MANUAL TÉCNICO

Requisitos:

- Cualquier computadora con windows o linux.
- Instalar python versión 3.6 o superior
- Instalar librerías con pip: ply, graphviz,prettytable.
- Instalar Pycharm o Visual Studio Code

Funcionalidad

PLY y GRAMÁTICA:

La gramática ascendente definida BNF.md se implemento en ply, la clase gammar2.py se compone de 2 apartados importantes. El primero el **análisis léxico**, que se compone de palabras reservadas que se enlistan en un arreglo y de algunas funciones que trabajan con expresiones regulares son capaces de retornar tokens.

Ejemplo para definir un identificador con una expresión regular :

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

La otra parte se compone de el análisis sintáctico, aquí se implementan las reglas (acorde a la construcción de un AST enfocado a una base de datos) y las producciones acorde a la gramática:

Ejemplo de una condición:

```
def p_cond(p):
    """
    cond    :   id MAYOR valortipo
            |   id MENOR valortipo
            |   id IGUAL valortipo
            |   id MENOR_IGUAL valortipo
            |   id MAYOR_IGUAL valortipo
    """
    p[0] = inst.cond(p[1],p[2],p[3])
    insertProduction(p.slice, len(p.slice))
```

También se destacan una dos producciones de error en donde cae, valga la redundancia cualquier error según la documentación en esta ocasión utilizamos utilizamos un método alterno para la recuperación de errores, según la documentación de ply:
“This function simply discards the bad token and tells the parser that the error was ok.” Lo que se traduce en que se descarta el error y que el token está bien, el código se recupera en la siguiente producción válida.

```
def p_error(t):
    if t:
        descript = 'error sintactico en el token ' + str(t.type)
        linea = str(t.lineno)
        columna = str(find_column(t))
        nuevo_error = CError(linea,columna,descript,'Sintactico')
        insert_error(nuevo_error)
        parser.errok()
    else:
        print("Syntax error at EOF")
    return
```

CLASES ABSTRACTAS

Las diferentes clases abstractas moldean las propiedades y atributos de una base de datos, cada clase cuenta con su método abstracto y todas las clases son capaces de invocar siempre y cuando se herede de la clase principal.

Ejemplo de una clase abstracta:

```
class select(query):

    def __init__(self, distinct=False, select_list=[], table_expression=[], condition=[], group=False, ha
        self.distinct = distinct
        self.select_list = select_list
        self.table_expression = table_expression
        self.condition = condition
        self.group = group
        self.having = having
        self.orderby = orderby
        self.limit = limit
        self.offset = offset
        if having is not None and condition is not None:
            self.condition.append(having)
```

FASE 2

Para la fase 2 se pidió realizar el código 3 direcciones, teniendo el análisis léxico, sintáctico y la ejecución haciendo uso de un análisis semántico, se procedió a realizar el código 3d mediante una salida en Python utilizando la librería goto, para simular las etiquetas en el lenguaje py.

```
1   from graphviz import Graph
2   import os
3   import re
4   import OptimizarObjetos as obj #Objetos utilizados en el optimizador mirilla
5
6   CodigoOptimizado = [] #Se guarda el codigo optimizado resultante para el reporte
7   ResultadoFinal = [] #Se guarda el resultado final de la optimizacion
8
9   class Optimizador:
10      def __init__(self, Asignaciones = []):
11          #Asignaciones = Pila de instrucciones del codigo de tres direcciones
12          self.Asignaciones = Asignaciones
13
```

