement();
```

4. Send and execute the Query

## Query

From DB administrator perspective queries are classified into 5 types

1. DDL (Create table,alter table,drop table,..)
2. DML(Insert,update,delete)
3. DQL(select)
4. DCL(alter password,grant access)
5. TCL(commit,rollback,savepoint)

According to java developer perspective, we catergorise queires into 2 types

a. Select Query<br>
b. NonSelect Query

Methods for executing the Query are

a. executeQuery() => for select query we use this method.<br>
b. executeUpdate() => for insert,update and delete query we use this method.<br>
c. execute() => for both select and non-select query we use this method

```
public abstract ResultSet executeQuery(String sqlSelectQuery) throws SQLException;
      String sqlSelectQuery ="select sid,sname,sage,saddr from Student";
      ResultSet resultSet = statement.executeQuery(sqlSelectQuery);
```

5. Process the result from ResultSet

```
            public abstract boolean next() throws java.sql.SQLException;
                                                   |=> To check whether next Record
is available or not
                                                          returns true if available
otherwise returns false.

            System.out.println("SID\tSNAME\tSAGE\tSADDR");
            while(resultSet.next()){
                  Integer id = resultSet.getInt(1);
                  String name = resultSet.getString(2);
                  Integer age = resultSet.getInt(3);
                  String team = resultSet.getString(4);
                  System.out.println(id+"\t"+name+"\t"+age+"\t"+team);
            }
```

6. Close the Connection

EG#1

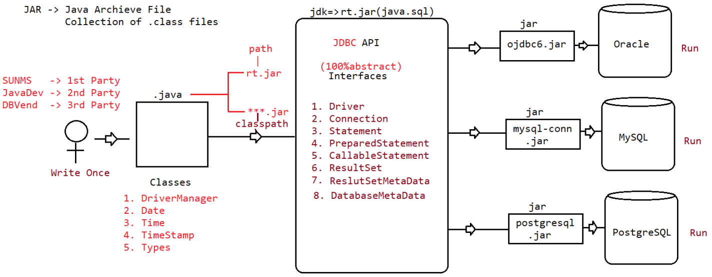
## Java code to communicate with database and execute select query

```
import com.mysql.cj.jdbc.Driver;
import java.sql.*;

class TestApp
{
      public static void main(String[] args)throws SQLException
      {
            //Step1. Load and register the Driver
            Driver driver = new Driver();//Creating driver object for MySQLDB
            DriverManager.registerDriver(driver);
            System.out.println("Driver registered succesfully");


            //Step2: Establish the connection b/w java and Database
            // JDBC URL SYNTAX:: <mainprotocol>:<subprotocol>:<subname>
            String url = "jdbc:mysql://localhost:3306/enterprisejavabatch";
            String username = "root";
            String password = "root123";

            Connection connection = 
DriverManager.getConnection(url,username,password);
            System.out.println("Connection object is created:: " + connection);

            // Create a Statement Object
            Statement statement = connection.createStatement();
            System.out.println("Statement object is created:: " + statement);

            //Sending and execute the Query
            String sqlSelectQuery ="select sid,sname,sage,saddr from Student";
            ResultSet resultSet = statement.executeQuery(sqlSelectQuery);
            System.out.println("ResultSet object is created:: " + resultSet);


            //Process the result from ResultSet
            System.out.println("SID\tSNAME\tSAGE\tSADDR");
            while(resultSet.next()){
                  Integer id = resultSet.getInt(1);
                  String name = resultSet.getString(2);
                  Integer age = resultSet.getInt(3);
                  String team = resultSet.getString(4);
                  System.out.println(id+"\t"+name+"\t"+age+"\t"+team);
            }

            //Close the Connection
            connection.close();
            System.out.println("Closing the connection...");

      }
}
```

Output

```
D:\JDBCPGMS>javac TestApp.java
D:\JDBCPGMS>java TestApp
Driver registered succesfully
Connection object is created:: com.mysql.cj.jdbc.ConnectionImpl@4e41089d
Statement object is created:: com.mysql.cj.jdbc.StatementImpl@23bb8443
ResultSet object is created:: com.mysql.cj.jdbc.result.ResultSetImpl@7364985f
SID     SNAME   SAGE    SADDR
7       dhoni   41      CSK
10      sachin  49      MI
18      kohli   35      RCB
45      rohith  37      MI
Closing the connection...
```

JAR -> Java Archieve File
       Collection of .class files



jdk=>rt.jar(java.sql)

JDBC API

(100%abstract)
Interfaces

1. Driver
2. Connection
3. Statement
4. PreparedStatement
5. CallableStatement
6. ResultSet
7. ReslutSetMetaData
8. DatabaseMetaData

SUNMS -> 1st Party
JavaDev -> 2nd Party
DBVend -> 3rd Party

Write Once

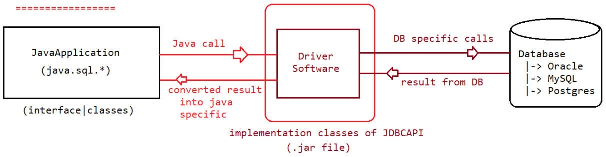
.java

path
rt.jar
***.jar
classpath

Classes
1. DriverManager
2. Date
3. Time
4. TimeStamp
5. Types

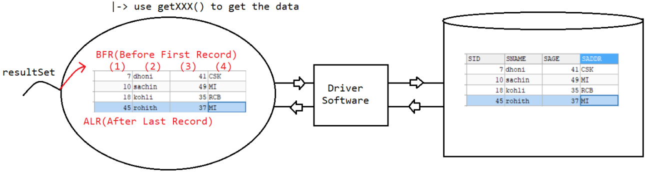
```
Path : It is an environmental variable associated with Operating System.
       These environmental variables are used to inform the location of .exe files
       of the softwares which needs to run from command prompt.
             set path =.........../bin
                                  |=> .exe files

