UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
ORGANIZACIÓN DE LENGUAJES Y COMPILADORES 2
MSC. LUIS ESPINO
AUX. JUAN CARLOS MAEDA
GRUPO 18

TytusDB:Manual Técnico

ALEX RENÉ LÓPEZ ROSA 201602999
MIAMIN ELIEL BARRIOS ARRIVILLAGA 201603016
EDWARD DANILO GÓMEZ HERNÁNDEZ 201602909
KEVIN GOLWER ENRIQUE RUIZ BARBALES 201603009

Indice

Objetivos

- **General:** Proporcionar una guía para el lector en la cual se describa el comportamiento y la manera en que fue desarrollado el software, brindando un entendimiento más sencillo para posteriores ediciones o actualizaciones.

- **Especifico:** Brindar una descripción total del software, especificando de manera detallada todo el proceso de desarrollo, como también la descripción de todas las herramientas externas empleadas para el desarrollo.

Introducción

Se espera que por medio de este documento el usuario pueda entender de mejor manera la forma en que fue desarrollado el software, los herramientas utilizadas y como fueron empleadas para el desarrollo y los requerimientos necesarios para su utilización.

Alcance

El contenido de este documento está enfocado hacia otros programadores, debido a que este contara con una gran variedad términos técnicos los cuales no podrán ser comprendidos por cualquier usuario.

Descripción del Proyecto

Este proyecto consiste en compilador capaz de reconocer entradas de texto realizadas en lenguaje SQL (PostgreSQL) específicamente, del cual se es capaz de analizar y ejecutar una gran cantidad de funcionalidades de igual manera en que las ejecutaría el gestor de PostgreSQL, este compilador a su vez hace uso de las funciones de la librería storage manager por medio de la cual es capaz de guardar la información trabajada por las consultas en un sistema de almacenamiento conformado por distintos tipos de árboles.

De igual manera de como este compilador hace uso del storage manager para su funcionamiento, este es empleado por un gestor web en el cual se escriben las consultas que posteriormente el compilador analiza y ejecuta a su vez generar un código en tres direcciones equivalente para la serie de instrucciones ejecutas desde la misma aplicación.

Requerimientos

El compilador desarrollado debe de ser capaz de analizar y ejecutar consultas tales como:

1) **DDL:**
    - ➢ Create Database
    - ➢ Create Table
    - ➢ Create Type
    - ➢ Create Index
    - ➢ Alter Database
    - ➢ Alter Table
    - ➢ Alter Index
    - ➢ Drop Database
    - ➢ Drop Table
    - ➢ Drop Index
2) **DML:**
    - ➢ Insert Table
    - ➢ Update Table
    - ➢ Delete Table
    - ➢ Select
    - ➢ Show Databases
    - ➢ Show Tables

- ➢ Use Database

3) **PLSQL:**
   - ➢ **Functions:**
     - **Create Function**
     - **Drop Function**
     - **Use Function**
   - ➢ **Procedure:**
     - **Create Procedure**
     - **Drop Procedure**
     - **Execute Procedure**

A su vez este cuenta con una gran variedad de funciones propias del lenguaje de las cuales el usuario puede emplear para realizar las distintas operaciones que el usuario desee implementar tales como abs,cbrt,cos,sen,etc.

Herramientas Utilizadas

Python: Este fue el lenguaje utilizado para la realización de todo el proyecto.
Download Python | Python.org

PLY: Esta fue la herramienta utilizada para la construcción del analizador haciendo uso de la gramática para el lenguaje se construyo el analizador capaz de ejecutar el lenguaje SQL.
PLY (Python Lex-Yacc) (dabeaz.com)

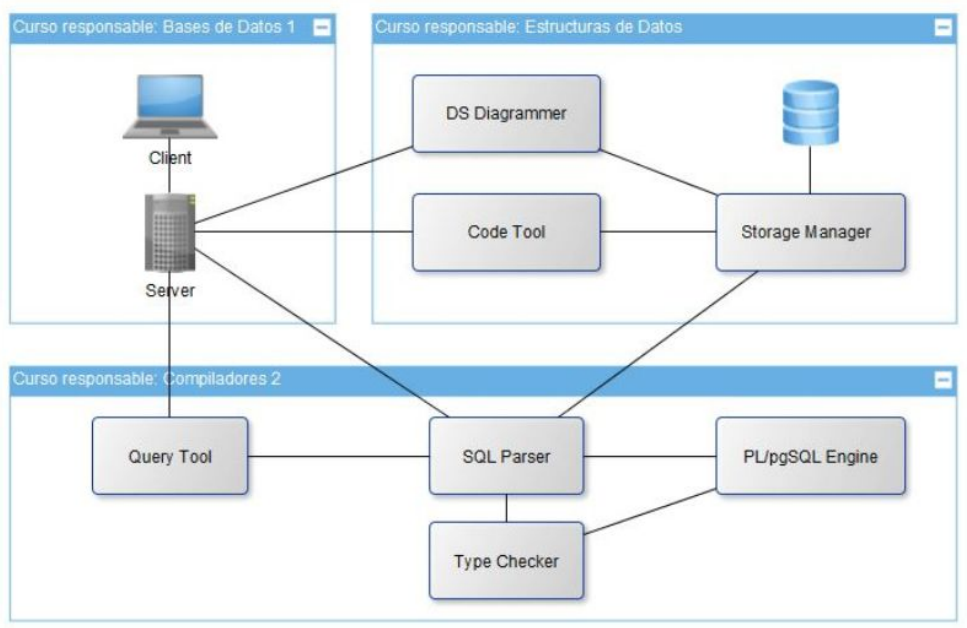
Visual Code: Este fue ide utilizado para llevar a cabo la codificación del proyecto fue seleccionado debido a la gran variedad de ventajas que brinda.
Download Visual Studio Code - Mac, Linux, Windows

Arquitectura del Software

Tal y como se explicó con anterioridad el proyecto se encuentra divido en varios bloques los cuales se encargan de codificar las distintas funcionalidades.