Para poder usar correctamente la librería goto, se tuvieron que fabricar funciones nativas para asi ejecutar las instrucciones mediante una simulación.

```
14      def ejecutar(self):
15          global CodigoOptimizado
16          global ResultadoFinal
17          CodigoOptimizado = []
18          ResultadoFinal = []
19          for objeto in self.Asignaciones:
20              if isinstance(objeto, obj.Asignacion):
21                  if self.Regla_8_9(objeto) or self.Regla_10_11(objeto):
22                      "NO SE AGREGA LA INSTRUCCION AL RESULTADO"
23                  elif self.Regla_12_13(objeto) or self.Regla_14_15(objeto) or self.Regla_16(objeto) or self.Regla_17_18(objeto):
24                      "SE AGREGA LA OPTIMIZACION AL RESULTADO"
25                      ResultadoFinal.append(CodigoOptimizado[-1].resultado)
26                  else:
27                      "SE AGREGA LA INSTRUCCION ORIGINAL AL RESULTADO"
28                      instruccion = objeto.indice + " = " + objeto.operador1 + " " + objeto.signo + " " + objeto.operador2
29                      ResultadoFinal.append(instruccion)
30              else:
31                  "NO ES OBJETO ASIGNACION ASI QUE SOLO SE AGREGA AL RESULTADO"
32                  ResultadoFinal.append(objeto)
33          self.GenerarReporte()
34          print("----------------------OPTIMIZACIONES--------------------------------")
35          for elem in CodigoOptimizado:
36              print(elem.resultado)
37          print("----------------------RESULTADO OPTIMIZACION-----------------------")
38          for elem in ResultadoFinal:
39              print(elem)
```

Aparte de traducir el código a de ejecución a código intermedio, también se requiere de un proceso que le sigue a este, el cual es la optimización, el cual por medio de reglas, reduce el código teniendo el mismo resultado.

Regla1:

```
40
41      #Metodos para realizar la optimizacion segun la regla.
42      #Retornan False si no se cumple la regla de optimizacion y True si se cumple y se optimiza
43      def Regla_1(self, asignacion, listaasignaciones):
44          "Regla 1"
45          global CodigoOptimizado
46          indice = asignacion.indice
47          op1 = asignacion.operador1
48          op2 = asignacion.operador2
49          signo = asignacion.signo
50          for elem in listaasignaciones:
51              if elem.indice == op1 and elem.operador1 == indice:
52                  "SE REALIZA EL CAMBIO"
53
54          return False
```

Es de suma importancia tener en cuenta las reglas que se utilizan para la optimización por el método de mirilla.

```
56      def Regla_2(self):
57          "Regla 2"
58          return False
59
60      def Regla_3(self):
61          "Regla 3"
62          return False
63
64      def Regla_4(self):
65          "Regla 4"
66          return False
67
68      def Regla_5(self):
69          "Regla 5"
70          return False
71
72      def Regla_6(self):
73          "Regla 6"
74          return False
75
76      def Regla_7(self):
77          "Regla 7"
78          return False
79
```

Regla 8 y 9 de optimización:
Metodo para realizar la eliminación de una instrucción con suma o resta con valor igual a 0.

```
80      def Regla_8_9(self, asignacion):
81          "(+)Regla 8 y (-)Regla 9 - Eliminacion de codigo"
82          global CodigoOptimizado
83          indice = asignacion.indice
84          op1 = asignacion.operador1
85          op2 = asignacion.operador2
86          signo = asignacion.signo
87          if (indice == op1 and op2 == '0' and signo == '+') or (indice == op2 and op1 == '0' and signo == '+'):
88              NuevoObjeto = obj.Optimizado("Regla 8", indice + " = " + op1 + " " + signo + " " + op2, "Se elimina la instruccion")
89              CodigoOptimizado.append(NuevoObjeto)
90              return True
91          elif indice == op1 and op2 == '0' and signo == '-':
92              NuevoObjeto = obj.Optimizado("Regla 9", indice + " = " + op1 + " " + signo + " " + op2, "Se elimina la instruccion")
93              CodigoOptimizado.append(NuevoObjeto)
94              return True
95          else:
96              return False
```

