MODERN ACADEMY

TECHNOLOGY & FOR ENGINEERING

Computer Engineering Department

Convert NLP language to SQL Query

Language and Compilers

Name	Id	Sec
Abdelrahman Hussein Mohamed Abdelfatah Elshimy	4190033	3
Mohamed Hussein Abdeltwab	4190067	2
Youssef Mostafa Youssef Abdelfatah	4190364	3
Mohamed Samir Mohamed Mohamed	4190289	3
Mohamed Rashed Mostafa	4190432	2

Dr / Khaled Morsy

Content:

Phase 1

Lexical analysis:

Fetch all the information of the students who have scored more than 70 in 10th

↓

Lexical Analyzer

↓

[('fetch', 'NNP'), ('all', 'NNP') , ('the'), ('information'),('of'),('the') (' student', 'NNP'), ('who have', 'NNP'), ('scored'),('more', 'VBZ'), ('than') ,('70', 'CD'), ('in') , ('10th', 'NN')]

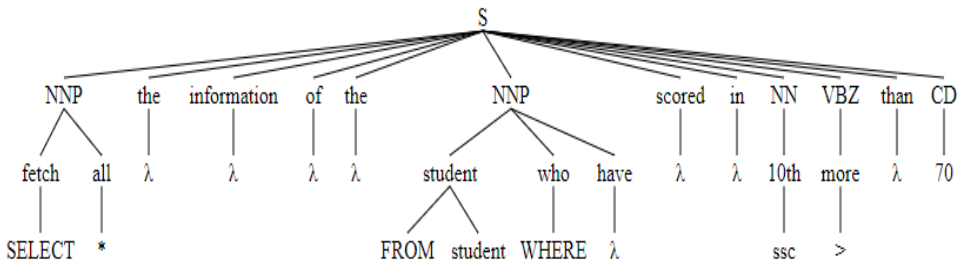
Syntax analysis:

[('fetch', 'NNP'), ('all', 'NNP') , ('the'), ('information'),('of'),('the') (' student', 'NNP'), ('who have', 'NNP'), ('scored'),('more', 'VBZ'), ('than') ,('70', 'CD'), ('in') , ('10th', 'NN')]

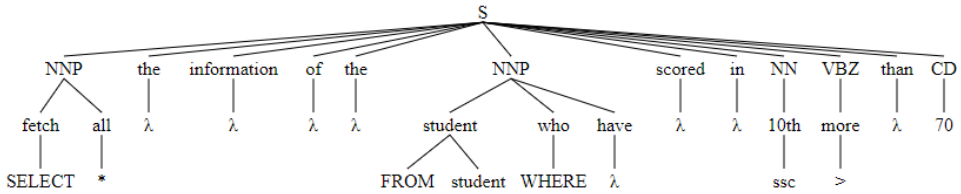
↓

Syntax Analyzer

↓



Intermediate representation & generation of SQL statements:



Intermediate code generator

['fetch', 'all', 'the', 'information', 'of', 'the', 'students', 'who', 'have', 'scored', 'more', 'than', '70', 'in', '10th']

['fetch', 'all', 'student', 'who', 'have', 'more', '70', '10th']

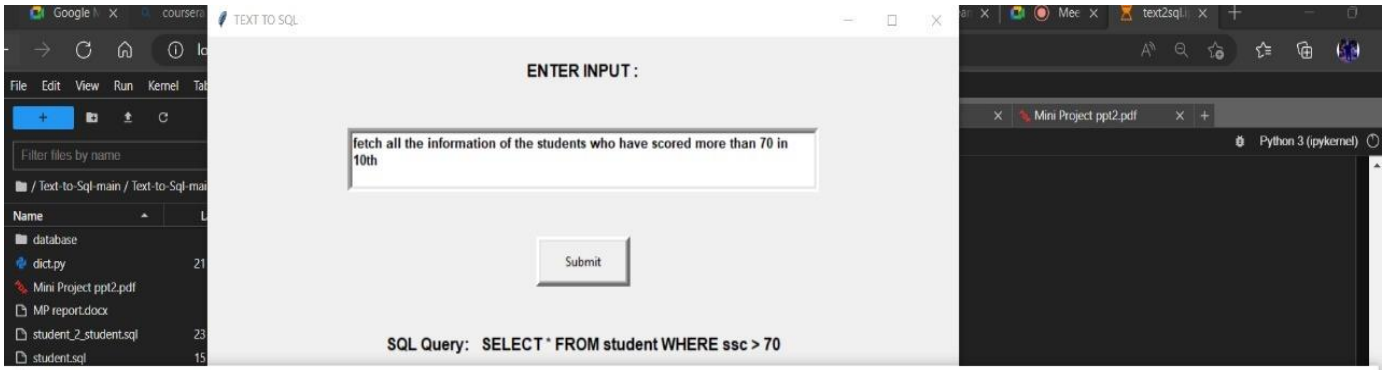
['SELECT', '*', 'FROM student', 'WHERE', 'more', '70', 'ssc']

