Universidad de San Carlos de Guatemala
Facultad de ingeniería
Escuela de Ciencias y Sistemas
Lenguajes y Compiladores 2

MANUAL TÉCNICO

Tytus DB – Fase2

Augusto German Mazariegos Salguero - 201114496
Glendy Marilucy Contreras González - 201025406
Brayan Ezequiel Santiago brito - 201114566
Luis Carlos valiente Salazar – 201122864

Grupo 4

Guatemala enero de 2021

CONTENIDO

DESCRIPCIÓN DEL LENGUAJE TYTUS

Tytus es un proyecto open source de un administrador de bases de datos. Está compuesto por tres componentes interrelacionados:

1. El administrador de almacenamiento de la base de datos.
2. El administrador de la base de datos.
3. SQL parser. El cual es el que se especifica en este documento.

SQL PARSER

Este componente proporciona al servidor una función encargada de interpretar sentencias del subconjunto del lenguaje SQL.

Está compuesto por tres subcomponentes:

SQL Parser: Es el intérprete de sentencias de SQL, que proporciona una función para invocar al parser, al recibir una consulta el parser luego del proceso interno y de la planificación de la consulta invoca las diferentes funciones proporcionadas por el componente de administrador de almacenamiento.

Type Checker: Es un subcomponente que ayudará al parser a la comprobación de tipos. Al crear un objeto cualquiera se debe crear una estructura que almacenará los tipos de datos y cualquier información necesaria para este fin.

Query Tool: Es un subcomponente que consiste en una ventana gráfica similar al Query Tool de pgadmin de PostgreSQL, para ingresar consultas y mostrar los resultados, incluyendo el resalto de la sintaxis. La ejecución se realiza de todo el contenido del área de texto.

HERRAMIENTAS UTILIZADAS PARA EL DESARROLLO

VISUAL STUDIO CODE:

Como editor de código fuente, ya que incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código.

Es de código abierto.

PYTHON (versión de cada integrante del equipo, 3.6-3.9)

Lenguaje de programación multiparadigma utilizado para la creación del parser.

GRAPHVIZ (versión 2.38)

Es un conjunto de herramientas de software para el diseño de diagramas definido en el lenguaje descriptivo DOT.

Utilizado para generar el árbol sintáctico de la gramática.

PAQUETE PLY (versión 3.11)

PLY es una implementación de Python puro de las populares herramientas de construcción de compiladores lex y yacc. El objetivo principal de PLY es mantenerse bastante fiel a la forma en que funcionan las herramientas tradicionales lex / yacc para la realización de analizadores léxico y sintáctico respectivamente.

PLATAFORMA DE USUARIO

REQUISITOS MÍNIMOS

SISTEMA OPERATIVO

- WINDOWS 8 EN ADELANTE
- UBUNTU 14 EN ADELANTE
- MAC OS 10.11 EN ADELANTE

RAM

2GB

ESPACIO DE ALMACENAMIENTO

40GB

ESPECIFICACIÓN DE CLASES Y MÉTODOS IMPORTANTES AGREGADOS EN LA 2DA FASE.

CLASES PARA FUNCIONES Y PROCEDIMIENTOS

CLASE	EXPRESIÓN	
DESCRIPCIÓN	En la carpeta "plsql" se encuentran las clases que administran todas las funciones, procedimientos que vienen en el archivo de entrada. Dentro de ellas pueden venir sentencias if, declaraciones, asignaciones y retornos. Hay una clase para manejar cada una de estas operaciones.	

Proc	Proc (Instruccion)	Clase que deriva de Instrucción. Se encarga de agregar las funciones y procedimientos del archivo de entrada a la tabla de símbolos y genera el código de 3 direcciones de la misma.
		Sus atributos son los siguientes:
		`self.name = name`
		`self.params = params`
		`self.block = block`
		`self.ret = ret`
		`self.linea = linea`
		`self.columna = columna`

	getCodigo()	Genera el código en 3 direcciones de las funciones y procedimientos y las agrega a la tabla de símbolos.

Statement	Statement(Instruccion)	Clase encargada de las declaraciones de variables que pueden venir dentro de una función o un procedimiento.
		Atributos:
		`self.dec = dec`
		`self.expresion = expresion`
		`self.linea = linea`
		`self.columna = columna`

	getCodigo()	Genera el código en 3 direcciones de las variables que vienen dentro de funciones y procedimientos y las agrega a la tabla de símbolos si estos no han sido creados anteriormente.