Regla 10 y 11 de optimización:
Metodo para realizar la eliminación de una instrucción con multiplicación o división con valor igual a 1.

```
98      def Regla_10_11(self, asignacion):
99          "(*)Regla 10 y (/)Regla 11"
100         global CodigoOptimizado
101         indice = asignacion.indice
102         op1 = asignacion.operador1
103         op2 = asignacion.operador2
104         signo = asignacion.signo
105         if (indice == op1 and op2 == '1' and signo == '*') or (indice == op2 and op1 == '1' and signo == '*'):
106             NuevoObjeto = obj.Optimizado("Regla 10", indice + " = " + op1 + " " + signo + " " + op2, "Se elimina la instruccion")
107             CodigoOptimizado.append(NuevoObjeto)
108             return True
109         elif indice == op1 and op2 == '1' and signo == '/':
110             NuevoObjeto = obj.Optimizado("Regla 11", indice + " = " + op1 + " " + signo + " " + op2, "Se elimina la instruccion")
111             CodigoOptimizado.append(NuevoObjeto)
112             return True
113         else:
114             return False
```

Regla 12 y 13

```
    def Regla_12_13(self, asignacion):
        "(+)Regla 12 y (-)Regla 13"
        global CodigoOptimizado
        indice = asignacion.indice
        op1 = asignacion.operador1
        op2 = asignacion.operador2
        signo = asignacion.signo
        if op1 == '0' and op2 != '0' and signo == '+' and op2 != indice:
            NuevoObjeto = obj.Optimizado("Regla 12", indice + " = " + op1 + " " + signo + " " + op2, indice + " = " + op2)
            CodigoOptimizado.append(NuevoObjeto)
            return True
        elif op2 == '0' and op1 != '0' and signo == '+' and op1 != indice:
            NuevoObjeto = obj.Optimizado("Regla 12", indice + " = " + op1 + " " + signo + " " + op2, indice + " = " + op1)
            CodigoOptimizado.append(NuevoObjeto)
            return True
        elif op1 != '0' and op2 == '0' and signo == '-' and op1 != indice:
            NuevoObjeto = obj.Optimizado("Regla 13", indice + " = " + op1 + " " + signo + " " + op2, indice + " = " + op1)
            CodigoOptimizado.append(NuevoObjeto)
            return True
        else:
            return False
```

Regla 14 y 15

```
def Regla_14_15(self, asignacion):
    "(*)Regla 14 y (/)Regla 15"
    global CodigoOptimizado
    indice = asignacion.indice
    op1 = asignacion.operador1
    op2 = asignacion.operador2
    signo = asignacion.signo
    if op1 == '1' and op2 != '1' and signo == '*' and op2 != indice:
        NuevoObjeto = obj.Optimizado("Regla 14", indice + " = " + op1 + " " + signo + " " + op2, indice + " = " + op2)
        CodigoOptimizado.append(NuevoObjeto)
        return True
    elif op2 == '1' and op1 != '1' and signo == '*' and op1 != indice:
        NuevoObjeto = obj.Optimizado("Regla 14", indice + " = " + op1 + " " + signo + " " + op2, indice + " = " + op1)
        CodigoOptimizado.append(NuevoObjeto)
        return True
    elif op1 != '1' and op2 == '1' and signo == '/' and op1 != indice:
        NuevoObjeto = obj.Optimizado("Regla 15", indice + " = " + op1 + " " + signo + " " + op2, indice + " = " + op1)
        CodigoOptimizado.append(NuevoObjeto)
        return True
    else:
        return False

```

Regla 16

```
def Regla_16(self, asignacion):
    "Regla 16"
    global CodigoOptimizado
    indice = asignacion.indice
    op1 = asignacion.operador1
    op2 = asignacion.operador2
    signo = asignacion.signo
    if op1 == '2' and not isinstance(op2, int) and signo == '*' and op2 != indice:
        NuevoObjeto = obj.Optimizado("Regla 16", indice + " = " + op1 + " " + signo + " " + op2, indice + " = " + op2 + " " + "+" + " " + op2)
        CodigoOptimizado.append(NuevoObjeto)
        return True
    elif op2 == '2' and not isinstance(op1, int) and signo == '*' and op1 != indice:
        NuevoObjeto = obj.Optimizado("Regla 16", indice + " = " + op1 + " " + signo + " " + op2, indice + " = " + op1 + " " + "+" + " " + op1)
        CodigoOptimizado.append(NuevoObjeto)
        return True
    else:
```

Regla 17 y18

```
178     def Regla_17_18(self, asignacion):
179         "(*)Regla 17 y (/)Regla18"
180         global CodigoOptimizado
181         indice = asignacion.indice
182         op1 = asignacion.operador1
183         op2 = asignacion.operador2
184         signo = asignacion.signo
185         if op1 == '0' and signo == '*' and op2 != indice:
186             NuevoObjeto = obj.Optimizado("Regla 17", indice + " = " + op1 + " " + signo + " " + op2, indice + " = 0")
187             CodigoOptimizado.append(NuevoObjeto)
188             return True
189         elif op2 == '0' and signo == '*' and op1 != indice:
190             NuevoObjeto = obj.Optimizado("Regla 17", indice + " = " + op1 + " " + signo + " " + op2, indice + " = 0")
191             CodigoOptimizado.append(NuevoObjeto)
192             return True
193         elif op1 == '0' and signo == '/' and op2 != indice:
194             NuevoObjeto = obj.Optimizado("Regla 18", indice + " = " + op1 + " " + signo + " " + op2, indice + " = 0")
195             CodigoOptimizado.append(NuevoObjeto)
196             return True
197         else:
198             return False
```

Metodo para genrar el reporte del resultado final junto con el archivo el cual contiene el código generado después de optimizar.

```
200     def GenerarReporte(self):
201         "Generar el reporte en graphviz y el archivo de optimizacion"
202         #PARA EL ARCHIVO
203         Nombre = "Salidas/CodigoOptimizado.py"
204         texto = ""
205         for elem in ResultadoFinal:
206             texto += elem + "\n"
207         try:
208             os.makedirs(os.path.dirname(Nombre), exist_ok=True)
209             with open(Nombre, "w") as f:
210                 f.write(texto)
211         except:
212             print("No se pudo generar el archivo del codigo generado")
213         #PARA REPORTE
```

Optimizacion del código:

```python
import hashlib
from datetime import date
from variables import tabla as ts
from variables import NombreDB
from variables import cont as ncont

import tablaDGA as TAS

import mathtrig as mt
import reportError as errores
#from Interfaz import lista
funciones = []
objopt = []
cont = ncont
class pl():
    'Clase abstacta'

def deleteF(name):

    name = name +'():'
    for i in range(len(funciones)):
        x = funciones[i].split(" ")
        print( 'tengo que eliminar la posicion '+ str(i) +' ya que elimine '+ str(x[1]))
        funciones.pop(i)
        break

```

Método para realizar la optimización de la declaración, recibimiento así el tipo y la expresión como parámetros más importantes.

```python
class declaration(pl):
    def __init__(self,id,constant,tipo,collate,notnull,exp):
        self.id = id
        self.constant = constant
        self.tipo = tipo
        self.collate = collate
        self.notnull = notnull
        self.exp = exp
        self.traduccion = None

    def c3d(self):
        if self.traduccion == None:
```

```
            c3d += '\tambitoFuncion =  ts.buscarIDF()\n'
            c3d += '\tNuevoSimbolo = TAS.Simbolo(cont,\''+str(self.id)+'\',TAS.TIPO.SMALLINT,ambitoFuncion,None, None, None, None, None, None, None ,None,None,'+valor+', '+col
            c3d += '\tts.agregar(NuevoSimbolo)\n'
            c3d += '\tcont+=1\n'

        elif self.tipo == 'INTEGER':
            c3d += '\tambitoFuncion =  ts.buscarIDF()\n'
            c3d += '\tNuevoSimbolo = TAS.Simbolo(cont,\''+str(self.id)+'\',TAS.TIPO.INTEGER,ambitoFuncion,None, None, None, None, None, None, None ,None,None,'+valor+', '+col+
            c3d += '\tts.agregar(NuevoSimbolo)\n'
            c3d += '\tcont+=1\n'
        elif self.tipo == 'BIGINT':
            c3d += '\tambitoFuncion =  ts.buscarIDF()\n'
            c3d += '\tNuevoSimbolo = TAS.Simbolo(cont,\''+str(self.id)+'\',TAS.TIPO.BIGINT,ambitoFuncion,None, None, None, None, None, None, None ,None,None,'+valor+', '+col+'
            c3d += '\tts.agregar(NuevoSimbolo)\n'
            c3d += '\tcont+=1\n'
        elif self.tipo == 'DECIMAL':
            c3d += '\tambitoFuncion =  ts.buscarIDF()\n'
            c3d += '\tNuevoSimbolo = TAS.Simbolo(cont,\''+str(self.id)+'\',TAS.TIPO.DECIMAL,ambitoFuncion,None, None, None, None, None, None, None ,None,None,'+valor+', '+col+
            c3d += '\tts.agregar(NuevoSimbolo)\n'
            c3d += '\tcont+=1\n'
        elif self.tipo == 'NUMERIC':
            c3d += '\tambitoFuncion =  ts.buscarIDF()\n'
            c3d += '\tNuevoSimbolo = TAS.Simbolo(cont,\''+str(self.id)+'\',TAS.TIPO.NUMERIC,ambitoFuncion,None, None, None, None, None, None, None ,None,None,'+valor+', '+col+
            c3d += '\ttabla.agregar(NuevoSimbolo)\n'
            c3d += '\tcont+=1\n'
        elif self.tipo == 'REAL':
            c3d += '\tambitoFuncion =  ts.buscarIDF()\n'
            c3d += '\tNuevoSimbolo = TAS.Simbolo(cont,\''+str(self.id)+'\',TAS.TIPO.REAL,ambitoFuncion,None, None, None, None, None, None, None ,None,None,'+valor+', '+col+','
            c3d += '\tts.agregar(NuevoSimbolo)\n'
```