['SELECT', '*', 'FROM student', 'WHERE', '>', '70', 'ssc']

Sample input and output queries as results:

```
['fetch', 'all', 'the', 'information', 'of', 'the', 'students', 'who', 'have', 'scored', 'more', 'than', '70', 'in', '10th']
['fetch', 'all', 'student', 'who', 'have', 'more', '70', '10th']
['SELECT', '*', 'FROM student', 'WHERE', 'more', '70', 'ssc']
['SELECT', '*', 'FROM student', 'WHERE', '>', '70', 'ssc']
string2
sentence
[('SELECT', 'NNP'), ('*', 'NNP'), ('FROM student', 'NNP'), ('WHERE', 'NNP'), ('>', 'VBZ'), ('70', 'CD'), ('ssc', 'NN')]
{'<action>': [], '<attr_name>': [], '<table_name>': [], '<condition_name>': [], '<condition>': [], '<value>': [], '<logic>': []}
['SELECT'] ['*'] ['FROM student'] ['ssc'] ['>'] ['70'] []

 SELECT * FROM student WHERE ssc > 70
```



1	18	rakesh	80	90	85
2	16	satyam	85	75	80
3	21	yogesh	90	70	80
6	22	simran	95	85	90
8	26	devesh	80	50	65
9	22	sharvil	75	65	70
10	23	shubham	80	80	80
19	23	rupesh	73	79	76
25	21	deepak	91	99	95
47	22	basit	80	50	45
84	22	harsh	89	68	65
86	19	sachin	80	90	55

Phase 2

Lexical analysis:

print id , name , age of students who age not more than twenty four and not less than eighteen

↓

Lexical Analyzer

↓

[('print', 'NNP'), ('id', 'NN'), ('name', 'NN'), ('age', 'NN'), (‘of’) ,('student', 'NNP'), ('who', 'NNP'), ('age', 'NN'), ('not', 'RB') , (‘more’) , (‘than’) , ('24', 'CD'), ('and’, 'NNP'), ('not', 'RB'),('less', 'VBD'), (‘than’) ,('18', 'CD')]

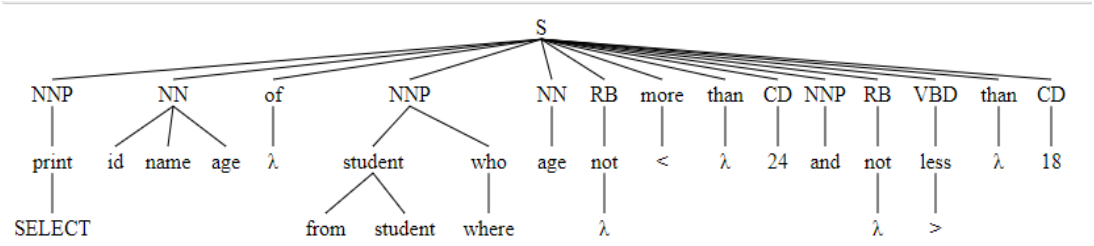
Syntax analysis:

[('print', 'NNP'), ('id', 'NN'), ('name', 'NN'), ('age', 'NN'), (‘of’) ,('student', 'NNP'), ('who', 'NNP'), ('age', 'NN'), ('not', 'RB') , (‘more’) , (‘than’) , ('24', 'CD'), ('and’, 'NNP'), ('not', 'RB'),('less', 'VBD'), (‘than’) ,('18', 'CD')]

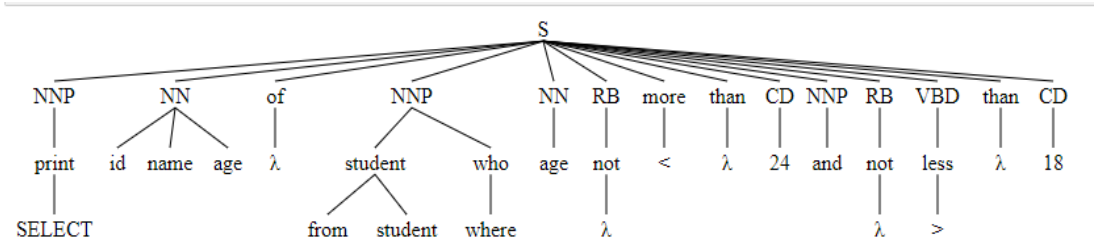
↓

Syntax Analyzer

↓



Intermediate representation & generation of SQL statements:



['print', 'id', ',', 'name', ',', 'age', 'of', 'students', 'who', 'name', 'not', 'more', 'than', 'twenty', 'four', 'and', 'not', 'less', 'than', 'eighteen']

['print', 'id', 'name', 'age', 'student', 'who', 'name', 'not', 'more', 'twenty', 'four', 'and', 'not', 'le', 'eighteen']

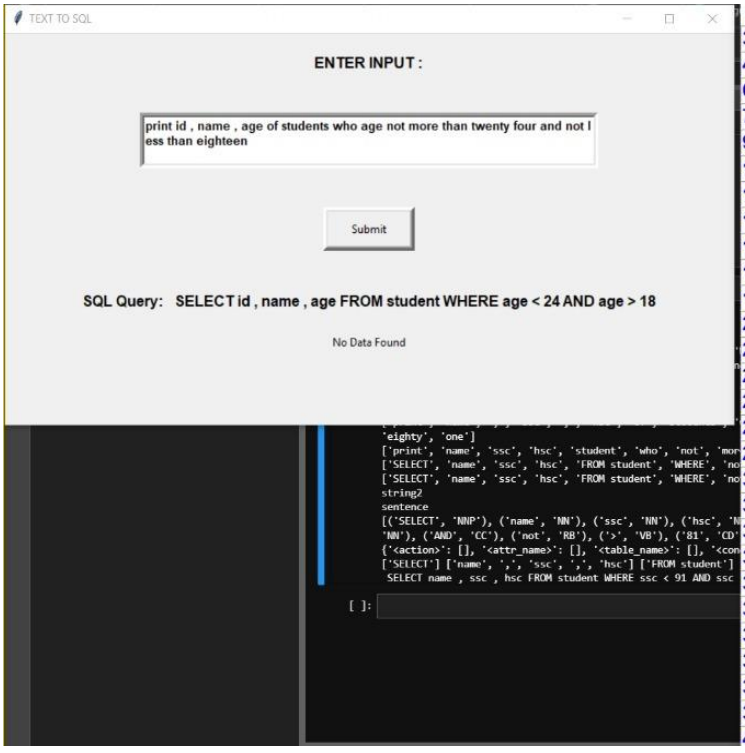
['SELECT', 'id', 'name', 'age', 'FROM student', 'WHERE', 'name', 'not', 'more', '20', '4', 'AND', 'not', 'le', '18']

['SELECT', 'id', 'name', 'age', 'FROM student', 'WHERE', 'name', 'not', '<', '24', 'AND', 'not', '>', '18']

Sample input and output queries as results:

```
['print', 'id', ',', 'name', ',', 'age', 'of', 'students', 'who', 'age', 'not', 'more', 'than', 'twenty', 'four', 'and', 'not', 'less', 'than', 'eighteen']
['print', 'id', 'name', 'age', 'student', 'who', 'age', 'not', 'more', 'twenty', 'four', 'and', 'not', 'le', 'eighteen']
['SELECT', 'id', 'name', 'age', 'FROM student', 'WHERE', 'age', 'not', 'more', '20', '4', 'AND', 'not', 'le', '18']

['SELECT', 'id', 'name', 'age', 'FROM student', 'WHERE', 'age', 'not', '<', '24', 'AND', 'not', '>', '18']
string2
sentence
[('SELECT', 'NNP'), ('id', 'NN'), ('name', 'NN'), ('age', 'NN'), ('FROM student', 'NNP'), ('WHERE', 'NNP'), ('age', 'NN'), ('not', 'RB'), ('<', 'JJ'), ('24', 'C
D'), ('AND', 'CC'), ('not', 'RB'), ('>', 'VB'), ('18', 'CD')]
{'<action>': [], '<attr_name>': [], '<table_name>': [], '<condition_name>': [], '<condition>': [], '<value>': [], '<logic>': []}
['SELECT'] ['id', ',', 'name', ',', 'age'] ['FROM student'] ['age'] ['<', '>'] ['24', '18'] ['AND']
 SELECT id , name , age FROM student WHERE age < 24 AND age > 18
```



Phase 3

Lexical analysis:

print name , ssc , hsc of students who scored not more than ninety one in 10th and not less than eighty

Lexical Analyzer

↓

[('print', 'NNP'), ('name', 'NN'), ('ssc', 'NN'), ('hsc', 'NN'), (‘of’) ,('student', 'NNP'), ('who', 'NNP'), (‘scored’), ('not', 'RB') , (‘more’) , (‘than’) , ('91', 'CD'), (‘in’) , (‘10th’) , ('and’, 'NNP'), ('not', 'RB'), ('less', 'VBD'), (‘than’) ,('81', 'CD')]