ClassPath : It is an environmental variables associated with "Java Programs".
            Java developer uses this environmental variable and inform the jdk s/w
            to search for the required .class files(normally used in jdbc,servlet,hibernate,springmvc,..)
              set classpath=****.jar
                                |=> .class files
```

JDBC Architecture
=================



DriverSoftware
1. Type-1 driver
2. Type-2 driver
3. Type-3 driver
✡4. Type-4 driver

Enapsulation
   |-> use getXXX() to get the data



|  |  |  |  |
|---|---|---|---|
| 7 | dhoni | 41 | CSK |
| 10 | sachin | 49 | MI |
| 18 | kohli | 35 | RCB |
| 45 | rohith | 37 | MI |

| SID | SNAME | SAGE | SADDR |
|---|---|---|---|
| 7 | dhoni | 41 | CSK |
| 10 | sachin | 49 | MI |
| 18 | kohli | 35 | RCB |
| 45 | rohith | 37 | MI |

## Standard Steps followed for developing JDBC(JDBC4.X) Application

=======================================================

1. Load and register the Driver
2. Establish the Connection b/w java application and database
3. Create a Statement Object
4. Send and execute the Query
5. Process the result from ResultSet
6. Close the Connection

1. Load and register the Driver

In earlier version of JDBC 3.X we were loading and registering the driver using the following approach

```
            Driver driver = new Driver();
            DriverManager.registerDriver(driver);
```

Alternate to this we can also load the driver as shown below

```
      Class.forName("com.mysql.cj.jdbc.Driver");
```

```
class Driver{
      static{
            Driver driver = new Driver();
            DriverManager.registerDriver(driver);
      }
}
```

Note: We say a class represents a Driver, iff the class has implemented an interface called "java.sql.Driver(I)".

```
MySQL       => Driver(c) implements Driver(I)
Oracle        => OracleDriver(c) implements Driver(I)
Postgresql => PostgreSqlDriver(c) implements Driver(I)
```

From JDBC4.X onwards loading and registering would happen automatically depending upon the jar added in the classpath
location of the project.

Note:
1. JVM will search for the jar in the classpath
2. It will open the jar,move to META-INF folder
3. It will open services folder
4. It will search for java.sql.Driver file
5. Whatever value which is present inside Driver file that would be loaded automatically using

```
        Class.forName(value)
```

The above feature of JDBC4.X is called as "AutoLoading".

## Formatting the String query to accept the dynamic inputs

===============================================

```
int sage = scanner.nextInt();
String sname = scanner.next();
String saddr = scanner.next();

sname = " ' " + sname + " ' ";
saddr  =  " ' "    + saddr   + " ' ";
```

In DB specific query

```
   String =>  Varchar ===> ' '
   int       =>  int           ===> direct values
String query = "insert into student(`sname`,`sage`,`saddr`) values
("+sname+","+sage+","+saddr+")";
System.out.println(query);
```

To resolve the problem of the above approach we use a inbuilt class called "String".

```
int sage = scanner.nextInt();
String sname = scanner.next();
String saddr = scanner.next();

public static String format(String format, Object... args) {
        return new Formatter().format(format, args).toString();
 }
```

```
note:
String use format specifier as '%s'
int       use format specifier as %d
flaot   use format specifier as  %f
String query =String.format( "insert into student(`sname`,`sage`,`saddr`) values
('%s',%d,'%s')",sname,sage,saddr );
```

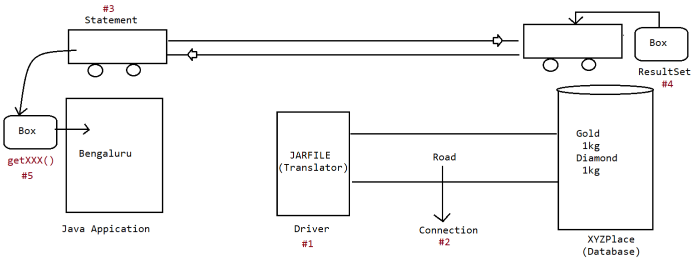
Through JDBC we have performed CRUD opeartion along with dynamic inputs from the user

```
Insert     => Create
Select    =>  Read
Update =>  Update
Delete   =>  Delete
```

Assignment
=========

Give the menu to the user as the operation listed below on student table

```
1. Create   2. Read    3. Update   4. Delete
```


#3
Statement
Box
ResultSet
#4
Box
getXXX()
#5
Bengaluru
Java Appication
JARFILE
(Translator)
Driver
#1
Road
Connection
#2
Gold
1kg
Diamond
1kg
XYZPlace
(Database)

Problem with statement Object
==========================

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

If we use Statement Object, same query will be compiled every time and the query should be executed everytime,this would
create performance problems.
eg: IRCTC App(select query), BMS APP(select query)

PreparedStatement Object
=======================

To resolve the above problem don't use Statement Object, use "PreparedStatement(Pre-CompiledStatement)".
In case of PreparedStatement, the query will be compiled only once eveythough we are executing it mulitple time with change
or no change in inputs. This would overall increase the performance.

signature

```
      public PreparedStatement prepareStatement(String sqlQuery) throws
SQLException
```

```
//Establish the connection
Connection con = DriverManger.getConnection(url,username,password);

//Creating a precompiled query which is used at the runtime to execute with the
value
String sqlSelectQuery = "select sid,sname,sage,saddr from student where sid = ?";
PreparedStatemetn pstmt = con.prepareStatement(sqlSelectQuery);
```

At this line, sqlquery will be sent to database, DatabaseEngine will compile the query and stores in database.
That precompiled query will be sent to the java application in the form of "PreparedStatement" Object.
Hence PreparedStatement Object is called "PreCompiledQuery" object.

