```
.java                    .java                                      Any changes made in the
                                                    .properties      properties file will not be
;;;;;;;; | data would    ;;;;;;;;  write code   k1 = v1             having any impactto the java
;;;;;;;; | vary          ;;;;;;;;  to get the data  k2 = v2          code.so no need of
;;;;;;;; |               ;;;;;;;;  from properties   k3 = v3           a. making changes in src code
;;;;;;;; |               ;;;;;;;;     file.                            b. no need of re-compilation
     make change in src code                                          c. no need to re-run
     compile the code                      #2
     re-run the code            k=v      .load(fis)
                        properties          #3
                        java.util.Properties
        use getter method to get the
        value associated with key.                FileInputStream
                  #4                                    #1
```

Life cycle of Query Execcution   DB activity

```
                                                              Tokenization
                        SQLQuery                  Compilation           |
JavaApplication  ──────────────>  DatabaseEngine         |        Stream of Tokens
                                                         |              |
                 <──────────────                  optimized         Parsing
                   ResultSet                      query tree           |
                      or                                          Query Tree
                   int rowCount          Execution    QueryOptimization
```

1. Compilation
       a. Query Tokenization
       b. Query Parsing
       c. Query Optimization

2. Execution
       once compilation is successful the optimized query is taken as input and the query
       will be executed by the query intrepreter.

3. Fetch the result
       Result will be sent in the form of ResultSet(selectquery) or in the form of rowcount(nonselect query)
       to the java applicaton by the database engine.

Need of PreparedStatement

```
Statement stmt = con.createStatement();          <── one statement object can be used to run multiple queries.
ResultSet rs = stmt.executeQuery("select * from employees");
```

```
                          Request time                          Compile time
Java Application  ──────────────────────>   Database
                  <──────────────────────
                          Response time                        Execution time
```

```
        Total time per query = Req.Time + C.T + E.T + Resp.T
                             =  1ms     + 1ms + 1ms + 1ms
                             = 4ms

        For 1000 queries = 1000*4ms
                         = 4000ms
```

```
                                                    clients
IRCTC
    select * from trains where source = 'XXXX' and destination = 'XXXX'   ♀ ♀ ♀ ♀ ♀

                                               clients
BMS
    select * from theatres where city = 'XXXX' and movie ='XXXX';   ♀ ♀ ♀ ♀ ♀
```

    In the above context, same query is executed mulitple times with change in the input.

    If we use Statement object approach
        a. same query has to be compiled every time for every client request, this would
             create performance issue.

    To resolve this problem we need to use "PreparedStatement(I)".

    Advantage of using PreparedStatement
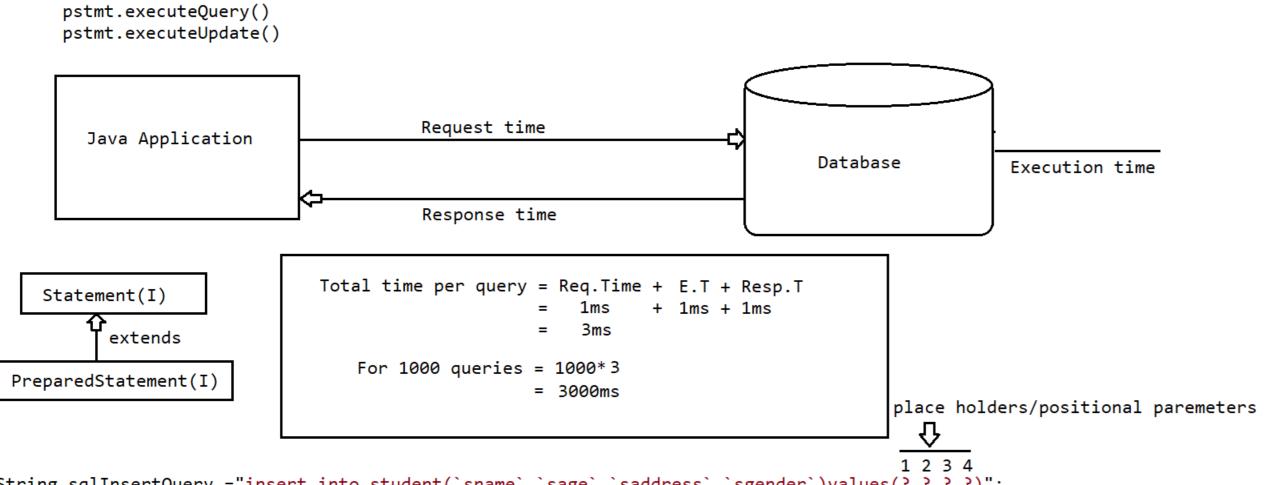   a. Query will be compiled only once even though we are executing in mulitple time with
           different inputs.
        b. through this performance is increased.

PreparedStatement prepareStatement(String query) throws SQLException;

    PreparedStatement pstmt = con.prepareStatement(sqlQuery)

    1. At this line query will be sent to database engine.
    2. DB engine will compile the query and stores in the database.
    3. The preCompiled query will be sent to the java application in the form of PreparedStatement Object.

    Note: PreparedStatement is also called as "PreCompiledQuery".

        pstmt.setXXXX(object,object);
        pstmt.executeQuery()
        pstmt.executeUpdate()

```
                          Request time
Java Application  ──────────────────────>   Database
                  <──────────────────────                        Execution time
                          Response time
```

```
Statement(I)            Total time per query = Req.Time + E.T + Resp.T
     ↑                                        =  1ms     + 1ms + 1ms
     | extends                                =  3ms
PreparedStatement(I)
                        For 1000 queries = 1000*3
                                         = 3000ms
```

```
                                                       place holders/positional paremeters
                                                                    ⇩
                                                                1 2 3 4
String sqlInsertQuery ="insert into student(`sname`,`sage`,`saddress`,`sgender`)values(?,?,?,?)";
PreparedStatement pstmt = con.prepareStatement(sqlInsertQuery); <──Query is compiled at DB level

    //query compiled ready for execution
    pstmt.setString(1,"apeksha");
    pstmt.setInt(2,26);                  |  using pstmt can we execute only one query
    pstmt.setString(3,"MI");             |  multiple times with change in the inputs.
    pstmt.setString(4,"F");              |

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