

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
ORGANIZACIÓN DE LENGUAJES Y
COMPILADORES 2
ESCUELA DE VACACIONES DICIEMBRE 2020
ING. LUIS ESPINO
TUTOR ACADÉMICO: JUAN CARLOS MAEDA



Proyecto1: BD Tytus

Manual Técnico

Integrantes:

Nombres:

Brian Daniel Xiloj del Cid

Javier Estuardo Lima Abrego

Yaiza Estefanía Pineda González

César Alejandro Chinchilla González

Carnes:

201603037

201612098

201610673

201612132

Introducción

El siguiente manual explica el funcionamiento y flujo del código de la aplicación que es un sistema administrador de base de datos (DBMS) Tytus, donde se pueden ingresar las consultas en un área de texto.

Para la elaboración de la aplicación se utilizó el IDE Pycharm, como lenguaje de la aplicación Python, con la herramienta PLY para el analizador del lenguaje SQL basado en PostgreSQL.

Descripción de los archivos Principales

Archivo main.py:

Es el archivo principal que ejecuta la aplicación, este despliega la interfaz gráfica de la aplicación y sus opciones para el funcionamiento de ella. Usa la librería tkinter para las partes gráficas y así simular el QueryTool.

```
main.py x
1 import Gramatica.Gramatica as g
2 import TablaSimbolos.TS as ts
3 import os
4 import graphviz
5 import sys
6 import threading
7 import Errores.Nodo_Error as error
8 import Errores.ListaErrores as lista_err
9 from Reportes.ReporteError import ReporteError
10 from Reportes.ReporteGramatical import ReporteGramatical
11 from tkinter import *
12 from tkinter import filedialog
13 from tkinter import scrolledtext
14 from tkinter import messagebox
15 from tkinter import ttk
16 import os, shutil
17
18
19 if __name__ == '__main__':
20     from LineNumber import LineMain
21     from Graphics import Tkinter
22     from ColorLight import ColorLight
23 else:
24     from LineNumber import LineMain
25     from Graphics import Tkinter
26     from ColorLight import ColorLight
27
28 class Connect:
29     def __init__(self, pad):
30         self.pad = pad
31         self.modules_connections()
32
33     def modules_connections(self):
34         LineMain(self.pad)
35         ColorLight(self.pad)
36         return
37
38
39 class Interfaz:
40     def __init__(self):
41         os.environ['DB'] = 'None'
42         self.borrarArchivos()
43         self.root = Tk()
44         #self.root.attributes('-zoomed', True)
45         self.root.title('TytusDB - Team 19')
46         self.root.geometry("1100x700")
47         self.root.resizable(1, 1)
48         self.txtarea = TextPad(self.root, pady=5)
49         self.consola = scrolledtext.ScrolledText(self.root, width=300, height=15, font=("Consolas", 11))
50         self.frame = ttk.Frame(self.root)
51         self.frame.pack(fill=X, side='left')
52         #self.tree_consultas = ttk.Treeview(self.frame)
53         #self.tree_consultas["columns"] = ("Tipo", "Descripcion", "Fila", "Columna")
54         #self.tree_consultas.grid(column=1, row=2, sticky=S)
55         #self.tree_consultas.column("#0", width=600, stretch=N0)
56         #self.tree_consultas.column("Tipo", width=50, minwidth=20, stretch=N0)
57         #self.tree_consultas.column("Descripcion", width=50, minwidth=20)
58         #self.tree_consultas.column("Fila", width=50, minwidth=20)
59         #self.tree_consultas.column("Columna", width=50, minwidth=20)
60         #self.tree_consultas.heading("#0", text="Hola", anchor=W)
61         #self.tree_consultas.heading("Descripcion", text="Descripcion", anchor=W)
62         #self.tree_consultas.heading("Tipo", text="Tipo", anchor=W)
63         #self.tree_consultas.heading("Fila", text="Fila", anchor=W)
64         #self.tree_consultas.heading("Columna", text="Columna", anchor=W)
65         self.v = StringVar()
66         lab = Label(self.root, textvariable=self.v, font=("Helvetica", 11, "bold"))
67         self.v.set('Base de Datos: ' + os.environ['DB'])
68         lab.pack(side=TOP, anchor=E)
69         lbl = Label(self.root, text="Consola de Salida:", font=("Helvetica", 11))
70         lbl.pack(side=TOP, anchor=W)
71         #self.tree_consultas.pack(side=LEFT, fill=X, pady=5, padx=5)
72         self.consola.pack(side=RIGHT)
73         self.consola.config(foreground='white', bg_='black')
74         self.menubar = Menu(self.root)
75         self.root.config(menu=self.menubar)
76         archivo = Menu(self.menubar, tearoff=0)
```

Archivo Gramatica/Gramatica.py:

Es el archivo donde está escrita la gramática que reconoce el lenguaje SQL para nuestra aplicación. También es el que nos permite construir el en forma lógica el árbol AST y nuestros reportes de errores a través del análisis sintáctico.

```
1 import Errores.Nodo_Error as err
2 from ply import lex
3 from AST.Sentencias import Raiz, Sentencia
4 import AST.SentenciasDDL as DDL
5 import ply.yacc as yacc
6
7 reservadas = {
8     'select': 't_select',
9     'distinct': 't_distinct',
10    'as': 't_as',
11    'from': 't_from',
12    'where': 't_where',
13    'having': 't_having',
14    'avg': 't_avg',
15    'min': 't_min',
16    'max': 't_max',
17    'sum': 't_sum',
18    'count': 't_count',
19    'insert': 't_insert',
20    'into': 't_into',
21    'values': 't_values',
22    'delete': 't_delete',
23    'update': 't_update',
24    'true': 't_true',
25    'false': 't_false',
26    'not': 't_not',
27    'and': 't_and',
28    'or': 't_or',
29    'smallint': 't_smallint',
30    'integer': 't_integer',
31    'bigint': 't_bigint',
32    'decimal': 't_decimal',
33    'numeric': 't_numeric',
34    'real': 't_real',
35    'double': 't_double',
36    'precision': 't_precision',
37    'money': 't_money',
38    'character': 't_character',
39    'varying': 't_varying',
40    'varchar': 't_varchar',
```

[illegible]

Archivo Nodo.py:

Es la clase que ayuda a la construcción del árbol AST al archivo gramática.

```
1  import abc
2
3  #id estatico para que cada nodo tenga un id unico al graficar
4  id_arbol = 1
5
6  def asign_id_arbol():
7      global id_arbol
8      temp_id = str(id_arbol)
9      id_arbol += 1
10     return temp_id
11
12     def reset_id_arbol():
13         global id_arbol
14         id_arbol = 1
15
16     class Nodo(metaclass=abc.ABCMeta):
17
18         def __init__(self, fila = 0, columna = 0):
19             self.fila = fila
20             self.columna = columna
21             self.mi_id = asign_id_arbol()
22
23         @abc.abstractmethod
24         def ejecutar(self, TS, Errores):
25             pass
26
27         @abc.abstractmethod
28         def getC3D(self, TS):
29             pass
30
31         def graficarasc(self, padre, grafica):
32             grafica.node(self.mi_id, self.__class__.__name__)
33             grafica.edge(padre, self.mi_id)
```

Archivo Expresiones.py:

Es el archivo que recibe y resuelve las expresiones que vengan en el lenguaje SQL.

```
1 import AST.Nodo as Node
2 import math as m
3 from TablaSimbolos.Tipos import *
4 from TablaSimbolos.TS import *
5 from Errores.Nodo_Error import *
6
7 class Expression(Node.Nodo):
8     def __init__(self, *args):
9         if len(args) == 6:
10             if args[5] == 'math2':
11                 self.val1 = args[1]
12                 self.val2 = args[2]
13                 self.line = args[3]
14                 self.column = args[4]
15                 self.function = args[0]
16                 self.op_type = args[5]
17             else:
18                 self.exp1 = args[0]
19                 self.exp2 = args[1]
20                 self.op = args[2]
21                 self.line = args[3]
22                 self.column = args[4]
23                 self.op_type = args[5]
24                 self.val = None
25                 self.type = None
26         elif len(args) == 5:
27             if args[4] == 'unario':
28                 self.op_type = args[4]
29                 self.type = args[0]
30                 self.val = args[1]
31                 self.line = args[2]
32                 self.column = args[3]
33             elif args[4] == 'as':
34                 self.type = None
35                 self.val = args[0]
36                 self.asid = args[1]
37                 self.line = args[2]
38                 self.column = args[3]
39                 self.op_type = 'as'
40             elif args[4] == 'aggregate':
41                 self.val1 = args[0]
42
43     def ejecutar(self, TS, Errores):
44         if self.op_type == 'valor':
45             return self
46         elif self.op_type == 'unario':
47             self.val.ejecutar(TS, Errores)
48             if self.type == '-':
49                 self.val = -self.val.val
50             return self
51         elif self.op_type == 'as' or self.op_type == 'in' or self.op_type == 'agg':
52             self.val.ejecutar(TS, Errores)
53             self.asid.ejecutar(TS, Errores)
54             return self
55         elif self.op_type == 'math':
56             if self.function == 'ceil' or self.function == 'ceiling':
57                 self.val.ejecutar(TS, Errores)
58                 if isinstance(self.val.val, int):
59                     self.val = m.ceil(self.val.val)
60                 else:
61                     self.val = m.ceil(self.val.val)
62             elif self.function == 'abs':
63                 self.val = m.fabs(self.val.val)
64             elif self.function == 'cbt':
65                 self.val = m.ceil(self.val.val**(1/3))
66             elif self.function == 'degrees':
67                 self.val = m.degrees(self.val.val)
68             elif self.function == 'div':
69                 self.val = m.exp(self.val.val)
70             elif self.function == 'exp':
71                 self.val = m.exp(self.val.val)
72             elif self.function == 'factorial':
73                 self.val = m.factorial(self.val.val)
74             elif self.function == 'floor':
75                 self.val = m.floor(self.val.val)
76             elif self.function == 'gcd':
77                 self.val = m.gcd(self.val.val)
```

Archivo Sentencias.py:

Construye gráficamente el árbol AST además de ejecutar de forma general las instrucciones.

```
from graphviz import Digraph
from AST.Nodo import Nodo
from AST.Nodo import reset_id_arbol
import AST.Nodo as Node
from AST.Expresiones import *

class Raiz(Node.Nodo):
    def __init__(self, Errores, sentencias = [], fila = 0, columna = 0):
        super().__init__(fila=fila, columna=columna)
        self.sentencias = sentencias
        self.errores = Errores

    def ejecutar(self, TS, Errores):
        respuesta = ''
        for hijo in self.sentencias:
            if isinstance(hijo, Expression):
                respuesta += ''
                respuesta += hijo.ejecutar(TS, Errores) + '\n'
        return respuesta

    def getC3D(self, TS):
        pass

    def graficarasc(self, padre = None, grafica=None):
        grafica = Digraph(name="AST", comment='AST generado')
        grafica.edge_attr.update(arrowhead='none')
        grafica.node(self.mi_id, "SentenciasSQL")
        for hijo in self.sentencias:
            hijo.graficarasc(self.mi_id, grafica)
        grafica.render(directory='Reportes/graficaAST', view=True)
        reset_id_arbol()

class Sentencia(Raiz):
    def __init__(self, nombre_sentencia, sentencias, fila = 0, columna = 0):
        super().__init__(None, sentencias=sentencias, fila=fila, columna=columna)
        self.nombre_sentencia = nombre_sentencia

#No necesito definir ejecutar porque ya lo hereda de Raiz y realiza lo mismo aqui que alli
```


Archivo SentenciasDDL.py:

Ejecuta las instrucciones DDL del lenguaje SQL de la aplicación, como create table y alterTable.

```
1 from typing import Type
2 from AST.Nodo import Nodo
3 import data.jsonMode as JM
4 import Errores.Nodo_Error as err
5 from prettytable import PrettyTable
6 import TypeCheck.Type_Checker as TypeChecker
7 import os
8
9
10 class CreateDatabase(Nodo):
11     def __init__(self, fila, columna, nombre_BD, or_replace=False, if_not_exist=False,
12                 super().__init__(fila, columna)
13                 self.nombre_DB = nombre_BD.lower()
14                 self.or_replace = or_replace
15                 self.if_not_exist = if_not_exist
16                 self.owner = owner
17                 self.mode = mode
18
19     def ejecutar(self, TS, Errores): #No se toca el owner para esta fase
20         if self.if_not_exist:
21             if self.nombre_DB in TypeChecker.showDataBases():
22                 Errores.insertar(err.Nodo_Error('42P04', 'duplicated database', self.fila, self.columna))
23                 return '42P04: duplicated database\n'
24
25         if self.or_replace:
26             respuesta = TypeChecker.dropDataBase(self.nombre_DB)
27             if respuesta == 1:
28                 Errores.insertar(err.Nodo_Error('XX000', 'internal_error', self.fila, self.columna))
29                 return 'XX000: internal_error\n'
30
31         if self.mode > 5 or self.mode < 1:
32             Errores.insertar(err.Nodo_Error('Semantico', 'El modo debe estar entre 1 y 5', self.fila, self.columna))
33             return 'Error semantico: El modo debe estar entre 1 y 5\n'
34
35         respuesta = TypeChecker.createDataBase(self.nombre_DB, self.mode, self.owner)
36         if respuesta == 2:
37             Errores.insertar(err.Nodo_Error('42P04', 'duplicated database', self.fila, self.columna))
38             return '42P04: duplicated database\n'
39         if respuesta == 1:
40             Errores.insertar(err.Nodo_Error('P0000', 'plpgsql_error', self.fila, self.columna))
41             return 'P0000: plpgsql_error\n'
```

Archivo SentenciasDML.py:

Ejecuta las instrucciones DML del lenguaje SQL de nuestra aplicación, como la instrucción select.

```
1  import ASI.Nodo as Node
2      from prettytable import PrettyTable
3      from Errores.Nodo_Error import *
4      import data.jsonMode as jm
5      import os
6      from operator import itemgetter
7      import TypeCheck.Type_Checker as tp
8
9
10 class Select(Node.Nodo):
11     def __init__(self, *args):
12         if args[0] == '*':
13             self.arguments = None
14             self.tables = args[1]
15             self.line = args[5]
16             self.column = args[6]
17             self.conditions = args[2]
18             self.grp = args[3]
19             self.ord = args[4]
20             self.result_query = PrettyTable()
21         else:
22             self.arguments = args[0]
23             self.tables = args[1]
24             self.line = args[5]
25             self.column = args[6]
26             self.conditions = args[2]
27             self.grp = args[3]
28             self.ord = args[4]
29             self.result_query = PrettyTable()
30
31     def ejecutar(self, TS, Errores):
32         columnas = []
33         col_dict = {}
34         tuplas = []
35         tuplas_aux = []
36         ordencol = []
37         db = os.environ['DB']
38         result = 'Query from tables: '
39         contador = 0
40         #-----FROM
41         if len(self.tables) != 0:
```

Archivo TypeChecker.py:

Es el archivo que realiza la funcionalidad del TypeChecker en nuestra aplicación, es decir el que permite verificar todos los tipos de datos de las Tablas y que su almacenamiento de todos los objetos sea correcto.

```
50
51 def dropDataBase(database: str):
52     # 0:operación exitosa, 1: error en la operación, 2: base de datos no existente
53     respuesta = JM.dropDataBase(database)
54     if respuesta == 0:
55         return lista_bases.eliminarBaseDatos(database)
56     return respuesta
57
58 def obtenerBase(database: str):
59     actual = lista_bases.primeros
60     while(actual != None):
61         if actual.nombreBase == database:
62             break
63         actual = actual.siguiente
64     return actual
65
66 def createTable(database: str, table: str, numberColumns: int):
67     # 0:operación exitosa, 1: error en la operación, 2: base inexistente, 3: tabla existente
68     respuesta = JM.createTable(database, table, numberColumns)
69     if respuesta == 0:
70         actual = obtenerBase(database)
71         if(actual != None):
72             if not actual.listaTablas.existeTabla(table):
73                 actual.listaTablas.agregarTabla(Tabla.Tabla(table))
74                 return 0
75             else:
76                 return 3
77         else:
78             return 2
79     return respuesta
80
81 def showTables(database:str):
82     respuesta = JM.showTables(database)
83     return respuesta
84
85 def createColumn(database:str, table:str, nombre:str, tipo):
86     # 0:operación exitosa, 1: error en la operación, 2: base de datos inexistente, 3: tabla inexistente, 4: columna ya exist
87     actualBase = obtenerBase(database)
88     if(actualBase!=None):
89         if not actualBase.listaTablas.existeTabla(table):
90             return 3
91     -
```