Assignment	Assignment(Instruccion)	Clase que realiza las asignaciones de las variables declaradas con anterioridad a la tabla de símbolos.
		Atributos:
		`self.id = id`
		`self.expresion = expresion`
		`self.linea = linea`
		`self.columna = columna`

	getCodigo()	Genera el código en 3 direcciones de las asignaciones que vienen dentro de funciones y procedimientos y las agrega a la tabla de símbolos si las variables correspondientes a ellas han sido creados anteriormente.

Drop	Drop_PF(Instruccion)	Clase que se encarga de la eliminación de procedimientos o funciones:
		Atributos:
		`self.id = id`
		`self.tipo = tipo`

	getCodigo()	Genera el código en 3 direcciones para la eliminación de procedimientos o funciones.
Block	Block (Instrucción)	Clase que deriva de Instrucción. Se encarga de agregar las las instrucciones que vienen dentro de una función o procedimiento a la tabla de símbolos y genera el código de 3 direcciones de las mismas. Sus atributos son los siguientes: `self.instructions = instructions`
	getCodigo()	Genera el código en 3 direcciones de las instrucciones que vienen dentro de funciones y procedimientos y las agrega a la tabla de símbolos.
Call	call(Instruccion)	Llama a cualquier función o procedimiento que ha sido creado previamente.
	getCodigo()	Genera el código en 3 direcciones de las llamadas a funciones y procedimientos y las agrega a la tabla de símbolos.
IF	clase_if(Instruccion)	Clase destinada para el manejo de la sentencias if e ifelse que pueden venir dentro de un procedimiento o función.

		Sus atributos son los siguientes:
		`self.expresion = expresion`
		`self.block = block`
		`self._else_block = _else_block`
		`self.linea = linea`
		`self.columna = columna`

	getCodigo()	Genera el código en 3 direcciones de las instrucciones que vienen dentro de un if o un ifelse y las agrega a la tabla de símbolos.

Execute	Execute(Instruccion)	Clase destinada para la ejecución de una función o procedimiento.
		Sus atributos son los siguientes:
		`self.id = _id`
		`self.expressions = expressions`
		`self.linea = linea`
		`self.columna = columna`

	getCodigo()	Genera el código en 3 direcciones de la instrucción Execute.

Retornar	clase_return(Instruccion)	Devuelve el valor de una variable que se encuentre en la tabla de símbolos.
		Sus atributos son los siguientes:
		`self.expresion = expresion`
		`self.linea = linea`
		`self.columna = columna`

	getCodigo()	Genera el código en 3 direcciones de la instrucción Return.

CODIGO EN TRES DIRECCIONES

Se agregó el método getCodigo() a todas las instrucciones creadas en la fase1, este método se encarga de la generación del código de tres direcciones para cada una de las instrucciones.

CLASES UTILIZADAS PARA LA OPTIMIZACIÓN DE CODIGO EN 3D

CLASE	EXPRESIÓN	
DESCRIPCIÓN	En la carpeta "Optimizacion" se encuentran las clases que administran todas las funciones, procedimientos que van a ser optimizadas una vez ha sido generado el código en tres direcciones.	

_From	_From(Instruccion)	Clase que deriva de Instrucción. Se encarga de optimizar las funciones y procedimientos del código en 3 direcciones.
		Sus atributos son los siguientes:
		`self.id_list = id_list`
		`self.id = id`
		`self.linea = linea`

_Import	_Import(Instruccion)	
Goto	Goto(Instruccion)	
	toString()	Genera la optimización del código en 3 direcciones de las funciones y procedimientos.

_def	Def(Instruccion)	Clase encargada de las declaraciones de variables que pueden venir dentro de una función o un procedimiento.
		Atributos:
		`self.with_goto = with_goto`
		`self.id = id`
		`self.instrucciones=instrucciones`
		`self.linea = linea`

	toString(self)	Genera la optimización del código en 3 direcciones de las variables que vienen dentro de funciones y procedimientos.
Asignación	Asignacion(Instrucción)	Clase que optimiza las asignaciones de las variables. Atributos:

		`self.izquierda = izquierda`
		`self.derecha = derecha`
		`self.linea = linea`

	toString()	Genera la optimización del código en 3 direcciones de las asginaciones.

_global	Global(Instruccion)	Clase que se encarga de de optimizar la variable "Global" Que corresponde al ambiente de las variables en las funciones y procedimientos.
		Atributos:
		`self.id = id`
		`self.linea = linea`

	toString()	Genera la optimización del código en 3 direcciones de las variables globales.
Instruccion	Instruccion(ABC)	clase abstracta utilizado para optimización de instrucciones que han sido traducidas a código de 3 direcciones.
Expresion	Expresion(ABC)	clase abstracta utilizado para optimización de expresiones que han sido traducidas a código de 3 direcciones.

llamada	call(Instruccion)	Llama a cualquier función o procedimiento que ha sido creado previamente.
		Atritutos:
		`self.expresion = expresion`
		`self.lista_expresiones=listaexpresiones`
		`self.linea = linea`

	toString()	Genera la optimización del código en 3 direcciones de llamadas a funciones y procedimientos.