```
// Execute the PreCompiledQuery by setting the inputs
Integer sid = 10;
pstmt.setInt(1,sid);
ResultSet resultSet = pstmt.executeQuery();
//process the resultSet


pstmt.setInt(1,100);
ResultSet resultSet = pstmt.executeQuery();
```

Whenever we execute methods, databasengine will not compile query once again and it will directly execute that query,
so that overall performance will be improved

Note:

```
String sqlQuery= insert into student(`sid`,`sname`,`sage`,`saddres`)
values(?,?,?,?);
PreparedStatement pstmt = con.prepareStatement(sqlQuery);

pstmt.setInt(1,10);
pstmt.setString(2,"sachin");
pstmt.setInt(3,45);
pstmt.setString(4,"MI");

int rowCount = pstmt.executeUpdate();
```

refer: PreparedStatementApp

## KeyPoints of methods

selectQuery => executeQuery()
nonSelectQuery => executeUpdate()
both select and nonSelect Query => execute()

## SQLInjection

users

| username | upwd |
|---|---|
| sachin | tendulkar |
| virat | kohli |

eg:
```
select count(*) from users where username ='"+uname+"'" and upwd =' "+upwd+"'";
      username = 'sachin'
      password = 'tendulkar'
```

Query nature
```
      select count(*) from users where username ='sachin' and upwd =' tendulkar' ";
            validation is succesful and given the authentication
```

eg:
```
select count(*) from users where username ='"+uname+"'" and upwd =' "+upwd+"'";
      username = 'sachin'--
      password = 'tendulkar'
```

Query nature
```
      select count(*) from users where username ='sachin'-- and upwd ='tendulkar' ";
            validation is succesfull and given the authentication
```

Note:
```
      1. -- Single line sql comment
      2. /*
            Multiline sql comment
        */
```

If we use Statement Object to send the Query, then the problem of SQLInjection will happen.

```
      eg: Statement stmt = con.createStatement();
             String query = "select count(*) from users where username ='"+uname+"'" and upwd =' "+upwd+"'";
             ResultSet resultSet =stmt.executeQuery(query);
                                    |
                                    |
         DB: select count(*) from users where username ='"+sachin'-- ";
                                    |
                              count(*) = 1 (validation is succesfull give authentication)
```

if we use PreparedStatement Object to send the Query, then the problem of SQLInjection will not happen.

```
      eg: String query = "select count(*) from users where username =? and upwd
=?";
            PreparedStatement pstmt = con.prepareStatement(query);
            pstmt.setString(1,"sachin'--");
            pstmt.setString(2,"tendulkar");
             ResultSet resultSet =pstmt.executeQuery();
                                    |
                                    | for compilation using PreparedStatement
                                    |
         DB: select count(*) from users where username =? and upwd =?;
                                    |
                                    |
             select count(*) from users where username ='sachin'--' and upwd
='tendulkar';
                                    |
                                 count(*) => 0 (validation not succesfull so no
authentication)
```

Note: In real time database used in production envrionment is "Oracle", only during development phase we use "MySQL" database.

In MySQLDatabase, we can't perform "SQLInjection" through comments,it happens only in "OracleDatabase".

eg:

```
   select * from users where userid = 1; (1 record will be pulled)
   select * from users where userid=  1 or 1=1;(All records in the table will be
pulled)
```

refer: PreparedStatementApp

How to handle Date object in Database?

Handling Date Values For Database Operations
=============================================

=> Sometimes as the part of programing requirement,we have to insert and retrieve Date like DOB,DOJ,DOM,DOP...wrt database.

=> It is not recommended to maintain date values in the form of String,b'z comparisons will become difficult.

In Java we have two Date classes

1. java.util.Date
2. java.sql.Date

=> java.sql.Date is the child class of java.util.Date.

=> java.sql.Date is specially designed class for handling Date values wrt database. Otherthan database operations,if we want to represent Date in our java program then we should go for java.util.Date.

=> java.util.Date can represent both Date and Time where as java.sql.Date represents only Date but not time.

```
class Test
{
   public static void main(String[] args)
   {
         java.util.Date udate=new java.util.Date();
         System.out.println("util Date:"+udate);
         long l =udate.getTime();
         java.sql.Date sdate= new java.sql.Date(l);
         System.out.println("sql Date:"+sdate);
    }
}
```

```
util Date:Mon Mar 20 19:07:29 IST 2017
sql Date:2017-03-20
```

Differences between java.util.Date and java.sql.Date

java.util.Date

1) It is general Utility Class to handle Dates in our Java Program.
2) It represents both Data and Time.

java.sql.Date

1) It is specially designed Class to handle Datesw.r.t DB Operations.
2) It represents only Date but not Time.

Note:
=> In sql package Time class is availble to represent Time values
=> In sql package TimeStamp class is available to represent both Date and Time.

-> Inserting Date Values into Database:
Various databases follow various styles to represent Date.

Eg:
Oracle: dd-MMM-yy  eg: 28-May-90
MySQL : yyyy-mm-dd eg: 1990-05-28

java.sql.Date => information is stored as "yyyy-mm-dd"

=> If we use simple Statement object to insert Date values then we should provide Date value in  the database supported format,which is difficult to the programmer.
=> If we use PreparedStatement,then we are not required to worry about database supported form,
just we have to call

```
pst.setDate (2, java.sql.Date);
```

This method internally converts date value into the database supported format.
Hence it is highly recommended to use PreparedStatement to insert Date values into database.

Steps to insert Date value into Database:
=> DB: create table users(name varchar2(10),dop date);
1. Read Date from the end user(in String form)

```
System.out.println("Enter DOP(dd-mm-yyyy):");
String dop=sc.next();
```

2. Convert date from String form to java.util.Date form by using SimpleDateFormat object.

```
SDF sdf= new SDF("dd-MM-yyyy");
java.util.Date udate=sdf.parse(dop);
```

3. convert date from java.util.Date to java.sql.Date

```
       long l = udate.getTime();
       java.sql.Date sdate=new java.sql.Date(l);
```

4. set sdate to query

```
       pst.setDate(2,sdate);
```

5. int rowAffected= pst.executeUpdate();//Execute the query.

```
UserInput => SimpleDateFormat====> java.util.Date => java.sql.Date =>
ps.setDate(1,date) =>DB
                  |-> parse()
```

Program To Demonstrate Inserting Date Values Into Database:
DB: create table users(name varchar2(10),dop date);

Note:
If end user provides Date in the form of "yyyy-MM-dd" then we can convert directly that String into java.sql.Date form as follows...
eg:

```
   String s = "1980-05-27";
   java.sql.Date sdate=java.sql.Date.valueOf(s);
```

Assignment1:
perform insertion opertion and also perform retrieval operation on the following data

```
      name    =>
      address=>
      gender  =>
      DOB      => dd-MM-yyyy
      DOJ       => MM-dd-yyyy
      DOM    => yyyy-MM-dd
```

Assignment2:
perform CRUD operation using preparedStatement
      1. insert 2. update 3. select 4. delete

Retrieving Date value from the database
========================================

=> For this we can use either simple Statement or PreparedStatement.
=> The retrieved Date values are Stored in ResultSet in the form of "java.sql.Date" and we can get this value by using getDate() method.
=> Once we got java.sql.Date object,we can format into our required form by using SimpleDateFormat object.

Sequence
========

1. Database

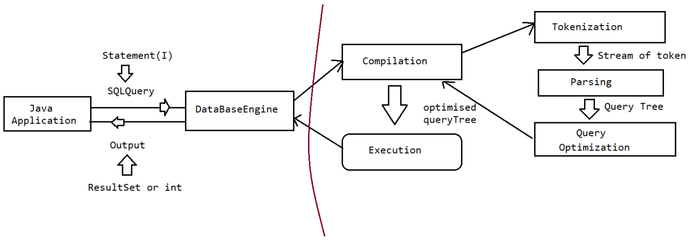
```
       (java.sql.Date)sqldate = rs.getDate(2);
```

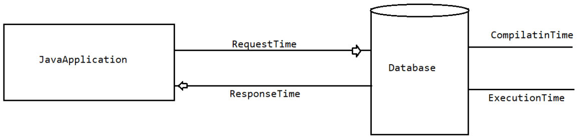
2. Our required String Form

```
       String s = sdf.format(sqldate);
```

3. String s holds the date.

refer: DateOpeartion





```
Total time per query = Req.Time + CompilationTime + ExecutionTime + ResponseTime
                         1ms     +   1ms            +    1ms         + 1ms
                     =  4ms

  per 1000Queries   = 4 * 1000    = 4000ms
```

Sometime in our application, we need same query but that query has to be executed multiple time with change in input.

IRCTC Application
  => book train ticket

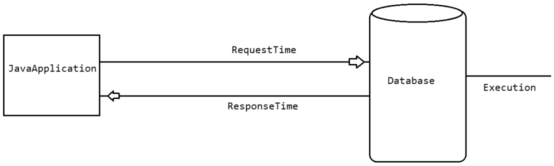
eg:   select * from trains where source = 'XXXX' and destination = 'YYYY'

Values changes as per the user

BMS Application
  => book tickets for movie based on theatre

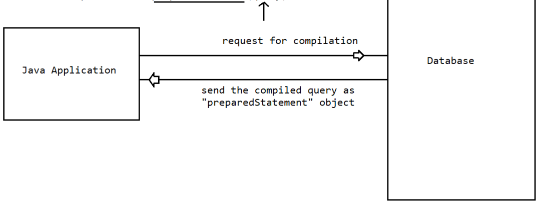
eg:    select * from theatres where city = 'XXXX' and movie = 'YYY'

Values changes as per the user



```
Total time taken per query =  RequestTime + ExecutionTime + ResponseTime
                                  1ms     +     1ms       +    1ms
                           =    3ms

 for 1000 queries   =  3ms * 1000 = 3000ms
```

```
String query ="select sid,sname,sage,saddr from student where sid = ?"
PreparedStatement pstmt = con.prepareStatement(query);
```

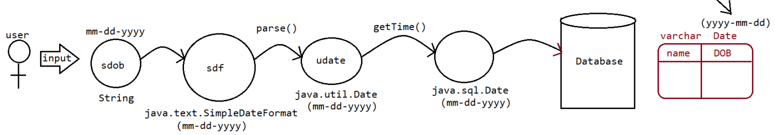


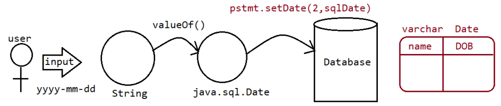
Note:

```
                                                                        1 2 3 4
String sqlQuery= insert into student(`sid`,`sname`,`sage`,`saddres`) values(?,?,?,?);
PreparedStatement pstmt = con.prepareStatement(sqlQuery);
                                                                       placeholder

pstmt.setInt(1,10);
pstmt.setString(2,"sachin");
pstmt.setInt(3,45);
pstmt.setString(4,"MI");

int rowCount = pstmt.executeUpdate();
```

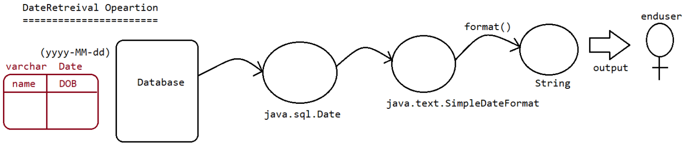
```
String query ="select sid,sname,sage,saddr from student where sid = ?'
```

Date Operation
==============





DateRetreival Opeartion
=======================



## Today's Agenda

BLOB,CLOB operation(PreparedStatement)
StoredProcedure(CallableStatement)
Connection Pooling(Servlet,Hiberante,SpringJDBC,SpringORM,SpringDataJpa)
Transaction
javax.sql.RowSet

## Working with Large Objects (BLOB And CLOB)

Sometimes as the part of programming requirement,we have to insert and retrieve large files like
images,video files,audio files,resume etc wrt database.
Eg:upload image in matrinomial web sites
     upload resume in job related web sites

To store and retrieve large information we should go for Large Objects(LOBs).
There are 2 types of Large Objects.
1. Binary Large Object (BLOB)
2. Character Large Object (CLOB)

1) Binary Large Object (BLOB)
    A BLOB is a collection of binary data stored as a single entity in the database.
    BLOB type objects can be images,video files,audio files etc..
    BLOB datatype can store maximum of "4GB" binary data.
            eg: sachin.jpg

2) CLOB (Character Large Objects):
    A CLOB is a collection of Character data stored as a single entity in the database.
    CLOB can be used to store large text documents(may plain text or xml documents)
    CLOB Type can store maximum of 4GB data.
            eg:  resume.txt

Steps to insert BLOB type into database:

1. create a table in the database which can accept BLOB type data.
```
        create table persons(name varchar2(10),image BLOB);
```
2. Represent image file in the form of Java File object.
```
        File f = new File("sachin.jpg");
```
3. Create FileInputStream to read binary data represented by image file
```
        FileInputStream fis = new FileInputStream(f)
```
4. Create PreparedStatement with insert query.
```
      PreparedStatement pst = con.prepareStatement("insert into persons values(?,?)");
```
5. Set values to positional parameters.
```
      pst.setString(1,"katrina");
```

To set values to BLOB datatype, we can use the following method: setBinaryStream()
```
public void setBinaryStream(int index,InputStream is)
public void setBinaryStream(int index,InputStream is,int length)
public void setBinaryStream(int index,InputStream is,long length)
```

6. execute sql query
```
      pst.executeUpdate();
```

Steps to Retrieve BLOB type from Database

==========================================

1. Prepare ResultSet object with BLOB type

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

2. Read Normal data from ResultSet

```
      String name=rs.getString(1);
```

3. Get InputStream to read binary data from ResultSet

```
      InputStream is = rs.getBinaryStream(2);
```

4. Prepare target resource to hold BLOB data by using FileOutputStream

```
       FileOutputStream fos = new FOS("katrina_new.jpg");
```

5. Read Binary Data from InputStream and write that Binary data to output Stream.

```
      int i=is.read();
      while(i!=-1)
      {
            fos.write(i);
            is.read();
      }

            or
      byte[] b= new byte[2048];
      while(is.read(b) > 0){
            fos.write(b);
        }
```

CLOB (Character Large Objects)

A CLOB is a collection of Character data stored as a single entity in the database.

CLOB can be used to store large text documents(may plain text or xml documents)

CLOB Type can store maximum of 4GB data.

Eg: resume.txt

Steps to insert CLOB type file in the database:
All steps are exactly same as BLOB, except the following differences
1. Instead of FileInputStream, we have to take FileReader.
2. Instead of setBinaryStream() method we have to use setCharacterStream() method.

```
public void setCharacterStream(int index,Reader r) throws SQLException
public void setCharacterStream(int index,Reader r,int length) throws SQLException
public void setCharacterStream(int index,Reader r,long length) throws SQLException
```

Retrieving CLOB Type from Database:
All steps are exactly same as BLOB, except the following differences..
1. Instead of using FileOutputStream,we have to use FileWriter
2. Instead of using getBinaryStream() method we have to use getCharacterStream() method

Q. What is the difference between BLOB and CLOB?
We can use BLOB Type to represent binary information like images, video files, audio files etc
Where as we can use CLOB Type to represent Character data like text file, xml file etc...

refer: BlobApp,ClobApp

Connection Pooling

================

=> If we required to communicate with database multiple times then it is not recommended to create separate Connection object every time, b'z creating and destroying Connection object every time creates performance problems.

=> To overcome this problem, we should go for Connection Pool.

=> Connection Pool is a pool of already created Connection objects which are ready to use.

=> If we want to communicate with database then we request Connection pool to provide Connection. Once we got the Connection, by using that we can communicates with database.

=>After completing our work, we can return Connection to the pool instead of destroying. Hence the main advantage of Connection Pool is we can reuse same Connection object multiple times, so that overall performance of application will be improved.

Process to implement Connection Pooling:

1. Creation of DataSource object
DataSource is responsible to manage connections in Connection Pool.
DataSource is an interface present in javax.sql package.
Driver Software vendor is responsible to provide implementation.
Oracle people provided implementation class name is :OracleConnectionPoolDataSource.
This class present inside oracle.jdbc.pool package and it is the part of ojdbc6.jar.

```
OracleConnectionPoolDataSource ds= new OracleConnectionPoolDataSource();
MySqlConnectionPoolDataSource  ds= new MySqlConnectionPoolDataSource();
```

2. Set required JDBC Properties to the DataSource object:

```
      ds.setURL("jdbc:oracle:thin:@localhost:1521:XE");
      ds.setUser("scott");
      ds.setPassword("tiger");
```

3. Get Connection from DataSource object:

```
      Connection con = ds.getConnection();
```

Once we got Connection object then remaining process is as usual.

Note:
This way of implementing Connection Pool is useful for Standalone applications. In the case of web and enterprise applications, we have to use server level connection pooling. Every web and application server can provide support for Connection Pooling.

Q. What is the difference Between getting Connection object by using DriverManager and DataSource object?
=> In the case of DriverManager.getConnection(), always a new Connection object will be created and returned.
=> But in the case of DataSourceObject.getConnection(), a new Connection object won't be created and existing Connection object will be returned from Connection Pool.

refer: ConnectionPoolingApp

StoredProcedure
==============

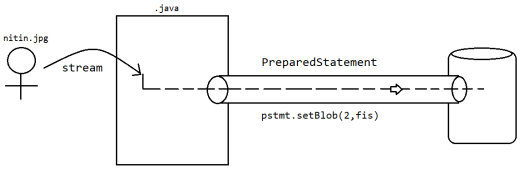
In our program,if we have any code which is repeatedly required, then we write that code inside function and we call that function mulitple times as per our needs.

so we say functions are reusablity component.

similary in database requirement, if we want set of sqlqueries which are repetadly used,then we write those set of statements in single group and we call that group based on our requirement.
This group of sql statements only we call as "StoredProcedure".

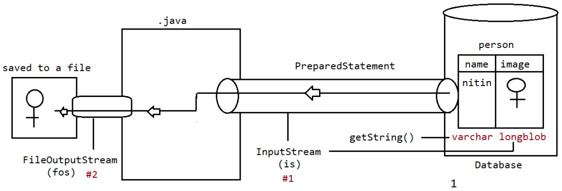
This storedprocedure is stored inside dbengine permanently and we need to just make a call to it.

refer: StoredProcedureApp

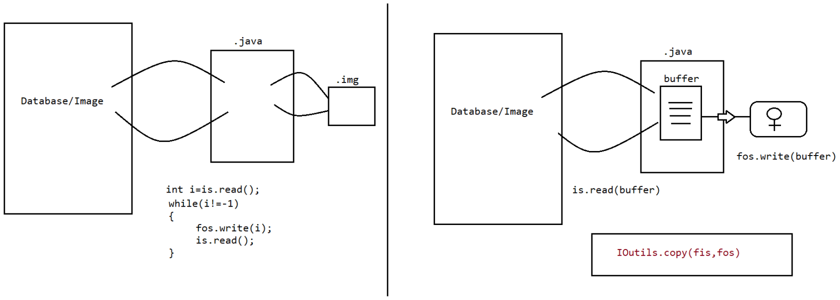


1,2