Para el bloque del compilador en especifico se cuenta con el compilador mismo, capaz de analizar cualquier entrada en lenguaje SQL de las funciones definidas con anterioridad, a su vez cuenta con una estructura en la cual se lleva el control de tipos para cualquier acción que el usuario realice la ejecución de alguna consulta por último también se cuenta con una interfaz gráfica por medio de la cual el usuario puede ingresar las distintas consultas a ejecutar y visualizar los resultados

obtenidos de dichas consultas. A su vez esta misma interfaz cuenta con un apartado desde el cual el usuario también puede acceder a los distintos reportes generados con la ejecución realizada.



Desarrollo

Para la construcción y acceso de manera más ordenada a la información importante de la obtenida de las distintas producciones de la gramática se optó por la realización de múltiples clases enfocadas a guardar la información más importante de cada funcionalidad.

Factores Tomados en cuenta al utilizar gramática ascendente

- PLY es una herramienta que implementa el análisis sintáctico LR
- PLY por defecto genera sus tablas de análisis sintáctico utilizando el LALR(1)
- PLY permite la resolución ambigüedad a través de reglas de precedencia
- Una gramática LR es adecuada para gramáticas extensas debido a que no se utilizan producciones para eliminar la recursividad
- Manejar el almacenamiento de información implementando clases abstractas directamente desde las producciones sin tener que acceder a la pila
- Agregar n elementos a un árbol cualquiera da como resultado una complejidad O(n log n) y recorrer los elementos del árbol en inorden tiene complejidad O(n), sin

Instrucciones

Este tipo de clases se creó para almacenar la información de cada uno de las distintas funcionalidades del sistema a continuación se mostrará la manera en que se crearon algunas de estas clases.

```
class Instruccion:
    '''This is an abstract class'''



class CrearBD(Instruccion) :
    '''
        Esta clase representa la funcion para crear una base de datos solo recibe el nombre d
    '''
    def __init__(self,reemplazar,verificacion,nombre, propietario, modo) :
        self.reemplazar = reemplazar
        self.verificacion = verificacion
        self.nombre = nombre
        self.propietario = propietario
        self.modo = modo


class CrearTabla(Instruccion) :
    '''
        Esta clase representa la instrucción crear tabla.
        La instrucción crear tabla recibe como parámetro nombre de tabla, lista de columnas y
    '''

    def __init__(self, nombre, padre, columnas = []) :
        self.nombre = nombre
        self.columnas = columnas
        self.padre = padre
```

```
class CrearType(Instruccion) :
    '''
        Esta clase representa la instrucción crear tipo.
        La instrucción crear tipo recibe como parámetro nombre del tipo, lista de valores
    '''

    def __init__(self, nombre, valores = []) :
        self.nombre = nombre
        self.valores = valores


class EliminarTabla(Instruccion) :
    '''
        Esta clase representa la instrucción drope table.
        La instrucción drope table recibe como parámetro la existencia y el nombre
    '''

    def __init__(self, existencia, nombre) :
        self.nombre = nombre
        self.existencia = existencia

class EliminarDB(Instruccion) :
    '''
        Esta clase representa la instrucción drope database.
        La instrucción drope database recibe como parámetro la existencia y el nombre
    '''

    def __init__(self, existencia, nombre) :
        self.nombre = nombre
        self.existencia = existencia
```

```
class Insertar(Instruccion):
    '''
        Estan clase representa los valores a insertar en una tabla
    '''
    def __init__(self, nombre, columnas, valores=[]) :
        self.nombre = nombre
        self.columnas = columnas
        self.valores = valores

class Actualizar(Instruccion):
    '''
        Esta clase representa los valores a actualizar de la tabla
    '''
    def __init__(self, nombre, condicion, valores=[]) :
        self.nombre = nombre
        self.condicion = condicion
        self.valores = valores

class columna_actualizar(Instruccion):
    '''
        Esta clase representa las columnas a actualizar
    '''
    def __init__(self, nombre, valor) :
        self.nombre = nombre
        self.valor = valor
```

```python
class SELECT(Instruccion):
    '''
        Esta clase representa a una select
    '''
    def __init__(self, cantidad, parametros, cuerpo, funcion_alias):
        self.cantida=cantidad
        self.parametros=parametros
        self.cuerpo=cuerpo
        self.funcion_alias=funcion_alias

class Funcion_Alias(Instruccion):
    '''
        Esta clase representa un funcion junto a su alias
    '''
    def __init__(self, nombre, alias):
        self.nombre=nombre
        self.alias=alias

class CUERPO_SELECT(Instruccion):
    '''
        Esta clase representa el cuerpo de un select
    '''
    def __init__(self, b_from, b_join, b_where, b_group, b_having, b_order):
        self.b_from=b_from
        self.b_join=b_join
        self.b_where=b_where
        self.b_group=b_group
        self.b_having=b_having
        self.b_order=b_order
```