_if	_If(Instruccion)	Clase que se encarga de optimizar sentencias if e ifelse del código en 3 direcciones.
		Sus atributos son los siguientes:
		`self.expresion = expresion`
		`self.id = id`

		`self.linea = linea`
	toString(self)	Genera la optimización del código en 3 direcciones de las sentencias if e ifelse.

Execute	Execute(Instruccion)	Clase destinada para la ejecución de una función o procedimiento.
		Sus atributos son los siguientes:
		`self.id = _id`
		`self.expressions = expressions`
		`self.linea = linea`
		`self.columna = columna`

	getCodigo()	Genera el código en 3 direcciones de la instrucción Return.
Optimizaciones	Métodos que reciben los parámetros: Instrucciones y arbol. optimizacion_uno optimizacion_ocho optimizacion_nueve optimizacion_diez optimizacion_once optimizacion_doce optimizacion_trece optimizacion_catroce optimizacion_quice optimizacion_dieciseis optimizacion_diecisiete optimizacion_dieciocho	Métodos que realizan cada una de las reglas de optimización y que son aplicadas a todas las generaciones de código en 3 direcciones.

TABLA DE SIMBOLOS PARA INSTRUCIONES, INDICES Y CODIGO EN 3D

Clase Arbol

Su objetivo es generar administración para los símbolos.

Su atributo principal es el diccionario que administra los símbolos.

ATRIBUTOS:

```
        self.instrucciones = instrucciones
        self.excepciones = []
        self.consola = []
        self.bdUsar = None
        self.listaBd = []
        self.where = False
        self.update = False
        self.relaciones = False
        self.nombreTabla = None
        self.tablaActual = []
        self.columnasActual = []
        self.lEnum = []
        self.lRepDin = []
        self.comprobacionCreate = False
        self.columnaCheck = None
        self.order = None
        self.temporal = -1
        self.index = []
        self.ts = {}
        self.scope = None
```

Métodos importantes de la clase árbol

create(parent)

Métodos importantes:

```
tree = Treeview(parent)
        tree['columns'] = ('tipo', 'nombreTabla')
        tree.heading('#0', text='Nombre')
        tree.column('#0', anchor='center', width=25)
        tree.heading('tipo', text='TIPO')
        tree.column('tipo', anchor='center', width=200)
        tree.heading('nombreTabla', text='Nombre Tabla')
        tree.column('nombreTabla', anchor='center', width=25)
```

	getBasedeDatos()	Retorna la base de datos.
	agregarTablaBD()	Devuelve la tabla de la base de datos
	llenarTablas()	Agrega las tablas a la tabla de símbolos
	DevolverColumnasTabla()	Devuelve las columnas en la tabla.
	elimnarDB() elimninarTabla()	Elimina todos los símbolos de la tabla correspondientes a la base de datos o a la tabla respectivamente.
Clase Símbolo	class Simbolo()	Esta clase se utiliza para crear un símbolo de base para una declaración de variable.
Clase Tabla	class Tabla()	Esta clase representa la tabla de símbolos.

	Atributos	`self.anterior = anterior`
		`self.variables = []`
		`self.funciones = []`

	setVariable() getVariable()	'Set' Introduce un nuevo dato de una variable a la tabla. 'Get' Obtiene un dato de variable que ya existente en la tabla de símbolos.

	setFunction() getFunction()	‘Set’ Introduce un nuevo dato de una variable a la tabla. ‘Get’ Obtiene un dato de variable que ya existente en la tabla de símbolos.
Clase TipoDato	Class TipoDato()	Esta clase es de utilidad para la comprobación de tipos