```
String sqlInsertQuery = "insert into person(`name`,`image`)values(?,?)";
```



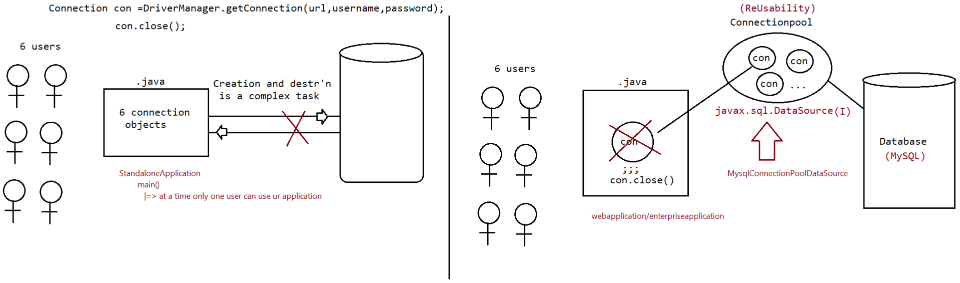
```
String sqlSelectQuery = "select name,image from person where name = ?";
```



```
int i=is.read();
while(i!=-1)
{
      fos.write(i);
      is.read();
}
```

```
IOutils.copy(fis,fos)
```

```
Connection con =DriverManager.getConnection(url,username,password);
         con.close();
```

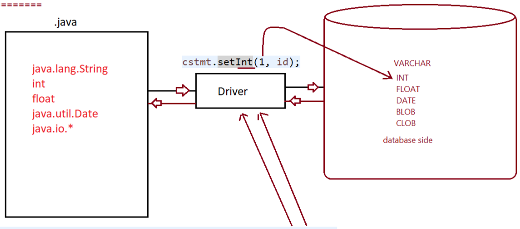


StandaloneApplication
main()
|=> at a time only one user can use ur application

MysqlConnectionPoolDataSource

webapplication/enterpriseapplication

CallableStatement(I)
===================



```
String storedProcedure = "{call getStudentsById(?,?,?)}";
```

2,3

```
cstmt.registerOutParameter(2, Types.VARCHAR);
cstmt.registerOutParameter(3, Types.VARCHAR);
```

informing the driver
do convert varchar output to
java.lang.String

JDBC Types helps in mapping the data
coming from database to java.

## Transaction Management in JDBC

=> Process of combining all related operations into a single unit and executing on the rule "either all or none", is called transaction management.

=> Hence transaction is a single unit of work and it will work on the rule "either all or none".

Case-1: Funds Transfer
1. debit funds from sender's account
2. credit funds into receiver's account

All operations should be performed as a single unit only. If debit from sender's account completed and credit into receiver's account fails then there may be a chance of data inconsistency problems.

Case-2: Movie Ticket Reservation
1. Verify the status
2. Reserve the tickets
3. Payment
4. issue tickets.

All operations should be performed as a single unit only. If some operations success and some operations fails then there may be data inconsistency problems.

Transaction Properties:
Every Transaction should follow the following four ACID properties.

1. A ➔ Atomiticity
   Either all operations should be done or None.
2. C ➔ Consistency(Reliabile Data)
   It ensures bringing database from one consistent state to another consistent state.
3. I ➔ isolation (Seperatation)
   Ensures that transaction is isolated from other transactions
4. D ➔ Durability
   It means once transaction committed, then the results are permanent even in the case of system restarts, errors etc.

## Types of Transactions

There are two types of Transactions
1. Local Transactions
2. Global Transactions

1. Local Transactions:
   All operations in a transaction are executed over same database.
   Eg: Funds transfer from one accoun to another account where both accounts in the same bank.
2. Global Transactions:
   All operations is a transaction are expected over different databases.
   Eg: Funds Transfer from one account to another account and accounts are related to different banks.

Note:
JDBC can provide support only for local transactions.
If we want global transactions then we have to go for EJB(Enterprise Java Bean) or Spring framework.

Process of Transaction Management in JDBC:

1. Disable auto commit mode of JDBC

By default auto commit mode is enabled. i.e after executing every sql query, the changes will   be committed automatically in the database.

We can disable auto commit mode as follows

```
con.setAutoCommit(false);
```

2. If all operations completed then we can commit the transaction by using the following method.

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

3. If any sql query fails then we have to rollback operations which are already completed by using rollback() method.

```
con.rollback();
```

Program to demonstrate Transaction app

Savepoint(I)
============

=> Savepoint is an interface present in java.sql package.

=> Introduced in JDBC 3.0 Version.

=> Driver Software Vendor is responsible to provide implementation.

=> Savepoint concept is applicable only in Transactions.

=> Within a transaction if we want to rollback a particular group of operations based on some condition then we should go for Savepoint.

=> We can set Savepoint by using setSavepoint() method of Connection interface.

```
Savepoint sp = con.setSavepoint();
```

=> To perform rollback operation for a particular group of operations wrt Savepoint, we can use rollback() method as follows.

```
con.rollback(sp);
```

=> We can release or delete Savepoint by using release Savepoint() method of Connection interface.

```
con.releaseSavepoint(sp);
```

```
con.setAutoCommit(false)
operation-1
operation-2
operation-3
SavePoint sp =new SavePoint();
operation-4
operation-5
      if(balance<=1000)
            con.rollback(sp);
      else
            con.releaseSavePoint();
operation-6
con.commit();
```

At line-1 if balance <10000 then operations 4 and 5 will be Rollback, otherwise all operations will be performed normally.

Note:

Some drivers won't provide support for Savepoint. Type-1 Driver won't provide support, but Type#4 Driver can provide support.

Type-4 Driver of Oracle provide support only for setSavepoint() and rollback() methods but not for releaseSavepoint() method.

Transaction Concurrency Problems:

Whenever multiple transactions are executing concurrently then there may be a chance of transaction concurrency problems.

The following are the most commonly occurred concurrency problems.

1. Dirty Read Problem
2. Non Repeatable Read Problem
3. Phantom Read Problem

1. Dirty Read Problem:

Also known as uncommitted dependency problem.

Before committing the transaction, if its intermediate results used by any other transaction then there may be a chance of Data inconsistency problems. This is called Dirty Read Problem.

```
nitin:50000
T1:update accounts set balance=balance+50000 where name='nitin'
T2:select balance from accounts where name='nitin'
T1: con.rollback();
```

At the end, T1 point of view, nitin has 50000 balance and T2 point of view nitin has 1Lakh. There may be a chance of data inconsistency problem. This is called Dirty Read Problem.

2. Non-Repeatable Read Problem:

For the same Read Operation, in the same transaction if we get different results at different times,then such type of problem is called Non-Repeatable Read Problem.

Eg:

```
T1: select * from employees;
T2: update employees set esal=10000 where ename='nitin';
T1: select * from employees;
```

In the above example Transaction-1 got different results at different times for the same query.

3. Phantom Read Problem:

A phantom read occurs when one transaction reads all the rows that satisfy a where condition and second transaction insert a new row that satisfy same where condition. If the first transaction reads for the same condition in the result an additional row will come.

This row is called phantom row and this problem is called phantom read problem.

```
T1: select * from employees where esal >5000;
T2: insert into employees values(300,'ravi',8000,'hyd');
T1: select * from employees where esal >5000;
```

In the above code whenever transaction-1 performing read operation second time, a new row will come in the result.

To overcome these problems we should go for Transaction isolation levels.

Connection interface defines the following 4 transaction isolation levels.
1. TRANSACTION_READ_UNCOMMITTED ➔ 1
2. TRANSACTION_READ_COMMITTED ➔ 2
3. TRANSACTION_REPEATABLE_READ ➔ 4
4. TRANSACTION_SERIALIZABLE ➔ 8

1. TRANSACTION_READ_UNCOMMITTED:
It is the lowest level of isolation.
Before committing the transaction its intermediate results can be used by other transactions.
Internally it won't use any locks.
It does not prevent Dirty Read Problem, Non-Repeatable Read Problem and Phantom Read Problem.
We can use this isolation level just to indicate database supports transactions.
This isolation level is not recommended to use.

2. TRANSACTION_READ_COMMITTED:
This isolation level ensures that only committed data can be read by other transactions.
It prevents Dirty Read Problem. But there may be a chance of Non Repeatable Read Problem and
Phantom Read Problem.

3. TRANSACTION_REPEATABLE_READ:
This is the default value for most of the databases. Internally the result of SQL Query will be locked for only one transaction. If we perform multiple read operations, then there is a guarantee that for same result.
It prevents Dirty Read Problem and Non Repeatable Read Problems. But still there may be a
chance of Phantom Read Problem.

4. TRANSACTION_SERIALIZABLE:
It is the highest level of isolation.
The total table will be locked for one transaction at a time.
It prevents Dirty Read, Non-Repeatable Read and Phantom Read Problems.
Not Recommended to use because it may creates performance problems.

Note:
Connection interface defines the following method to know isolation level.
getTransactionIsolation()
Connection interface defines the following method to set our own isolation level.
setTransactionIsolation(int level)

Eg:
```
System.out.println(con.getTransactionIsolation());
con.setTransactionIsolation(8);
System.out.println(con.getTransactionIsolation());
```

Note:
For Oracle database, the default isolation level is: 2(TRANSACTION_READ_COMMITED).
Oracle database provides support only for isolation levels 2 and 8.
For MySql database, the default isolation level is: 4(TRANSACTION_REPEATABLE_READ).
MySql database can provide support for all isolation levels (1, 2, 4 and 8).

Note:
ResultSet(holds the data which is used for reading purpose)
|=> Using resultset we have just performed read operation(best suited)
|=> Is it possible to perform update,inserte and delete operation(possible

but not recomended)

## RowSet(ALL DB vendors jar support for RowSet is not available)

=> It is alternative to ResultSet.

=> We can use RowSet to handle a group of records in more effective way than ResultSet.

=> RowSet interface present in javax.sql package

=> RowSet is child interface of ResultSet.

=> RowSet implementations will be provided by Java vendor and database vendor.

=> By default RowSet is scrollable and updatable.

=> By default RowSet is serializable and hence we can send RowSet object across the network. But ResultSet object is not serializable.

=> ResultSet is connected i.e to use ResultSet compulsary database Connection must be required.

=> RowSet is disconnected. ie to use RowSet database connection is not required.

## Types of RowSets

There are two types of RowSets

1. Connected RowSets
2. Disconnected RowSets

## Connected RowSets

Connected RowSets are just like ResultSets.

To access RowSet data compulsary connection should be available to database.

We cannot serialize Connected RowSets.

Eg: JdbcRowSet

Disconnected RowSets:

Without having Connection to the database we can access RowSet data.

We can serialize Disconnected RowSets.

Eg:

 CachedRowSet

 WebRowSet

   a.FilteredRowSet

   b.JoinRowSet

How to create RowSet objects?

We can create different types of RowSet objects as follows

```
      RowSetFactory rsf = RowSetProvider.newFactory();
      JdbcRowSet jrs = rsf.createJdbcRowSet();
      CachedRowSet crs = rsf.createCachedRowSet();
      WebRowSet wrs = rsf.createWebRowSet();
      JoinRowSet jnrs = rsf.createJoinRowSet();
      FilteredRowSet frs = rsf.createFilteredRowSet();
```

## JdbcRowSet

=> It is exactly same as ResultSet except that it is scrollable and updatable.

=> JdbcRowSet is connected and hence to access JdbcRowSet compulsary Connection must be required.

=> JdbcRowSet is non serializable and hence we cannot send RowSet object across the network.

Note:

```
      jdbcRowSet.setUrl("jdbc:mysql:///abc");
      jdbcRowSet.setUser("root");
      jdbcRowSet.setPassword("root123");
      jdbcRowSet.setCommand("select eid,ename,esal,eaddress from employee");
      jdbcRowSet.execute();
```

Application to demonstrate

1. Retreive records from jdbcRowSet
2. Insert records into jdbcRowSet
3. Update record  into jdbcRowSet
4. delete record  into jdbcRowSet

CachedRowSet:

=> It is the child interface of RowSet.
=> It is bydefault scrollable and updatable.
=> It is disconnected RowSet. ie we can use RowSet without having database connection.
=> It is Serializable.
=> The main advantage of CachedRowSet is we can send this RowSet object for multiple people
   across the network and all those people can access RowSet data without having DBConnection.
=> If we perform any update operations(like insert,delete and update) to the CachedRowSet,to
   reflect those changes compulsary Connection should be established.
=> Once Connection established then only those changes will be reflected in Database.

Application to demonstrate

1. Retreive records from CachedRowSet
2. Insert records from CachedRowSet
3. Update record from CachedRowSet
4. delete record from CachedRowSet

## Retreive a record

1. Use Connection Object and get Statement,resultSet object
2. Get CachedRowSet Object and populate(resultSet) into CachedRowSet
3. use CachedRowSet to retreive the records.

## Update record from CachedRowSet

Make sure get the Connection Object with autocommit as false.

1. crs.setTableName(tableName);
2. crs.populate(resultSet)
3. crs.absolute(rowNo)
4. crs.updateString(2,ename); crs.updateFloat(3,esal); crs.updateString(4,eaddr);
5. crs.updateRow()
6. crs.acceptChanges(connection)

## delete record from CachedRowSet

Make sure get the Connection Object with autocommit as false.

1. crs.setTableName(tableName);
2. crs.populate(resultSet)
3. crs.last();

```
   4. crs.deleteRow();
   5. crs.acceptChanges(connection)
```

## insert record into CachedRowSet

Make sure get the Connection Object with autocommit as false.

```
   1. crs.setTableName(tableName);
   2. crs.populate(resultSet)
   3. crs.moveToInsertRow();
   4. crs.updateNull(eid);//Autogenerated value
   5. crs.updateString(2,ename);crs.updateFloat(3,esal); crs.updateString(4,eaddr);
   6. crs.insertRow();
   7. crs.moveToCurrentRow();
   8. crs.acceptChanges(connection)
```

WebRowSet(I):
=> It is the child interface of CachedRowSet.
=> It is bydefault scrollable and updatable.
=> It is disconnected and serializable
=> WebRowSet can publish data to xml files,which are very helpful for enterprise applications.

```
      FileWriter fw=new FileWriter("emp.xml");
      rs.writeXml(fw);
```

=> We can read XML data into RowSet as follows

```
      FileReader fr=new FileReader("emp.xml");
      rs.readXml(fr);
```

## selecting the records

```
 1. rs.setCommand("select eid,ename,esal,eaddr from emp");
 2. rs.execute();
 3. FileWriter fw=new FileWriter("emp.xml");
 4.  rs.writeXml(fw);
 5.  rs.acceptChanges()
```

## inserting the records

```
 1. rs.setCommand("select eid,ename,esal,eaddr from emp");
 2. rs.execute();
 3. FileReader fr=new FileReader("input.xml");
 4.  rs.readXml(fr);
 5.  rs.acceptChanges()
```

## input.xml

```
<data>
   <insertRow>
      <columnValue>11</columnValue>
      <columnValue>dupples</columnValue>
      <columnValue>RCB</columnValue>
      <columnValue>45</columnValue>
   </insertRow>
</data>
```

## deleting the records

```
 1. rs.setCommand("select eid,ename,esal,eaddr from emp");
 2. rs.execute();
 3. FileReader fr=new FileReader("input.xml");
 4.  rs.readXml(fr);
 5.  rs.acceptChanges()
```

input.xml
=========

```
<data>
   <deleteRow>
      <columnValue>11</columnValue>
      <columnValue>dupples</columnValue>
      <columnValue>RCB</columnValue>
      <columnValue>45</columnValue>
   </deletRow>
</data>
```

JoinRowSet:
==========

=> It is the child interface of WebRowSet.

=> It is by default scrollable and updatable

=> It is disconnected and serializable

=> If we want to join rows from different rowsets into a single rowset based on matched column(common column) then we should go for JoinRowSet.

=> We can add RowSets to the JoinRowSet by using addRowSet() method.

```
    addRowSet(RowSet rs,int commonColumnIndex);
```

eg#1.

```
  CachedRowSet crs1=rsf.createCachedRowSet();
  crs1.setCommand("select sid,sname,saddr,cid from student");
  crs1.exeucte(con);

  CachedRowSet crs2=rsf.createCachedRowSet();
  crs2.setCommand("select cid,cname,cost from course");
  crs2.execute(con);

  JoinRowSet jrs=rsf.joinRowSet();
  rs.addRowSet(crs1,4);
  rs.addRowSet(crs2,1);

  //process the resultSet
```

FilteredRowSet(I):
=================

=> It is the child interface of WebRowSet.

=> If we want to filter rows based on some condition then we should go for FilteredRowSet.

```
      public interface FilteredRowSet{
            public boolean evaluate(RowSet rs);//for filtering logic
            public boolean evaluate(Object obj,int colIndex);//for insertion of record
            public boolean evaluate(Object obj,String colName);//for insertion of record
      }
```

Note:

```
public boolean evaluate(RowSet rs){
      try {
            String colValue = rs.getString(colName);
            if (colValue.startsWith(condValue)) {
                  return true;
            } else {
                  return false;
            }
            } catch (SQLException e) {
                  e.printStackTrace();
            }

       }
}
```

Behind the scenes
=================

for every rs.next(), the entire record will be pulled and it will be given to RowSet(rs)

so from RowSet(rs) we need to get the ColValue based on ColName.

check the colValue with our condValue,if it matches return true, if it is true then that particular row will be available in rowSet.if not that rowSet will not be availabe for rendering.