* 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");
* url service user name pwd
* ExecuteQuery: execute query is using only select queries. Then we get Result set object to get values

Statement s=con.createStatement();

ResultSet r=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 And getUpdateCount 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** {

ResultSet r= 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,

**public** **class** Example {

**public** **static** **void** 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();

ResultSet r= 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:invalid credentialsm 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. Get no performance issue

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

PreparedStatement p= con.prepareStatement(query);

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

p.setString(2, sname);

ResultSet r= p.executeQuery();