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
2) {
3)    public static void main(String[] args)
4)    {
5)          java.util.Date udate=new java.util.Date();
6)          System.out.println("util Date:"+udate);
7)          long l =udate.getTime();
8)          java.sql.Date sdate= new java.sql.Date(l);
9)          System.out.println("sql Date:"+sdate);
10)    }
11) }

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
```