Optimización de parámetros:

```
318  class llamadaP(pl):
319      def __init__(self,id,lparams) -> None:
320          self.id = id
321          self.lparams = lparams
322
323      def traducir(self):
324          if  not ts.existeF(str(self.id)):
325              print('Funcion '+str(self.id) +' no existe')
326              e = errores.CError(0,0,"Error en llamada de proceso, no existe",'Semantico')
327              errores.insert_error(e)
328              return '\tprint( \'Funcion '+ str(self.id) + ' no existe\')\n'
329          c3d = ''
330          contadorP = 0
331          for expresion in self.lparams:
332
333              trad = expresion.traducir()
334              c3d += trad[0] +'\n'
335              c3d += 'pila['+contadorP+'] = '+trad[1]
336              contadorP +=1
```

método para optimizar el código para eliminar una función.

```
385  class dropfunc(pl):
386      def __init__(self,ids) -> None:
387          self.ids = ids
388
389      def traducir(self):
390
391          c3d = ''
392          self.ejecutar()
393          for identificador in self.ids:
394
395              c3d += '\tts.deleteFP(str(\''+identificador+'\'))\n'
396              if  not ts.existeF(str(identificador)):
397                  e = errores.CError(0,0,"Error drop funcion, "+str(identificador)+" no existe como funcion",'Semantico')
398                  errores.insert_error(e)
399          return c3d
400
401
```

SALIDA Y REPORTES

Se utilizaron dos librerías más para la parte de visualización de datos de datos, en que caso de graphviz es sencillo construir tablas y mostrarla en pdf, en el caso del reporte se usaron conceptos más avanzados, debido a que la gramática cambio conforme al desarrollo, se implementó una clase capaz de graficar cualquier otra clase abstracta, para ello en siguiente link explican como es que los objetos json se pueden transformar en un árbol. Python create tree from a JSON file (visbud.blogspot.com) la solución 1 es sencilla, utiliza recursión y joins, a partir de esta solución se saca un archivo de texto que es mucho más comprensible (no es un árbol en forma ast pero es un árbol que funciona con tabulaciones) por suerte para graficar ese archivo de texto nos encontramos con la siguiente solución algorithm - Python file parsing: Build tree from text file - Stack Overflow.

En el caso del pretty table es solo una librería que permite transformar arreglos a texto tabulado en una tabla, esto casa bien imprimir tablas la consola.

LICENCIAS:

Las 3 librerías utilizadas son compatibles con la licencia MIT por lo que no supone un problema el uso de estas librerías.