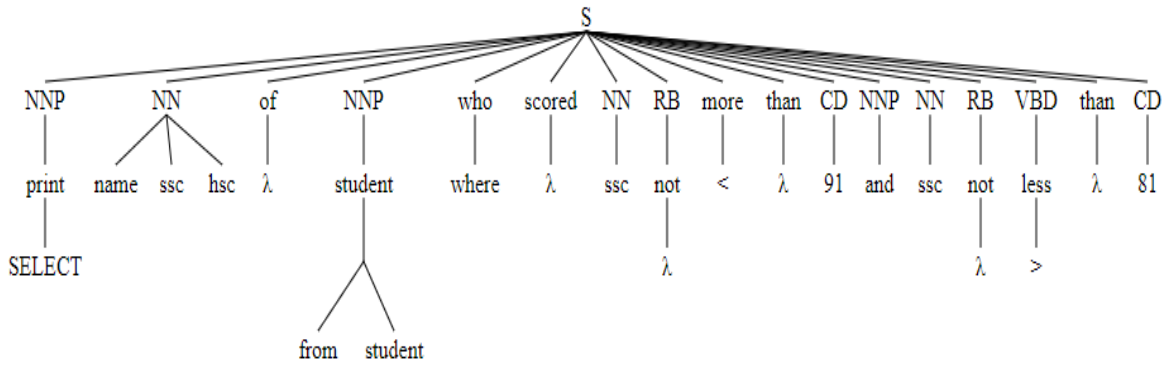
Syntax analysis:

[('print', 'NNP'), ('name', 'NN'), ('ssc', 'NN'), ('hsc', 'NN'), (‘of’) ,('student', 'NNP'), ('who', 'NNP'), (‘scored’), ('not', 'RB') , (‘more’) , (‘than’) , ('91', 'CD'), (‘in’) , (‘10th’) , ('and’, 'NNP'), ('not', 'RB'), ('less', 'VBD'), (‘than’) ,('81', 'CD')]

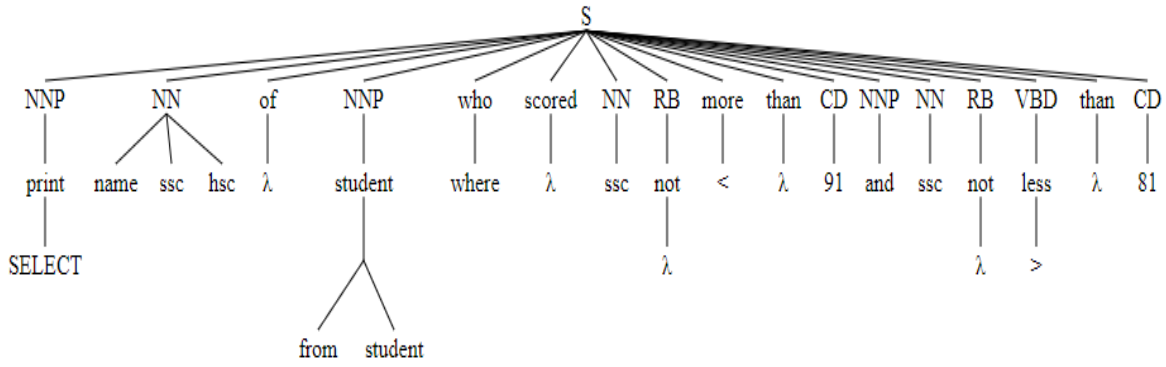
↓

Syntax Analyzer

↓



Intermediate representation & generation of SQL statements:



Intermediate code generator

['print', 'name', ',', 'ssc', ',', 'hsc', 'of', 'students', 'who', 'scored', 'not', 'more', 'than', 'ninety', 'one', 'in', '10th', 'and', 'not', 'less', 'than', 'eighty', 'one']