INTERFAZ GRAFICA (MAIN1 Y MAIN2)		
DESCRIPCIÓN	Su objetivo es manejar la interfaz gráfica del programa. Utiliza la librería ***tkinter*** principalmente para cumplir su cometido. Existen dos archivos que contienen la interfaz Gráfica, el archivo 'Main1' y el archivo 'Main2' ambos con los mismos métodos y funciones.	
	Interfaz()	Maneja la ventana principal del IDE. Contiene los menús, barras de herramientas, área de texto para el editor y consola y los botones.
	btnanalizar_click(self)	Ejecuta la entrada de texto correspondiente a las instrucciones psql.
	btnanalizar3D_click(self):	Ejecuta la entrada de texto correspondiente a las instrucciones en Código en 3 direcciones.
	btngetCodigo_click(self)	Genera el código en 3 direcciones de la entrada de texto.
	getFuncionIntermedia(self, arbol)	Genera el código en 3 direcciones de las cadenas que se envían de forma literal.
	abrir_click(self) guardar_click(self) guardar_como_click(self) tblerrores_click(self) tblsimbolos_click(self) tblsimbolos3D_click(self) ast_click(self) tbindex_click(self)	Métodos para las pestañas de la interfaz gráfica, estas sirven para: abrir, guardar, guardar como de la entrada de texto. Abrir la tabla de errores, la tabla de símbolos, la tabla de símbolos en 3 direcciones, la tabla de símbolos de los índices y mostrar el árbol ast.

CLASE EXCEPCIÓN		
DESCRIPCIÓN	Esta clase se utiliza para guardar errores de tipo léxico, sintáctico y semántico.	

	Atributos de la clase:	`self.cod_error = cod_error`
		`self.tipo = tipo`
		`self.descripcion = descripcion`
		`self.linea = linea`
		`self.columna = columna`

GRAMÁTICA

```
                    FUNCIONES Y PROCEDIMIENTOS

Definición de la gramática

      init: instrucciones

      instrucciones:  instrucciones instrucción

      instrucciones: instruccion


Funciones con y sin parametros

      instruccion: CREATE FUNCTION ID PARIZQ PARDER retorna DOLLAR body
      DOLLAR lenguaje PUNTO_COMA

      instruccion: CREATE FUNCTION ID PARIZQ parametros PARDER retorna
      DOLLAR body DOLLAR lenguaje PUNTO_COMA

Procedimientos con y sin parámetros

      instruccion: CREATE PROCEDURE ID PARIZQ PARDER lenguaje AS DOLLAR
      body DOLLAR

      instruccion: CREATE PROCEDURE ID PARIZQ parametros PARDER lenguaje
      AS DOLLAR body DOLLAR

Definición del cuerpo

      body : BEGIN instrucciones END PUNTO_COMA

      body : lfun_declare BEGIN instrucciones END PUNTO_COMA

Tipo para retornar

      retorna: RETURNS tipo AS

      parametros: parametros COMA parametro

      parametros: parametro
```

```
       parametro: ID tipo

Lenguaje
      lenguaje:  LANGUAGE PLPGSQL


Declaraciones

      instruccion : fun_declare PUNTO_COMA

      lfun_declare : lfun_declare fun_declare

      lfun_declare : fun_declare

      fun_declare : DECLARE lcont_declare

      lcont_declare : lcont_declare cont_declare

      lcont_declare : cont_declare

      cont_declare :  ID tipo PUNTO_COMA

Asignaciones

      instruccion : ID 2PUNTOS IGUAL expre PUNTO_COMA

Sentencias IF e IFELSE

      instruccion : IF expre THEN instrucciones END IF PUNTO_COMA

      instruccion : IF expre THEN instrucciones ELSE instrucciones END
      IF PUNTO_COMA

Llamadas a funciones y procedimientos (EXECUTE)

      instruccion : EXECUTE ID PARIZQ  PARDER PUNTO_COMA

      instruccion : EXECUTE ID PARIZQ l_expresiones PARDER PUNTO_COMA

Return

      instruccion : RETURN expre PUNTO_COMA

      instruccion : expre PUNTO_COMA
```

```
Eliminación de Funciones y Procedimientos

      instruccion: DROP FUNCTION ID PARIZQ PARDER PUNTO_COMA

      instruccion: DROP FUNCTION ID PARIZQ tipo PARDER PUNTO_COMA

      instruccion: DROP PROCEDURE ID PARIZQ PARDER PUNTO_COMA



                          MANEJO DE INDICES

Creación de índices

      instruccion: CREATE INDEX ON ID PARIZQ op_index PARDER PUNTO_COMA

      instruccion: CREATE INDEX ID ON ID PARIZQ op_index PARDER PUNTO_COMA

      instruccion : CREATE opcion INDEX ID ON ID  PARIZQ op_index PARDER
      PUNTO_COMA

      instruccion : CREATE INDEX ID ON ID USING HASH PARIZQ op_index
      PARDER  PUNTO_COMA

      nstruccion : CREATE opcion INDEX ID ON ID USING HASH PARIZQ op_index
      PARDER  PUNTO_COMA

      instruccion : CREATE INDEX ID ON ID PARIZQ op_index PARDER
      instructionWhere PUNTO_COMA

Alter de índices

      instruccion : ALTER INDEX IF EXISTS ID RENAME TO ID PUNTO_COMA
                  | ALTER INDEX           ID RENAME TO ID PUNTO_COMA

      instruccion : ALTER INDEX IF EXISTS ID SET TABLESPACE ID PUNTO_COMA
                  | ALTER INDEX           ID SET TABLESPACE ID PUNTO_COMA

      instruccion : ALTER INDEX ID DEPENDS ON EXTENSION ID PUNTO_COMA

      instruccion : ALTER INDEX  IF EXISTS ID SET PARIZQ op_index PARDER
      PUNTO_COMA
                  | ALTER INDEX ID SET PARIZQ op_index PARDER PUNTO_COMA

      instruccion : ALTER INDEX  IF EXISTS ID RESET PARIZQ op_index
```

```
      PARDER PUNTO_COMA
                  | ALTER INDEX  ID RESET PARIZQ op_index PARDER PUNTO_COMA


      instruccion : ALTER INDEX ALL IN TABLESPACE ID OWNER BY ID PUNTO_COMA
                  | ALTER INDEX ALL IN TABLESPACE ID PUNTO_COMA

      instruccion : ALTER INDEX IF EXISTS ID ALTER ID ENTERO PUNTO_COMA
                  | ALTER INDEX           ID ALTER ID ENTERO PUNTO_COMA

Eliminación de índices

      instruccion : DROP INDEX IF EXISTS ID cr_index PUNTO_COMA

      instruccion : DROP INDEX ID cr_index PUNTO_COMA

      instruccion : DROP INDEX IF EXISTS  ID PUNTO_COMA
                  | DROP INDEX            ID PUNTO_COMA


      cr_index : CASCADE
               | RESTRICT

      op_index : ID NULLS FIRST

      op_index : ID ASC NULLS LAST

      op_index : ID DESC NULLS LAST

      op_index : LOWER PARIZQ ID PARDER

      op_index : PARIZQ LOWER PARIZQ ID PARDER PARDER

      op_index : ID IGUAL expresion

      op_index : namecs

      namecs : namecs COMA ID

      namecs : ID
```

GRAMATICA QUE RECONOCE EL CODIGO EN 3 DIRECCIONES

```
      init : instrucciones

      instrucciones    :  instrucciones instruccion

      instrucciones : instrucción

      instruccion : asignacion
                  | _if
                  | definicion_etiqueta
                  | goto_etiqueta
                  | llamada_funcion
                  | WITH_GOTO


Asignación de temporales y variables(globales, locales)

    asignacion :  ID IGUAL expresion
                |  TEMPORAL IGUAL expresion
                |  arreglo IGUAL expresion


Definición de la estructura de una nueva función

      instruccion : DEF ID PARIZQ PARDER DPUNTOS instrucciones

Definicion de una nueva etiqueta

      definicion_etiqueta : LABEL PUNTO ID

Definicion de salto a etiqueta especifica

      goto_etiqueta : GOTO PUNTO ID

Sentencia IF
      _if : IF expresion DPUNTOS GOTO PUNTO ID
         | IF expresion DPUNTOS
```

```
importaciones con from
      instruccion : FROM lubicacion IMPORT ID

      lubicacion : lubicacion PUNTO lubicacion


      lubicacion : ID
                 | GOTO


Importaciones simples

      instruccion : IMPORT ID


Definicion de variables globales
      instruccion : GLOBAL ID


Expresiones
    expresion : expresion MENOS    expresion
              | expresion MAS      expresion
              | expresion POR      expresion
              | expresion DIVIDIDO expresion
              | expresion MAYORQ expresion
              | expresion MENORQ expresion
              | expresion MAYOR_IGUALQ expresion
              | expresion MENOR_IGUALQ expresion
              | expresion IGUAL_IGUAL  expresion
              | PARIZQ expresion PARDER
              | llamada_funcion
              | ENTERO
              | CADENA
              | CARACTER
              | TEMPORAL
              | ID
              | NAME
              | arreglo
              | NONE
```

```
Definición de la estructura de un arreglo

      arreglo : ID CORIZQ list_parametros CORDER
            | CORIZQ list_parametros CORDER


Definición de una llamada a función

      llamada_funcion :  ID PARIZQ list_parametros PARDER
                      |  ID PUNTO ID PARIZQ list_parametros PARDER
                      |  arreglo PUNTO ID PARIZQ list_parametros PARDER
                      |  ID PARIZQ  PARDER


Definicion de lista de parámetros que puede tener una función

      list_parametros : list_parametros COMA expresion
                      | expresion
```