* Class.forName() :forName () is static method to load the given class ,its not create object of the class , only load the class whenever load the class static blocks are executed…..
* The process of saving and managing data long time is called persistence data.
* Curd operations to perform on that data
* Serialization: the process of sending object to file is called …. All about data converting in to bits and bytes. That bytes is stored in a file or can be send over the network is called
* De-Serialization: the process of reading data from a file and constructing new object
* Normaliazation is a put data into tabular form and removing redundant data ( అనవసరమై)
* The process of taking a database design, and apply a set of formal criteria and rules, is called Normal Forms.
* 1NF: In relation table all the columns are atomic.it means no repeating values …that repeating values convert in to another table and represent to the current table.
* 2NF: is in 1NF and The table must be already in 1 NF and all non-key columns of the tables must depend on the PRIMARY KEY

1. The partial dependencies are removed and placed in a separate table

Normalization link:<https://www.sqlshack.com/what-is-database-normalization-in-sql-server/>

* Drivers in jdbc:
* **Type**-1 **driver** or **JDBC**-ODBC bridge **driver**.
* **Type**-2 **driver** or Native-API **driver**.
* **Type**-3 **driver** or Network Protocol **driver**.
* **Type**-**4 driver** or Thin **driver**.

**Actually we are using thin driver it is fully java code** . and its directly connect without any odbc driver…… giving best performance. Thin drive directly contact with database. Pure java base driver… wont require any any extra component like odbc , native api and etc.

* Steps to create jdbc application:
  1. Registered jdbc driver with DriverManager service.
  2. Create connection with data base software.
  3. Create jdbc statement.
  4. Send and execute sql query.
  5. Gather sql result back to java app.
  6. Close jdbc objects.
* Connection con=DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe","system","abc123");
* urlservice user name pwd
* ExecuteQuery: execute query is using only select queries. Then we get Result set object to get values

Statement s=con.createStatement();

ResultSetr=s.executeQuery("select \* from students");// resulset is gathering the values

**while**(r.next()) {

System.***out***.println("sno "+r.getInt(1)+"\nname "+r.getString(2)+"\naddress "+r.getString(3));

* ExecuteUpdate(): Execute non select queries. It returns int no of records got effected.

**int**i=s.executeUpdate(" delete from emp where empno=7839");

**if**(i==0)

System.***out***.println(" deleted successfully");

**else**

System.***out***.println(" items not available");

* Execute(): with this method we can execute both select and non select queries.
* if it runs false execute non select queries AndgetUpdateCount to get numeric value how many record got effected.
* Effect if it returns true execute select queries and getResultSet object to store bunch of records

**boolean**b=s.execute("Select \* from students");

**if**(b==**false**) {

**int**i=s.getUpdateCount();

**if**(i==0)

System.***out***.println(" deleted successfully");

**else**

System.***out***.println(" items not available");

}**else** {

ResultSetr= s.getResultSet();

**while**(r.next()) {

System.***out***.println("sno "+r.getInt(1)+"\nname "+r.getString(2)+"\naddress "+r.getString(3));

} }

* Sql injection problem: Supplying special instruction along with input values like(--) and making them sql query behavior completely. Completely wrong result will come,

**publicclass** Example {

**publicstaticvoid**main(String[] args) {

Scanner sc= **new**Scanner(System.***in***);

System.***out***.println(" enter sno");

String sno=sc.nextLine();

System.***out***.println(" enter sname");

String sname=sc.nextLine();

System.***out***.println();

**try** {

**int**count=0;

sno="'"+sno+"'";

sname="'"+sname+"'";

String query="select count(\*) from Students where sno="+sno+" and"+" sname="+sname;

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

Connection con= DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe","system","abc123");

Statement s= con.createStatement();

ResultSetr= s.executeQuery(query);

**if**(r.next()) {

count=r.getInt(1);

}

**if**(count==0)

System.***out***.println("invalid credentials");

**else**

System.***out***.println(" valid credentials");

}**catch** (Exception e) {

e.printStackTrace();

}

}

Op: valid credentials

enter sno

11

enter sname

balu

valid credentials

Op:invalidcredentialsm n by using - - ( comment) will get login this is called sql injection

enter sno

11

enter sname

hyd' or(1=1)--

'hyd' or(1=1)--'

valid credentials

* Statement Objects
* Simple statement :

1. not suitable for executing same query multiple time with same or different values. Get performance issue
2. Input values are very complex
3. May raise sql injection
4. Does not allow parameter(?) in sql query to set values to the query later.
5. Does not allow large objects (files) in database table.