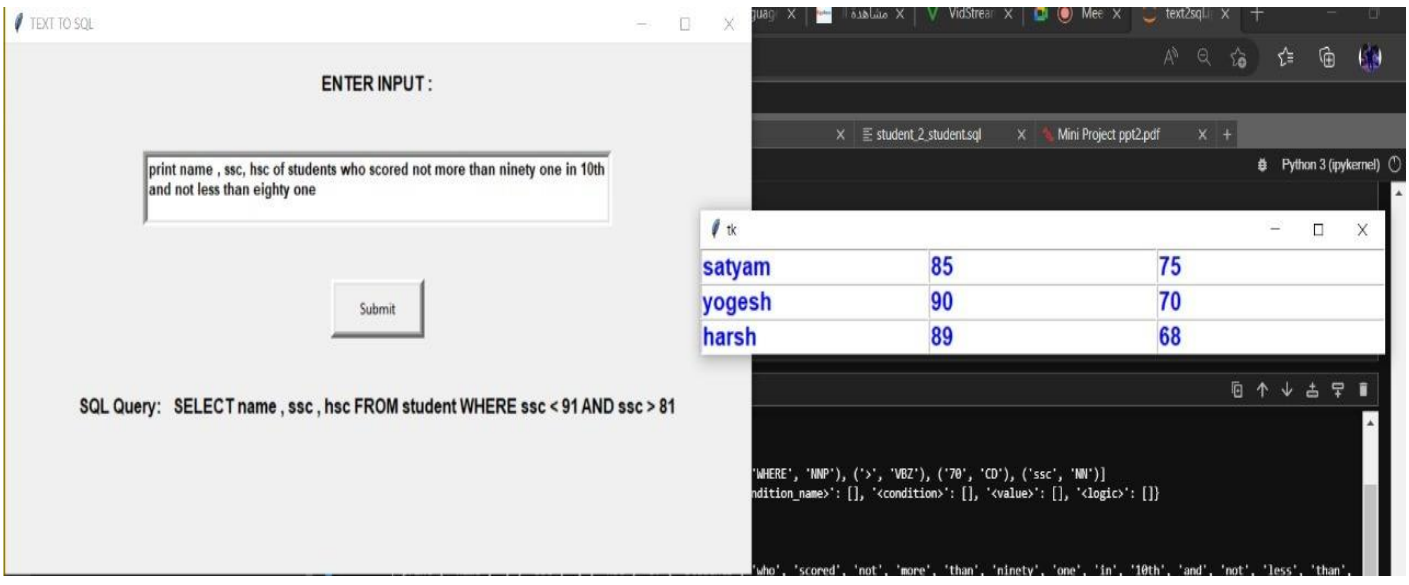
['print', 'name', 'ssc', 'hsc', 'student', 'who', 'not', 'more', 'ninety', 'one', '10th', 'and', 'not', 'less', 'eighty', 'one']

['SELECT', 'name', 'ssc', 'hsc', 'FROM student', 'WHERE', 'not', 'more', '90', '1', 'ssc', 'AND', 'not', 'less', '80', '1']

['SELECT', 'name', 'ssc', 'hsc', 'FROM student', 'WHERE', 'not', '<', '91', 'ssc', 'AND', 'not', '>', '81']

Sample input and output queries as results

```
['print', 'name', ',', 'ssc', ',', 'hsc', 'of', 'students', 'who', 'scored', 'not', 'more', 'than', 'ninety', 'one', 'in', '10th', 'and', 'not', 'less', 'than',
'eighty', 'one']
['print', 'name', 'ssc', 'hsc', 'student', 'who', 'not', 'more', 'ninety', 'one', '10th', 'and', 'not', 'le', 'eighty', 'one']
['SELECT', 'name', 'ssc', 'hsc', 'FROM student', 'WHERE', 'not', 'more', '90', '1', 'ssc', 'AND', 'not', 'le', '80', '1']
['SELECT', 'name', 'ssc', 'hsc', 'FROM student', 'WHERE', 'not', '<', '91', 'ssc', 'AND', 'not', '>', '81']
string2
sentence
[('SELECT', 'NNP'), ('name', 'NN'), ('ssc', 'NN'), ('hsc', 'NN'), ('FROM student', 'NNP'), ('WHERE', 'NNP'), ('not', 'RB'), ('<', 'VBZ'), ('91', 'CD'), ('ssc',
'NN'), ('AND', 'CC'), ('not', 'RB'), ('>', 'VB'), ('81', 'CD')]
{'<action>': [], '<attr_name>': [], '<table_name>': [], '<condition_name>': [], '<condition>': [], '<value>': [], '<logic>': []}
['SELECT'] ['name', ',', 'ssc', ',', 'hsc'] ['FROM student'] ['ssc'] ['<', '>'] ['91', '81'] ['AND']
 SELECT name , ssc , hsc FROM student WHERE ssc < 91 AND ssc > 81
```



Source code:

In [1]:
```
# !pip install mysql.connector
```

In [2]:
```
# !pip install word2number
```

In [3]:
```
# nltk.download('punkt')
```

In [4]:
```
# nltk.download('wordnet')
```

In [5]:
```
# !pip install mysql-connector-python
```

In [6]:
```
import nltk
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import mysql.connector
from word2number import w2n
import tkinter as tk
from tkinter import *

import dict
```

In [8]:
```
#frame = tk.Tk()
frame = tk.Tk()
frame.title("TEXT TO SQL")
frame.geometry('800x500')
```

Out[8]: ''

In [9]:
```
dict1= {
    "<action>":[],
    "<attr_name>":[],
    "<table_name>":[],
    "<condition_name>":[],
    "<condition>":[],
    "<value>":[],
    "<logic>":[]
      }
dictionary1=dict.dictionary11


dictionary3=dict.dictionary3
```

In [10]:
```
def printInput():
    word_list=[]
    word_list=module1()
    print(word_list)
    tokens_without_sw=[]
    tokens_without_sw=module2(word_list)
    print(tokens_without_sw)
    string1=[]
    string1=module3(tokens_without_sw)
    print(string1)
    string2=[]
    string2=module4(string1)
    print(string2)
    sentence=[]
```

```
sentence=module5(string2)
print(sentence)


action,attr_name,table_name,condition_name,condition,value,logic=module6(sentence)
print(action,attr_name,table_name,condition_name,condition,value,logic)

string4=""
string4=module7(action,attr_name,table_name,condition_name,condition,value,logic)
print(string4)

module8(string4)
```

In [11]:
```
def module1():

    text = inputtxt.get(1.0, "end-1c")
    text = text.lower()
    word_list = word_tokenize(text)
    return word_list
```

In [12]:
```
def module2(word_list):
    lemmatizer = WordNetLemmatizer()
    lem=[]
    for r in word_list:
        lem.append(lemmatizer.lemmatize(r))
```

```
    ignore_list =['the','record','database','table','information','a','are','is','to','marks','mark','of','in','than',"'s",',
    tokens_without_sw = [word for word in lem if not word in ignore_list]
    return tokens_without_sw
```

In [13]:
```
def module3(tokens_without_sw):

    global dictionary1




    to_be_append=""
    location_is=100
    selected_is=0

    string1=[]
    counter=0
    set_word=0
    for x in tokens_without_sw:
        i=0
        if x in dictionary1["ssc"]:
            tokens_without_sw[i]="ssc"
        elif x in dictionary1["aggregate"]:
            tokens_without_sw[i]="aggregate"
        elif x in dictionary1["name"]:
            tokens_without_sw[i]="name"
        elif x in dictionary1["hsc"]:
            tokens_without_sw[i]="hsc"
        elif x in dictionary1["DESC"]:
            tokens_without_sw[i]="DESC"
```

```
        elif x in dictionary1["ASC"]:
            tokens_without_sw[i]="ASC"
        elif x in dictionary1["SELECT"] and selected_is==0:
            tokens_without_sw[i]="SELECT"
            selected_is=1
        elif x in dictionary1["*"]:
            if location_is > i:
                if counter==0 :
                    tokens_without_sw[i]="*"
                    counter=1

        elif x in dictionary1["WHERE"]:
            tokens_without_sw[i]="WHERE"
        elif x in dictionary1["AND"]:
            tokens_without_sw[i]="AND"
        elif x in dictionary1["OR"]:
            tokens_without_sw[i]="OR"

        elif x in dictionary1["FROM student"]:
            if location_is==100:

                tokens_without_sw[i]="FROM student"
                location_is=i
        elif x in dictionary1["ORDER BY"]:
            tokens_without_sw[i]="ORDER BY"
        elif x in dictionary1["word"]:
             tokens_without_sw[i]=str(w2n.word_to_num(x))
```

```
            else:
                tokens_without_sw[i]=x
            if to_be_append!=tokens_without_sw[i]:
                to_be_append = tokens_without_sw[i]
                string1.append(tokens_without_sw[i])
            if counter==1 :
                  tokens_without_sw[i]=""
                  counter=2

            i=i+1
        return string1
```

In [14]:

```
def module4(string1):
    global dictionary1
    global dictionary3
    string2=[]
    i=0
    countercounter=0
    to_append=""
    len_string1=len(string1)

    for x in string1:

        if x in dictionary3["<"]:

            if string1[i-1]=="not":
                string1[i]=">"
            else:
                string1[i]="<"
        elif x in dictionary3[">"]:
```

```
            if string1[i-1]=="not":
                string1[i]="<"
            else:
                string1[i]=">"

        elif x in dictionary3["="]:

            string1[i]="="


        elif x in dictionary3["* FROM"]:

            string1[i]="* FROM"

        elif x in dictionary1["AND"]:

            countercounter=0
        elif x in dictionary1["OR"]:

            countercounter=0

        elif x in dictionary1["<number>"]:

            if i < len_string1:
                if countercounter==0:
                    try:
                        if string1[i+1] in dictionary1["<number>"]:
                            string1[i]=str(int(string1[i+1])+int(x))

                            countercounter=1
                    except:
                        print()

            else:
                string1[i]=x
            if to_append!=string1[i] and string1[i]!="":
                to_append = string1[i]
                string2.append(string1[i])

            if countercounter==1 :
                string1[i+1]=""
                countercounter=2



            i=i+1


        return string2
```

In [15]:

```
def module5(string2):
    print("string2")
    sentence=nltk.pos_tag(string2)
    print("sentence")
    return sentence
```

In [16]:

```
def module6(sentence):
    global dict1
    print(dict1)
    global dictionary1
    global dictionary3

    dict= {
      "NN": ["ssc","hsc","name","id","student","*"],
      }
    i=0
    locc=0
    for x in sentence:
        x1,x2=x
        if(x1=="WHERE"):
            locc=i
        i=i+1
    i=0

    set_select=0
    for x in sentence:

        x1,x2=x
        if i<locc and locc!=0:
            if x1 in dictionary1["SELECT"]:
                if set_select==0:
                    dict1["<action>"].append(x1)
                    set_select=1
            elif  x2=="NN" and x1!="FROM student":
                dict1["<attr_name>"].append(x1)
            elif  x2=="VB" and x1=="aggregate":
                dict1["<attr_name>"].append(x1)
        elif x1=="FROM student":
            dict1["<table_name>"].append(x1)
        elif x2!="NN":
            if x1 in dict["NN"]:
                dict1["<attr_name>"].append(x1)
    elif i>locc and locc!=0:
        if x2=="NN" and x1!="FROM student":
            dict1["<condition_name>"].append(x1)
        elif  x2=="VB" and x1=="aggregate":
            dict1["<condition_name>"].append(x1)
        elif x2!="NN":
            if x1 in dict["NN"]:
                dict1["<condition_name>"].append(x1)
    else:

        if x1 in dictionary1["SELECT"]:
            if set_select==0:
                dict1["<action>"].append(x1)
                set_select=1
        elif  x2=="NN" and x1!="FROM student":
            dict1["<attr_name>"].append(x1)
        elif  x2=="VB" and x1=="aggregate":
            dict1["<attr_name>"].append(x1)
        elif x1=="FROM student":
            dict1["<table_name>"].append(x1)
        elif x2!="NN":
            if x1 in dict["NN"]:
                dict1["<attr_name>"].append(x1)
```

```
    if x2=="$" or (x1 in dictionary3):
        dict1["<condition>"].append(x1)
    elif x1 in dictionary1["AND"]:
        dict1["<logic>"].append(x1)
    elif x1 in dictionary1["OR"]:
        dict1["<logic>"].append(x1)
    elif x2=="CD":
        dict1["<value>"].append(x1)
    i=i+1

action = dict1["<action>"]

attr_name = dict1["<attr_name>"]
arr=[]
j=1
for i in attr_name:
    if i=="*" and len(attr_name)>1:
        attr_name.remove("*")
for i in attr_name:
    if i!="*":
        arr.append(i)
        if j<len(attr_name):
            arr.append(",")
    else:
        arr.append("*")
    j=j+1
attr_name=arr
```

```
    table_name = dict1["<table_name>"]

    condition_name = dict1["<condition_name>"]
    count=0
    for i in condition_name:
        if i=="":
            count=count+1
    for j in range(count):
        condition_name.remove("")

    condition = dict1["<condition>"]
    value= dict1["<value>"]
    logic= dict1["<logic>"]
    dict1.clear()
    dict1={
    "<action>":[],
    "<attr_name>":[],
    "<table_name>":[],
    "<condition_name>":[],
    "<condition>":[],
    "<value>":[],
    "<logic>":[]
      }

    return action,attr_name,table_name,condition_name,condition,value,logic
```

In [17]:
```
def module7(action,attr_name,table_name,condition_name,condition,value,logic):
    len_condition_name = len(condition_name)


    len_condition = len(condition)


    len_value = len(value)

    len_logic = len(logic)


    phase1 = action + attr_name+ table_name

    phase2=""
    #lenx=maximum(len_condition_name, len_condition, len_value)

    a=len_condition_name
    b=len_condition
    c=len_value
    def maximum(a,b,c):

        if (a>=b) and (a>=c):
            largest = a

        elif (b >= a) and (b >= c):
            largest = b
        else:
            largest = c

        return largest
```

```
# Driven code

lenx=maximum(a, b, c)

if lenx==len_condition_name:
    print()
else:

    cond=condition_name[len_condition_name-1]

    for x in range(lenx-1):
        condition_name.append(cond)
len_condition_name=len(condition_name)

for i in range(len_condition_name):
    phase2 = phase2 + " " + condition_name[i]

    if i <len_condition:
        phase2 = phase2 + " " + condition[i]
    elif i>= len_condition:
        phase2 = phase2 + " " + condition[len_condition-1]
    if i <len_value:
        phase2 = phase2 + " " + value[i]
    elif i>= len_value:
        phase2 = phase2 + " " + value[len_value-1]
    if len_condition_name > len_logic and i < len_logic:
        phase2 = phase2 + " " + logic[i]
```

```
    result =""
    for i in phase1:
        result = result + " " +i
    if phase2!="":
        string4=(result +" "+"WHERE"+ phase2)
    else:
        string4=(result)
    return string4
```

In [18]:
```
def module8(string4):

    resultlist=[]
    myconn = mysql.connector.connect(host = "localhost", user = "root",passwd = "1111",database = "student")

    #creating the cursor object
    cur = myconn.cursor()

    try:
        #Reading the Employee data
        cur.execute(string4)

        #fetching the rows from the cursor object
        result = cur.fetchall()
        resultlist = result
        #printing the result

        #print(resultlist)
        #for x in resultlist:
```

```
except:
    print("error")
    myconn.rollback()

myconn.close()
class Table:

    def __init__(self,root):

        # code for creating table
        for i in range(total_rows):
            for j in range(total_columns):

                self.e = Entry(root, width=20, fg='blue',
                               font=('Arial',16,'bold'))

                self.e.grid(row=i, column=j)
                self.e.insert(END, resultlist[i][j])
if resultlist != [] :
    total_rows = len(resultlist)
    total_columns = len(resultlist[0])

    # create root window
    root = Tk()
    t = Table(root)
    root.mainloop
    lbl.config(text = "SQL Query:  "+string4)
```

```
    else :
        print("No Data Found")
        lbl.config(text = "SQL Query:  "+string4)

        lbl1.config(text = "No Data Found")
```

In [19]:
```
lbl10 = tk.Label(frame, text = "ENTER INPUT :",font='Helvetica 12 bold')
lbl10.pack(pady=20)
inputtxt = tk.Text(frame,
                   height = 3,
                   width = 70,
                   bd=5,
                   font='Helvetica 10 bold'

                  )
lbl6 = tk.Label(frame, text = "")
lbl6.pack()
inputtxt.pack()
lbl4 = tk.Label(frame, text = "")
lbl4.pack()
lbl5 = tk.Label(frame, text = "")
lbl5.pack()

# Button Creation
printButton = tk.Button(frame,
                        text = "Submit",
                        bd=5,
                        height = 2,
                        width = 12,
                        command = printInput)
printButton.pack()
lbl8 = tk.Label(frame, text = "")
lbl8.pack()
lbl9 = tk.Label(frame, text = "")
lbl9.pack()
# Label Creation
lbl = tk.Label(frame, text = "",font='Helvetica 12 bold')
lbl.pack()
lbl2 = tk.Label(frame, text = "")
lbl2.pack()
lbl1 = tk.Label(frame, text = "")
lbl1.pack()
```

In [ ]:
```
frame.mainloop()
```

```
['fetch', 'all', 'the', 'information', 'of', 'the', 'students', 'who', 'have', 'scored', 'more', 'than', '70', 'in', '10t
h']
['fetch', 'all', 'student', 'who', 'have', 'more', '70', '10th']
['SELECT', '*', 'FROM student', 'WHERE', 'more', '70', 'ssc']
['SELECT', '*', 'FROM student', 'WHERE', '>', '70', 'ssc']
string2
sentence
[('SELECT', 'NNP'), ('*', 'NNP'), ('FROM student', 'NNP'), ('WHERE', 'NNP'), ('>', 'VBZ'), ('70', 'CD'), ('ssc', 'NN')]
{'<action>': [], '<attr_name>': [], '<table_name>': [], '<condition_name>': [], '<condition>': [], '<value>': [], '<logic
>': []}
['SELECT'] ['*'] ['FROM student'] ['ssc'] ['>'] ['70'] []

 SELECT * FROM student WHERE ssc > 70
['print', 'name', ',', 'ssc', ',', 'hsc', 'of', 'students', 'who', 'scored', 'not', 'more', 'than', 'ninety', 'one', 'i
n', '10th', 'and', 'not', 'less', 'than', 'eighty', 'one']
```

References used it:

1. **Jupyter notebook in Anaconda**
2. **Python language.**
3. **SQL workbench.**
4. **Libraries (nltk , mysql.connector , word2number , mysql-connector-python , tkinter).**

Link of all Project source:

https://github.com/AbdelrahmanHusseinElshimy/NLP-to-Query-project.git