Expresiones

Este tipo de clase fue utilizada para clasificar e identificar los distintos tipos de operaciones que se podían realizar haciendo uso de enums se listaron en distintos tipos las operaciones de cada clase, con la ayuda de esto se formó la parte de operaciones de la gramática y posteriormente al ejecutar identificar de manera sencilla el tipo de operación encontrado.

```python
from enum import Enum

class OPERACION_ARITMETICA(Enum) :...

class OPERACION_RELACIONAL(Enum) :...

class OPERACION_LOGICA(Enum):...

class OPERACION_ESPECIAL(Enum):...

class OPERACION_MATH(Enum):...

class OPERACION_BINARY_STRING(Enum):...

class OPERACION_PATRONES(Enum):...

class FUNCIONES_SELECT(Enum):...

class Expresion:
    '''
        Esta clase representa una expresión numérica
    '''

class Operacion_Aritmetica(Expresion) :
    '''
        Esta clase representa una operacion aritmetica entre dos expresiones y
    '''

    def __init__(self, op1, op2, operador) :
        self.op1 = op1
        self.op2 = op2
        self.operador = operador
```

```python
class Operacion_Relacional(Expresion):
    '''
        Esta clase representa una operacion relacional
    '''

    def __init__(self, op1, op2, operador):
        self.op1 = op1
        self.op2 = op2
        self.operador = operador

class Operacion_Logica_Binaria(Expresion):
    '''
        Esta clase representa una operacion logica binaria
    '''

    def __init__(self, op1, op2, operador):
        self.op1 = op1
        self.op2 = op2
        self.operador = operador

class Operacion_Logica_Unaria(Expresion):
    '''
        Esta clase representa una operacion logica unaria
    '''

    def __init__(self,op):
        self.op = op
```

```python
class Operacion_Especial_Binaria(Expresion):
    '''
        Esta clase representa una operacion especial binaria
    '''

    def __init__(self, op1, op2, operador):
        self.op1 = op1
        self.op2 = op2
        self.operador = operador

class Operacion_Especial_Unaria(Expresion):
    '''
        Esta clase representa una operacion especial unaria
    '''

    def __init__(self, op, operador):
        self.op = op
        self.operador = operador

class Negacion_Unaria(Expresion):
    '''
        Esta clase representa una negacion unaria para un operador
    '''

    def __init__(self,op):
        self.op=op

class Operando_Numerico(Expresion):
    '''
        Esta clase representa un operando del tipo numerico el cual puede ser entero o decimal
    '''
```

```python
class Operando_ID(Expresion):
    '''
        Esta clase represnta un operando del tipo identificador
    '''

    def __init__(self,id=""):
        self.id=id

class Operando_Cadena(Expresion):
    '''
        Esta clase representa un operando de tipo cadena
    '''

    def __init__(self,valor=""):
        self.valor=valor

class Operando_Booleano(Expresion):
    '''
        Esta clase representa un operando de tipo booleano
    '''

    def __init__(self,valor=False):
        self.valor=valor
```

Gramática

Para la implementación del lenguaje usando PLY se eligió una gramática ascendente la cual se adapta de mejor manera a la herramienta que se iba a utilizar, ya que de haberse utilizado una gramática descendente con PLY se hubiera tenido que hacer uso de la pila para podrida realizar ciertas acciones lo cual hubiera complicado más el proceso de creación del analizador.

En un inicio se agregaron tanto las palabras reservadas a utilizar como también las expresiones regulares que servirían posteriormente.

```
# Imports Librerias
from reportes import *

# Analisis Lexico
Reservadas = { 'create':'CREATE', 'database':'DATABASE', 'table': 'TABLE', 'replace':'REPLACE', 'if':'IF', 'exists':'EX

tokens = [ 'ID', 'PTCOMA', 'IGUAL', 'DECIMAL', 'ENTERO', 'PAR_A', 'PAR_C', 'PUNTO', 'COMA', 'CADENA1', 'CADENA2', 'BOOL

t_PTCOMA = r';'
t_PAR_A = r'\('
t_PAR_C = r'\)'
t_COMA = r'\,'
t_PUNTO = r'\.'
t_ASTERISCO = r'\*'
t_DOSPUNTOS =r'::'
t_SQRT2 = r'\|'
t_CBRT2 = r'\|\|'
t_AND2 = r'\&'
t_NOT2 = r'\~'
t_XOR = r'\#'
t_SH_LEFT = r'\<\<'
t_SH_RIGHT = r'\>\>'

#Comparision operators
t_IGUAL = r'\='
t_DESIGUAL = r'\!\='
t_DESIGUAL2 = r'\<\>'
t_MAYORIGUAL = r'\>\='
t_MENORIGUAL = r'\<\='
t_MAYOR = r'\>'
```

```
def t_DECIMAL(t):
    r'\d+\.\d+'
    try:
        t.value = float(t.value)
    except ValueError:
        print("Valor no es parseable a decimal %d",t.value)
        t.value = 0
    return t


def t_ENTERO(t):
    r'\d+'
    try:
       t.value = int(t.value)
    except ValueError:
       print('Int valor muy grande %d', t.value)
       t.value = 0
    return t

def t_BOOLEAN(t):
    r'(true|false)'
    mapping = {"true": True, "false": False}
    t.value = mapping[t.value]
    return t

def t_ID(t):
    r'[a-zA-Z_][a-zA-Z_0-9]*'
    t.type = Reservadas.get(t.value.lower(),'ID')
    return t
```

Posteriormente se importa tanto lex, como también los archivos descritos con anterioridad los cuales contienen las clases que almacenarán la información más importante de cada funcionalidad, también se definió la precedencia para su uso en posteriores operaciones

```
import ply.lex as lex
lexer = lex.lex()

# Analisis Sintactico
# Definición de la gramática

from expresiones import *
from instrucciones import *

# Precedencia de operadores
precedence = (
                ('right','NOT'),
                ('left', 'AND', 'OR'),
                ('left', 'IGUAL', 'DESIGUAL'),
                ('left', 'DESIGUAL2', 'MAYORIGUAL'),
                ('left', 'MENORIGUAL', 'MAYOR'),
                ('left', 'MENOR'),
                ('left', 'SUMA', 'RESTA'),
                ('left', 'ASTERISCO', 'DIVISION'),
                ('left', 'POTENCIA', 'MODULO'),
                ('right', 'UMENOS', 'USQRT2'),
                ('right', 'CBRT2', 'NOT2'),
                ('left', 'SQRT2', 'AND2'),
                ('left', 'XOR', 'SH_LEFT'),
                ('left', 'SH_RIGHT')
              )
```

Luego se procedió a la realización de múltiples funciones las cuales representan cada una de las producciones principales de la gramática a las cuales se les aplica en el apartado de las reglas semánticas el llamado a la clase que representa y el guardado de la información.

```
def p_init(t):
    'init           : l_sentencias'
    t[0] = t[1]

def p_l_sentencias1(t):
    'l_sentencias : l_sentencias sentencias'
    t[1].append(t[2])
    t[0] = t[1]

def p_l_sentencias2(t):
    'l_sentencias : sentencias'
    t[0] = [t[1]]

def p_lista_instrucciones(t):
    'sentencias : sentencia PTCOMA'
    t[0] = t[1]


def p_instruccion(t):
    '''sentencia : sentencia_ddl
                 | sentencia_dml'''
    t[0] = t[1]

def p_sentencia_ddl(t):
    '''sentencia_ddl : crear
                     | liberar'''
    t[0] = t[1]
```

```
def p_sentencia_dml(t):
    '''sentencia_dml : insertar
                     | actualizar
                     | eliminar
                     | seleccionH
                     | mostrar
                     | altert
                     | usar'''
    t[0] = t[1]
#NUEVO YO-----------------------------------------

def p_seleccionH1(t):
    '''seleccionH  : seleccionH UNION seleccionar
                   | seleccionH INTERSECT seleccionar
                   | seleccionH EXCEPT  seleccionar
                   | seleccionH UNION ALL  seleccionar
                   | seleccionH INTERSECT ALL seleccionar
                   | seleccionH EXCEPT ALL seleccionar
                   | PAR_A seleccionH PAR_C
                   | seleccionar'''
    if len(t) ==  4:
        if t[2].lower() == "union":
            t[1].append(t[3])
            t[0] = t[1]
        elif t[2].lower() == "intersect":
            t[1].append(t[3])
            t[0] = t[1]
        elif t[2].lower() == "except":
            t[1].append(t[3])
```

De ese mismo modo haciendo uso de las clases definidas estas se adaptaron según la situación, almacenando de distinta manera la información según fuera necesario por la producción.

```
def p_value_from(t):
     '''value_from : tabla_name
                   | PAR_A seleccionar PAR_C ID
                   | PAR_A seleccionar PAR_C AS ID'''
     if len(t) == 2:
          t[0] = Valor_From(t[1],None,None)
     elif len(t) == 5:
          t[0] = Valor_From(None,t[2],Operando_ID(t[4]))
     else:
          t[0] = Valor_From(None,t[2],Operando_ID(t[5]))

def p_tabla_name(t):
     '''tabla_name : ID
                   | ID ID'''
     if len(t) == 2:
          t[0] = t[1]
     else:
          t[0] = t[1]+' '+t[2]
```

```
def p_expresion_par(t):
    '''E : PAR_A exp PAR_C'''
    t[0] = t[2]


def p_crear(t):
    '''crear : CREATE reemplazar DATABASE verificacion ID propietario modo
             | CREATE TABLE ID PAR_A columnas PAR_C herencia
             | CREATE TYPE ID AS ENUM PAR_A lista_exp PAR_C'''
    if(t[3].lower()=='database'):
        t[0]=CrearBD(t[2], t[4], Operando_ID(t[5]), t[6], t[7])
    else:
        if(t[2].lower()=='table'):
            t[0]=CrearTabla(Operando_ID(t[3]),t[7],t[5])
        else:
            t[0]=CrearType(Operando_ID(t[3]),t[7])
```

Para el manejo de los distintos tipos de expresión también sé
tuvo que dividir la gramática de manera que fuera posible idéntica cada uno de los tipos de expresión definidos.

```
def p_listaexp(t):
     '''lista_exp : lista_exp COMA exp
                  | exp'''
     if(len(t) == 4):
          t[1].append(t[3])
          t[0] = t[1]
     else:
          t[0] = [t[1]]

def p_expresiones(t):
     '''exp : exp_log
            | exp_rel
            | exp_ar
            | exp_select
            | expresion_patron
            | E'''
     t[0] = t[1]

def p_expresion_logica(t):
     '''exp_log : NOT exp
                | exp AND exp
                | exp OR exp'''
     if(len(t) == 4):
```

```
def p_expresion_relacional(t):
    '''exp_rel : exp toperador exp'''
    if t[2] == "=":
        t[0] = Operacion_Relacional(t[1],t[3],OPERACION_RELACIONAL.IGUAL)
    elif t[2] == "!=":
        t[0] = Operacion_Relacional(t[1],t[3],OPERACION_RELACIONAL.DIFERENTE)
    elif t[2] == "<>":
        t[0] = Operacion_Relacional(t[1],t[3],OPERACION_RELACIONAL.DIFERENTE)
    elif t[2] == ">=":
        t[0] = Operacion_Relacional(t[1],t[3],OPERACION_RELACIONAL.MAYORIGUALQUE)
    elif t[2] == "<=":
        t[0] = Operacion_Relacional(t[1],t[3],OPERACION_RELACIONAL.MENORIGUALQUE)
    elif t[2] == ">":
        t[0] = Operacion_Relacional(t[1],t[3],OPERACION_RELACIONAL.MAYOR_QUE)
    elif t[2] == "<":
        t[0] = Operacion_Relacional(t[1],t[3],OPERACION_RELACIONAL.MENOR_QUE)
```

```
def p_expresion_aritmetica(t):
    '''exp_ar : exp SUMA exp
              | exp RESTA exp
              | exp ASTERISCO exp
              | exp DIVISION exp
              | exp POTENCIA exp
              | exp MODULO exp'''
    if t[2] == "+":
        t[0] = Operacion_Aritmetica(t[1],t[3],OPERACION_ARITMETICA.MAS)
    elif t[2] == "-":
        t[0] = Operacion_Aritmetica(t[1],t[3],OPERACION_ARITMETICA.MENOS)
    elif t[2] == "*":
        t[0] = Operacion_Aritmetica(t[1],t[3],OPERACION_ARITMETICA.POR)
    elif t[2] == "/":
        t[0] = Operacion_Aritmetica(t[1],t[3],OPERACION_ARITMETICA.DIVIDIDO)
    elif t[2] == "^":
        t[0] = Operacion_Aritmetica(t[1],t[3],OPERACION_ARITMETICA.POTENCIA)
    elif t[2] == "%":
        t[0] = Operacion_Aritmetica(t[1],t[3],OPERACION_ARITMETICA.MODULO)
```

Por último se definió tanto el método para detectar errores sintácticos y recuperación, como también la manera en la cual se exporta el analizador.

```
def p_error(t):
    if(t!=None):
        print("Error sintactico en: '%s'" % t.value)
        Error_Sin.append("Error sintactico: Lexema: "+str(t.value)+ " Fila: "+str(t.lineno))


        while(True):
            tk = parser.token()
            if(tk==None):
                break
            elif(tk.type=="PTCOMA"):
                break

        #parser.errok()
        parser.restart()
        #return tk


Error_Lex = []
Error_Sin = []

import ply.yacc as yacc
parser = yacc.yacc()

def parse(input) :
    global Error_Lex
    global Error_Sin
    Reporte_Errores(Error_Lex,Error_Sin)
    return parser.parse(input)
```

AST

Esta sería la clase principal en la cual se recorre el listado de instrucciones generadas por el analizador y de las cuales se obtienen los datos almacenados en las distintas clases definidas para su ejecución.

Cada uno de los métodos definidos cumplen con la función de obtener los datos almacenados en la clase en especifico y poder realizar su función.

```
#---------variables globales
listaInstrucciones = []
listaTablas = [] #guarda las cabeceras de las tablas creadas
outputTxt = [] #guarda los mensajes a mostrar en consola
outputTS = [] #guarda el reporte tabla simbolos
baseActiva = "" #Guarda la base temporalmente activa
listaFunciones=[]
Errores_Semanticos = []
ListaResExpID=[]
listaProcedure = []
#--------Ejecucion Datos temporales-----------
def reiniciarVariables(): ···

def agregarFuncion(nombre,tipo,contenido,parametros): ···

def buscarFuncion(nombre): ···


def addProcedure(name,cuerpo,parametros): ···

def findProcedure(nombre): ···

def validarTipoF(T): ···

def eliminarFuncion(nombre): ···

def eliminarProcedure(name): ···

def EliminarIndice(instr,ts): ···
```

```
def obtenerPKexp(expresion,nombres,ts): ···

def getpks(baseAc,nombret): ···

def llaves_tabla(exp,llaves,ts): ···

def indice_llaves(llaves,baseAc,ntabla): ···

#---------Ejecucion Funciones EDD-------------
def crear_BaseDatos(instr,ts): ···

def eliminar_BaseDatos(instr,ts): ···

def mostrar_db(instr,ts): ···

def eliminar_Tabla(instr,ts): ···

def seleccion_db(instr,ts): ···

#---------pendientes-----------------------
def crear_Tabla(instr,ts): ···

def crear_Type(instr,ts): ···

def insertar_en_tabla(instr,ts): ···

def update_register(exp,llaves,ts,baseAc,tablenm,nameC,valor): ···

def actualizar_en_tabla(instr,ts): ···

def eliminar_de_tabla(instr,ts): ···
```

Los principales métodos definidos para esta clase son el resolver operación el cual recibe cualquier tipo de expresión y devuelve el resultado calculado.
Y el procesar instrucción el cual recorre cada una de las instrucciones de la lista y por medio de la validaciones colocadas direcciona cada instrucción al método encargado de para procesar su información y ejecutarla.

```
def resolver_operacion(operacion,ts):
    if isinstance(operacion, Operacion_Logica_Unaria):
        op = resolver_operacion(operacion.op, ts)
        if isinstance(op, bool):
            return not(op)
        else:
            print('Error: No se permite operar los tipos involucrados')
    elif isinstance(operacion, Operacion_Logica_Binaria):
        op1 = resolver_operacion(operacion.op1,ts)
        op2 = resolver_operacion(operacion.op2,ts)
        if isinstance(op1, bool) and isinstance(op2, bool):
            if operacion.operador == OPERACION_LOGICA.AND: return op1 and op2
            elif operacion.operador == OPERACION_LOGICA.OR: return op1 or op2
        else:
            print('Error: No se permite operar los tipos involucrados')
    elif isinstance(operacion, Operacion_Relacional):
        op1 = resolver_operacion(operacion.op1,ts)
        op2 = resolver_operacion(operacion.op2,ts)
        if isinstance(op1, (int,float)) and isinstance(op2, (int,float)):
            if operacion.operador == OPERACION_RELACIONAL.IGUAL: return op1 == op2
            elif operacion.operador == OPERACION_RELACIONAL.DIFERENTE: return op1 != op2
            elif operacion.operador == OPERACION_RELACIONAL.MAYORIGUALQUE: return op1 >= op2
            elif operacion.operador == OPERACION_RELACIONAL.MENORIGUALQUE: return op1 <= op2
            elif operacion.operador == OPERACION_RELACIONAL.MAYOR_QUE: return op1 > op2
            elif operacion.operador == OPERACION_RELACIONAL.MENOR_QUE: return op1 < op2
        elif isinstance(op1, (str)) and isinstance(op2, (str)):
            if operacion.operador == OPERACION_RELACIONAL.IGUAL: return op1 == op2
            elif operacion.operador == OPERACION_RELACIONAL.DIFERENTE: return op1 != op2
            elif operacion.operador == OPERACION_RELACIONAL.MAYORIGUALQUE: return op1 >= op2
            elif operacion.operador == OPERACION_RELACIONAL.MENORIGUALQUE: return op1 <= op2
            elif operacion.operador == OPERACION_RELACIONAL.MAYOR_QUE: return op1 > op2
```

```
        else:
            print('Error: No se permite operar los tipos involucrados')
    elif isinstance(operacion, Operacion_Aritmetica):
        op1 = resolver_operacion(operacion.op1,ts)
        op2 = resolver_operacion(operacion.op2,ts)
        if isinstance(op1, (int,float)) and isinstance(op2, (int,float)):
            if operacion.operador == OPERACION_ARITMETICA.MAS: return op1 + op2
            elif operacion.operador == OPERACION_ARITMETICA.MENOS: return op1 - op2
            elif operacion.operador == OPERACION_ARITMETICA.POR: return op1 * op2
            elif operacion.operador == OPERACION_ARITMETICA.DIVIDIDO: return op1 / op2
            elif operacion.operador == OPERACION_ARITMETICA.POTENCIA: return op1 ** op2
            elif operacion.operador == OPERACION_ARITMETICA.MODULO: return op1 % op2
        else:
            print('Error: No se permite operar los tipos involucrados')
    elif isinstance(operacion, Operacion_Especial_Binaria):
        op1 = resolver_operacion(operacion.op1,ts)
        op2 = resolver_operacion(operacion.op2,ts)
        if isinstance(op1, int) and isinstance(op2, int):
            if operacion.operador == OPERACION_ESPECIAL.AND2: return op1 & op2
            elif operacion.operador == OPERACION_ESPECIAL.OR2: return op1 | op2
            elif operacion.operador == OPERACION_ESPECIAL.XOR: return op1 ^ op2
            elif operacion.operador == OPERACION_ESPECIAL.DEPDER: return op1 >> op2
            elif operacion.operador == OPERACION_ESPECIAL.DEPIZQ: return op1 << op2
        else:
            print('Error: No se permite operar los tipos involucrados')
    elif isinstance(operacion, Operacion_Especial_Unaria):
        op = resolver_operacion(operacion.op,ts)
        if isinstance(op, (int,float)):
            if operacion.operador == OPERACION_ESPECIAL.SQRT2: return op ** (1/2)
            elif operacion.operador == OPERACION_ESPECIAL.CBRT2: return op ** (1/3)
            elif operacion.operador == OPERACION_ESPECIAL.NOT2:
                if isinstance(op, int): return ~op
```

```python
def procesar_instrucciones(instrucciones, ts) :
    ## lista de instrucciones recolectadas
    global Errores_Semanticos
    global listaInstrucciones
    listaInstrucciones  = instrucciones
    if instrucciones is not None:
        for instr in instrucciones :
            if isinstance(instr, CrearBD) : crear_BaseDatos(instr,ts)
            elif isinstance(instr, CrearTabla) : crear_Tabla(instr,ts)
            elif isinstance(instr, CrearType) : crear_Type(instr,ts)
            elif isinstance(instr, EliminarDB) : eliminar_BaseDatos(instr,ts)
            elif isinstance(instr, EliminarTabla) : eliminar_Tabla(instr,ts)
            elif isinstance(instr, Insertar) : insertar_en_tabla(instr,ts)
            elif isinstance(instr, Actualizar) : actualizar_en_tabla(instr,ts)
            elif isinstance(instr, Eliminar) : eliminar_de_tabla(instr,ts)
            elif isinstance(instr, DBElegida) : seleccion_db(instr,ts)
            elif isinstance(instr, MostrarDB) : mostrar_db(instr,ts)
            elif isinstance(instr, ALTERDBO) : AlterDBF(instr,ts)
            elif isinstance(instr, ALTERTBO) : AlterTBF(instr,ts)
            elif isinstance(instr, MostrarTB) : Mostrar_TB(instr,ts)
            else:
                for val in instr:
                    if(isinstance (val,SELECT)):
                        if val.funcion_alias is not None:
                            ejecutar_select(val,ts)
                        else:
                            select_table(val,ts)
                    else : print('Error: instrucción no válida')
    else: agregarMensjae('error','El arbol no se genero debido a un error en la entrada','')
    Reporte_Errores_Sem(Errores_Semanticos)
```

C3D

Para instrucción trabajada tanto durante la fase 1 del proyecto como cambien para las nuevas incorporadas para la segunda fase fue generado su equivalente a tres direcciones el cual es contenido dentro de un archivo que se genera con cada ejecución de instrucciones de la aplicación, dicho archivo puede ser compilado de igual manera y la ejecución de este es equivalente en resultados a la última ejecución realizada por la aplicación. A continuación se mostrarán algunos ejemplos de la traducción a código de tres direcciones como también de la pila utilizada para el manejo de valores en memoria.

```
def PCreateDatabase(nombreBase,result):
    reinicar_contOP()
    txt="\t#Create DataBase\n"
    txt+="\tt"+str(numT())+"='"+nombreBase+"'\n"
    varT="t"+str(numT())
    txt+="\t"+varT+"=CD3.EReplace()\n"
    txt+="\tif("+varT+"):\n"
    txt+="\t\tgoto .dropDB"+str(contT)+"\n"
    txt+="\t\tlabel.dropDB"+str(contT)+"\n"
    txt+="\t\tCD3.EDropDatabase()"

    replac=False

    if(result==1):
        #'eliminar, luego crear'
        replac=True
        agregarInstr(replac,txt)#agregar replace
        agregarInstr(nombreBase,'')#agregar Drop
    else:
        agregarInstr(replac,txt)#agregar replace
    #crear tabla
    txt3="\tCD3.ECreateDatabase()\n"
    agregarInstr(nombreBase,txt3)#agregar create

def PDropDatabase(nombreBase):
    reinicar_contOP()
    txt="\t#Drop DataBase\n"
    txt+="\tt"+str(numT())+"='"+nombreBase+"'\n"
    txt+="\tCD3.EDropDatabase()\n"
    agregarInstr(nombreBase,txt)
```

```
def PSelectTablas(nombreTabla,cabeceras,filas,cantidadRegistros):
    reinicar_contOP()
    registros=[cantidadRegistros]
    txt="\t#Select table\n"
    varT="t"+str(numT())
    txt+="\t"+varT+"="+str(nombreTabla)+"\n"
    varC="t"+str(numT())
    txt+="\t"+varC+"="+str(cabeceras)+"\n"
    varR="t"+str(numT())
    txt+="\t"+varR+"=CD3.ECantidadRegistros()\n"
    varfilas="t"+str(numT())
    txt+="\t"+varfilas+"="+str(filas)+"\n"
    varCont="t"+str(numT())
    txt+="\t"+varCont+"=0\n"
    txt+="\tprint(\'tablas seleccionadas:\',str("+varT+"))\n"
    txt+="\tprint(\'cabeceras:\',str("+varC+"))\n"
    txt+="\tlabel.mostrarFila"+str(contT)+"\n"
    txt+="\tif("+varCont+"<"+varR+"):"+"\n"
    txt+="\t\tprint(\'\t\',"+varfilas+"["+varCont+"])\n"
    txt+="\t\t"+varCont+"="+varCont+"+1\n"
    txt+="\t\tgoto.mostrarFila"+str(contT)+"\n"
    agregarInstr(registros,txt)

def PUseDatabase(nombreBase):
    reinicar_contOP()
    txt="\t#Use Database\n"
    txt+="\tt"+str(numT())+"='"+nombreBase+"'\n"
    txt+="\tCD3.EUseDatabase()\n"
    agregarInstr(nombreBase,txt)
```

```python
def ECreateDatabase():
    cargarMemoria()
    if(len(listaMemoria)>0):
        print("base de datos creada:",listaMemoria[0])
        EDD.createDatabase(listaMemoria[0])
        listaMemoria.pop(0)

def EDropDatabase():
    cargarMemoria()
    #Llamar la funcion de EDD
    if(len(listaMemoria)>0):
        print("base de datos eliminada:",listaMemoria[0])
        EDD.dropDatabase(listaMemoria[0])
        listaMemoria.pop(0)

def EUseDatabase():
    cargarMemoria()
    #Llamar la funcion de EDD
    if(len(listaMemoria)>0):
        print("selecionada base de datos:",listaMemoria[0])
        listaMemoria.pop(0)
```

```python
def EInsert():
    cargarMemoria()
    #Llamar la funcion de EDD
    if(len(listaMemoria)>0):
        Data_insert=listaMemoria[0]
        EDD.insert(Data_insert[0],Data_insert[1],Data_insert[2])
        print("insert en tabla ",Data_insert[1]," \n\tvalores ",Data_insert[2])
        listaMemoria.pop(0)

def EObtenerTabla():
    cargarMemoria()
    #Llamar la funcion de EDD
    if(len(listaMemoria)>0):
        get_tb=listaMemoria[0]
        result=EDD.showTables(get_tb[0])
        if get_tb[1] in result:
            listaMemoria.pop(0)
            return True
    return False
```

Estos métodos corresponden a la creación del código en tres direcciones para cada instrucción ejecutada por la aplicación dentro estos mismos métodos se detectan las posibles optimizaciones aplicando las reglas de mirilla haciendo uso de estas es posible genera una salida de código de tres direcciones más óptima.

```
def PCreateProcedure(nombre,contenido,parametros,reemplazada):
    reinicar_contOP()
    txt="\t#Crear Stored  Procedure\n"
    txt+="\tt"+str(numT())+"='"+nombre+"'\n"
    txt+="\tt"+str(numT())+"="+str(parametros)+"\n"
    txt+="\tt"+str(numT())+"="+str(reemplazada)+" #Reemplazar funcion\n"
    varT="t"+str(numT())
    txt+="\t"+varT+"=CD3.ECreateProcedure()\n"
    #------------------optimizacion--------------
    regla="3 - se nego condicion para poder eliminar etiqueta"
    msg="if("+varT+"):\n"
    msg+="\tgoto .bodyFun"+str(contT)+"\n"
    msg+="else:\n"
    msg+="\tgoto .endFun"+str(contT)
    #--------------------------------------------
    txt2="\tif("+varT+"==False):\n"
    fin=contT
    txt2+="\t\tgoto .endProc"+str(fin)+"\n"
    varT="t"+str(numT())
    #txt2+="\t"+varT+"=CD3.ExecuteProc()\n"
    txt2+="\tif("+varT+"==False):\n"
    txt2+="\t\tgoto .endProc"+str(fin)+"\n"
    #declaraciones
    txt2+="\tlabel.decProc"+str(contT)+" #Declaraciones Procedure\n"
    for i in contenido.declaraciones:
        txt2+="\tt"+str(numT())+"='"+i.nombre+"'\n"
        print("CD3------>",i)
    #contenido
    txt2+="\tlabel.bodyFun"+str(contT)+" #Contenido Procedure\n"
    #txt2+=PInstrFun(contenido.contenido)+"\n"
    txt2+="\tlabel.endProc"+str(fin)+"\n"
```

Una vez se a encontrada un bloque de código el cual se puede optimizar se aplica la regla correspondiente y posteriormente es agregado al reporte de optimizaciones tanto la regla utilizada como también el código antes y después de ser optimizado, esto para que sea más clara la lectura de la manera en que el código mejoro.

Ya que se está trabajando con código de tres direcciones no es posible enviar valores como parámetro a las funciones utilizadas durante la ejecución para solucionar este problema se hizo uso de una pila en la cual se guardan todos los valores necesarios para cada instrucción para que al momento de ser ejecutada pueda obtener la información y realizar su función.

```
[
    false,
    "dbfase2",
    "dbfase2",
    [
        "myfuncion",
        "text",
        "<temporal.contenido_run object at 0x000002CEE89747F0>",
        [
            "texto"
        ],
        false
    ],
    false,
    [
        "dbfase2",
        "tbproducto",
        4
    ],
    [
        "dbfase2",
        "tbproducto",
        [
            0
        ]
    ],
    [
        "dbfase2",
        "tbcalificacion",
        3
    ],
    [
```

```
#Funciones para ejecutar codigo de 3 direcciones
def cargarMemoria():
    #Lectura memoria temp
    global memoriTrue
    global listaMemoria
    if(memoriTrue==0):
        print("----------Verificando Heap-------")
        memoriTrue=1
        with open('memoria.json') as file:
            data = json.load(file)
        listaMemoria=data
```

Conclusión

Luego de esta descripción general del proyecto se espera que el usuario haya podido resolver sus dudas acerca de cómo es que el proyecto fue desarrollado y su funcionamiento en general.