* Prepared Statement:

1. allows precompile and dynamic sql query
2. allows place holder in sql query(?)
3. does not raise sql injection
4. allows large object files also.
5. suitable for executing same query multiple time with same or different values. Getno performance issue

String query="select count(\*) from Students where sno=? and sname=?";

PreparedStatementp= con.prepareStatement(query);

p.setInt(1,Integer.*parseInt*(sno));

p.setString(2, sname);

ResultSetr= p.executeQuery();

Working with Date values:

* While developing with date values user is given date is converted as java.sql.Date
* We should use Prepared statement supported while using date

package com.naren;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import java.text.SimpleDateFormat;

import java.util.Scanner;

class DateEx {

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

try {

// actually we are getting string that is converted into sql by using these code

Scanner sc = new Scanner(System.in);

System.out.println(" enter date value dd-mm-yyyy");

String date = sc.nextLine();

// convert string date to sql date

SimpleDateFormat d = new SimpleDateFormat("dd-mm-yyyy");

java.util.Date dt = d.parse(date);// gives java.util.date

Long ms = dt.getTime();

java.sql.Date sdt = new java.sql.Date(ms);

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

Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system", "abc123");

PreparedStatement ps = con.prepareStatement("insert into students values(?,?,?,?)");

ps.setInt(1, 13);

ps.setString(2, "narendra dasara");

ps.setString(3, " tp");

ps.setDate(4, sdt);

int r = ps.executeUpdate();

if (r != 0)

System.out.println(" records inserted");

else

System.out.println(" records not inserted");

} catch (ClassNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (Exception e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

Working with large object files:

1. Blob : images , audio files , video files.. etc
2. Clob : text files , word docs ..etc

While working with photo we can wan write file object to locate the path of the photo. We use stream object to store data.

**package** com.nare;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**public** **class** Ex {

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

**try** {

File f= **new** File("D:\\NARENDRA.JPEG");

FileInputStream is=**new** FileInputStream(f);

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

Connection con = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "system", "abc123");

PreparedStatement ps = con.prepareStatement("insert into emp values(?,?)");

ps.setString(1, "narendra dasara");

ps.setBinaryStream(2,is,f.length());

**int** r = ps.executeUpdate();

**if** (r != 0)

System.***out***.println(" records inserted");

**else**

System.***out***.println(" records not inserted");

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

Download image from Database:

**package** com.nare;

**import** java.io.FileOutputStream;

**import** java.io.InputStream;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**public** **class** Ex {

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

**try** {

InputStream is=**null**;

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

Connection con = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "system", "abc123");

PreparedStatement ps = con.prepareStatement("select \* from emp where name=?");

ps.setString(1, "narendra dasara");

ResultSet r= ps.executeQuery();

**if** (r.next()) {

System.***out***.println(" photo downloaded");

is=r.getBinaryStream(2);

}

FileOutputStream os=**new** FileOutputStream("D:\\naren.gif");

**byte**[] b=**new** **byte**[4095];

**int** byteRead;

**while**((byteRead=is.read(b))!=-1){

os.write(b,0,byteRead);

}

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (Exception e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

**Same as like CLOB also**

*Understanding ResultSets:*

Result set represents set of records given by selectQuery

1) non Scrollable Resultset : allows us to read record sequentially and uni directionally.

Statement st =con.creteStatement();

ResultSet rs =st.executeQuery(“select \* from student”);

2) scrollable Resultset: allows us to read record non sequentially and randomly like first record or last record and get data fastly.

Statement st =con.creteStatement(type ,mode);

ResultSet rs =st.executeQuery(“select \* from student”);

To possible values for type:

ResultSet.TYPE\_SCROLLABLE\_SENSITIVE(1005)

ResultSet.TYPE\_SCROLLABLE\_iNSENSITIVE(1005)

To possible values for mode:

ResultSet.CONCUR\_READ\_ONLY(1005)

ResultSet.CONCUR\_UPDATABLE(1005)

*METADATA*

Data about data is called metadata

* + 1. DatabaseMetadata
    2. ResultSetMetaData
    3. ParameterMetaData
    4. DatabaseMetadata

1) DatabaseMetadata

Give limitations and underlyings on database software and also give some details of database software.

DatabaseMetaData dbmd=con.getMetaData();

Connection con = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:xe", "system", "abc123");

DatabaseMetaData db = con.getMetaData();

System.***out***.println(db.getDatabaseMajorVersion());

System.***out***.println(db.getSQLKeywords());

System.***out***.println(db.getProcedures(**null**, **null**, **null**));

System.***out***.println(db.getMaxColumnNameLength());

System.***out***.println(db.getDriverName());

2) ResultSetMetaData

Gives more info about table that is represented by resultset object like column names , column datatype etc.

PreparedStatement ps=con.prepareStatement("select \* from students");

ResultSet r=ps.executeQuery();

ResultSetMetaData rd=r.getMetaData();

System.***out***.println(rd.getColumnCount());

System.***out***.println(rd.getColumnClassName(1));

System.***out***.println(rd.getColumnTypeName(1));

System.***out***.println(rd.isSigned(1));

1. ParameterMetaData:

sIt is give more details of parameter that are there in sql query

PreparedStatement ps1=con.prepareStatement(" insert into students(sno,sname,sadd) values(?,?,?)");

ps1.setInt(1, 99);

ps1.setString(2,"some");

ps1.setString(3," some village");

**int** rs=ps1.executeUpdate();

ParameterMetaData p= ps.getParameterMetaData();

System.***out***.println(p.getParameterCount());

System.***out***.println();

BatchUpdation and Processing: