

Universidad de San Carlos de Guatemala  
Ingeniería en Ciencias y Sistemas  
Organización de Lenguajes y Compiladores 2  
Ing. Luis Espino  
Aux. Juan Carlos Maeda

# Manual Tecnico SQL Parser

Grupo 29

Quetzaltenango, diciembre 2020



## Índice

<b>Introducción</b>	<b>3</b>
<b>Objetivos</b>	<b>4</b>
Objetivos de SQL Parser	4
Objetivos específicos de SQL Parser	4
Dirigido a	4
<b>Herramientas utilizadas para el desarrollo</b>	<b>5</b>
Python	5
Visual Studio Code	5
Ply	5
GitHub	6
<b>Descripción</b>	<b>7</b>
<b>Gramática</b>	<b>8</b>
<b>Diagramas de Modelamiento</b>	<b>8</b>
SQL Parser	8
Diagrama de paquetes	8
Diagrama de clases	9
PL/PgSQL	11
Diagrama de paquetes	11
Diagrama de clases	12
Optimizador	15
Diseño de reportes y pantallas	16
Pantalla Principal	16
Pantalla de Reporte de Errores	17



## Introducción

El siguiente manual muestra una descripción clara y precisa de los procesos y herramientas que se utilizaron al momento de desarrollar el componente SQL parser de Tytus, un proyecto Open Source para desarrollar un administrador de bases de datos , así como también las funciones que tiene dicha aplicación, realizando un análisis, el cual recopila todos los requerimientos necesarios para su correcta funcionalidad.

Además se incluye una descripción y explicación del análisis lexico, sintactico y semantico el cual fue utilizado para la correcta manipulación de la información (inserción, actualización, eliminación,visualización), de esta forma se presenta un documento más completo para que facilite al programador su comprensión y futuro mantenimiento.

El objetivo del desarrollo de SQL parser, es interpretar sentencias SQL para manipular de manera correcta y eficaz la información.



## Objetivos

El objetivo del presente manual es entregar herramientas apropiadas para el futuro mantenimiento y mejoramiento de SQL parser

### Objetivos de SQL Parser

- Interpretar sentencias SQL
- Interpretar sentencias de PL/pgSQL

### Objetivos específicos de SQL Parser

- Invocar las funciones proporcionadas por el administrador de almacenamiento, para realizar operaciones sobre la base de datos.
- Manipular el resultado de las funciones para restringir y mostrar los resultados indicados por la/las consulta/s.
- Retornar información detallada de la consulta al servidor.
- Almacenar funciones y procedimientos para poder reutilizarlas.

### Dirigido a

Personas que darán un futuro mantenimiento a la aplicación, o personas que desean comprender el funcionamiento básico del mismo al momento de la ejecución.



## Herramientas utilizadas para el desarrollo

- **Python**

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional.

- **Visual Studio Code**

Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código.

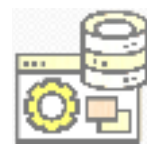
- **Ply**

PLY es una herramienta de análisis escrita exclusivamente en Python. Es, en esencia, una reimplementación de Lex y Yacc originalmente en lenguaje C. Fue escrito por David M. Beazley. PLY utiliza la misma técnica de análisis LALR que Lex y Yacc.



- **GitHub**

GitHub es una plataforma de desarrollo colaborativo de software para alojar proyectos utilizando el sistema de control de versiones Git. GitHub aloja un repositorio de código y brinda herramientas muy útiles para el trabajo en equipo, dentro de un proyecto. Además de eso, se puede contribuir a mejorar el software de los demás contribuidores. Para poder alcanzar esta meta, GitHub provee de funcionalidades para hacer un fork y solicitar pulls. Realizar un fork es simplemente clonar un repositorio ajeno (genera una copia en tu cuenta), para eliminar algún bug o modificar cosas de él. Una vez realizadas las modificaciones se puede enviar un pull al dueño del proyecto. Éste podrá analizar los cambios que se ha realizado fácilmente, y si considera interesante la contribución, adjuntarlo con el repositorio original.



## Descripción

SQL Parser al ser un componente de Tytus, proporciona la servidor una función la cual se encarga de interpretar sentencias del subconjunto del lenguaje SQL especificado en la documentación [siguiente](#)

SQL parser está compuesto por 3 subcomponentes:

- Query Tool: es una herramienta gráfica en la cual se ingresan las consultas y se muestran los resultados, además de proporcionar mensajes de error si las consultas lo poseen.

Nota: La consulta abarca todo el contenido ingresado en el área de texto. (Ver figura )

- Type Checker: Este subcomponente es un apoyo a SQL parser para la comprobación de tipos.
- SQL Parser: este subcomponente es el intérprete de las sentencias y consultas SQL y el que se conectara con el componente de administración de datos.
- PL/pgSQL Engine: este subcomponente es un intérprete del Procedural Language/PostgreSQL Structure Query Language, el cual permite ejecutar comandos SQL mediante un lenguaje de sentencias imperativas.

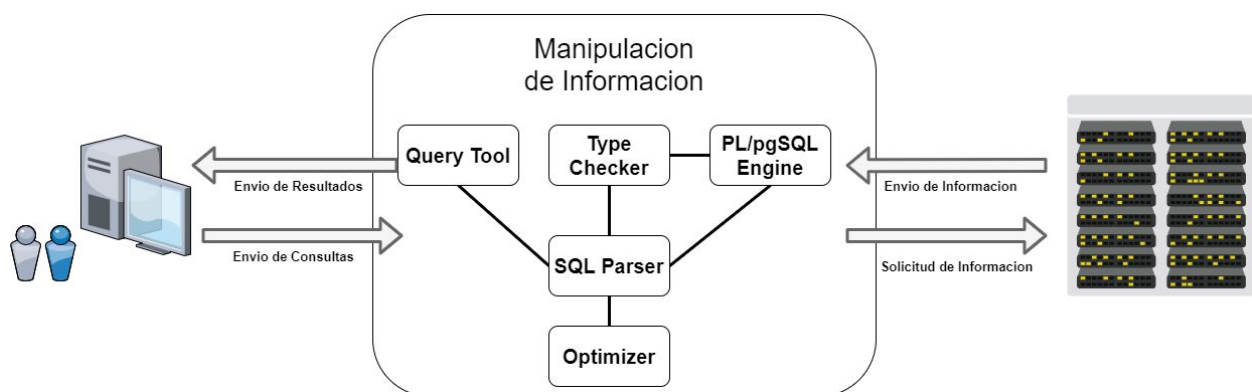


Figura 1. Descripción de la relación entre componentes



## Gramática

Para el desarrollo de SQL parser se crearon 2 gramáticas, una [ascendente](#) y una [descendente](#). Se realizó un análisis verificando las ventajas y desventajas de cada una, donde se llegó a la conclusión que la ascendente es la opción más viable debido a lo siguiente:

- Según la [documentación de PLY](#) esta librería realiza un parseo LR, por lo que solo es compatible con gramáticas ascendentes.
- La gramática ascendente tiene menos producciones que la descendente, esto tiene como consecuencia:
  - Un tiempo de implementación más corto.
  - Un tiempo de parseo menor.
  - Menos nodos del AST, lo cual se traduce en menos espacio de memoria ocupado por el programa y un menor tiempo de ejecución.
  - Menor probabilidad de errores en la implementación.
- La gramática ascendente presenta un menor nivel de complejidad, al no tener que acceder a la pila.
- La mayoría de los integrantes del grupo, poseen más experiencia implementando gramáticas ascendentes que descendentes.

## Diagramas de Modelamiento

### SQL Parser

#### Diagrama de paquetes

SQL Parser posee los paquetes de UI, este posee un conjunto de ventanas las cuales sirven para recibir las consultas y mostrar la información procesada, y el paquete Analizador el cual recibe las consultas enviadas por el paquete UI, las procesa y ejecuta funciones proporcionadas por la interrelación con el administrador de almacenamiento, este proporcionará los datos que se deberán manipular y ya manipulados serán mostrados por el paquete UI.



# SQL Parser

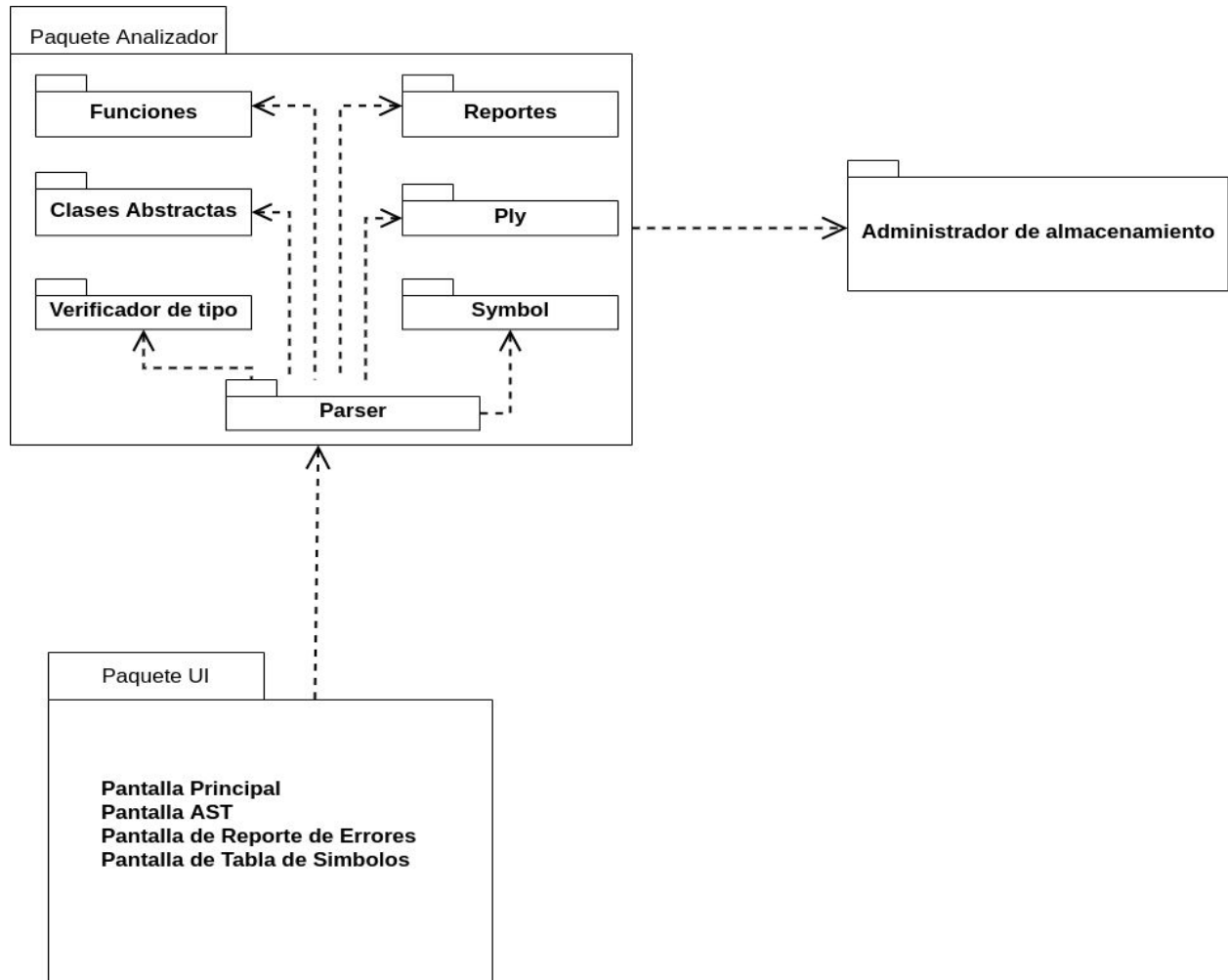


Figura 2. Diagrama de paquetes SQL parser

## Diagrama de clases

El diagrama de clases está compuesto de las entidades y atributos que se crearon para el procesamiento de consultas y la manipulación de la información.

# SQL Parser

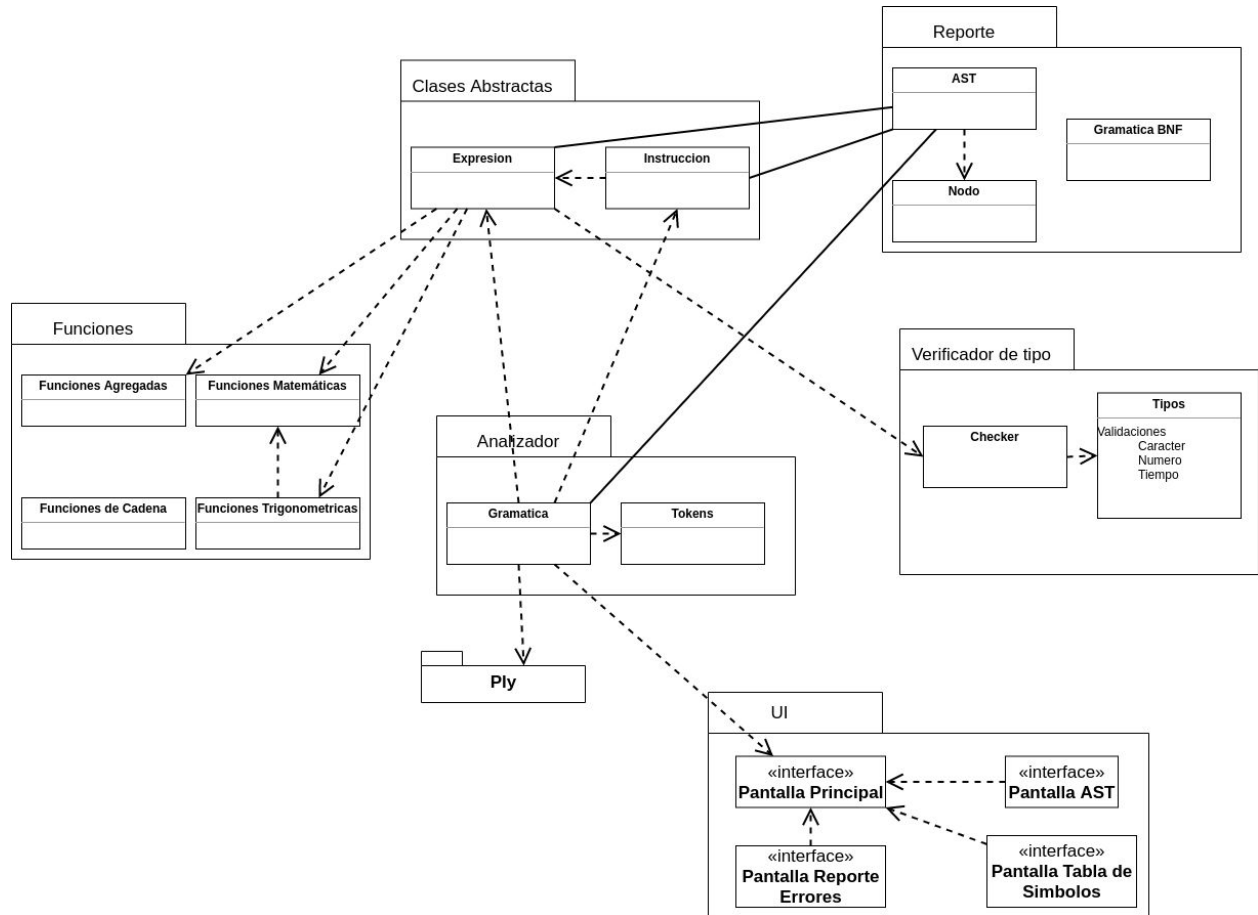


Figura 3. Diagrama de clases SQL parser

## Analizador

- Gramática
- Tokens

## Funciones

- Funciones Agregadas
- Funciones Matemáticas
- Funciones Trigonómicas
- Funciones De Cadena

## Reporte

- AST
- Nodo



# SQL Parser

- Gramática BNF

## **Clases Abstractas**

- Expresión
- Instrucción

## **Verificador de tipo**

- Checker
- Tipos
  - Validaciones
    - Carácter
    - Número
    - Fecha

## **UI**

- Pantalla AST
- Pantalla Reporte de Errores
- Pantalla Tabla de Símbolos
- Pantalla Principal

## **PL/PgSQL**

### **Diagrama de paquetes**

PL/PgSQL posee los paquetes de UI, este posee un conjunto de ventanas las cuales sirven para recibir las consultas y mostrar la información procesada, y el paquete Analizador el cual recibe las consultas enviadas por el paquete UI, las procesa y ejecuta funciones proporcionadas por la interrelación con el administrador de almacenamiento, este proporcionará los datos que se deberán manipular y ya manipulados serán mostrados por el paquete UI, además de un paquete Optimizer\_Folder el cual posee un conjunto de clases las cuales sirven para la optimización del código.

# SQL Parser

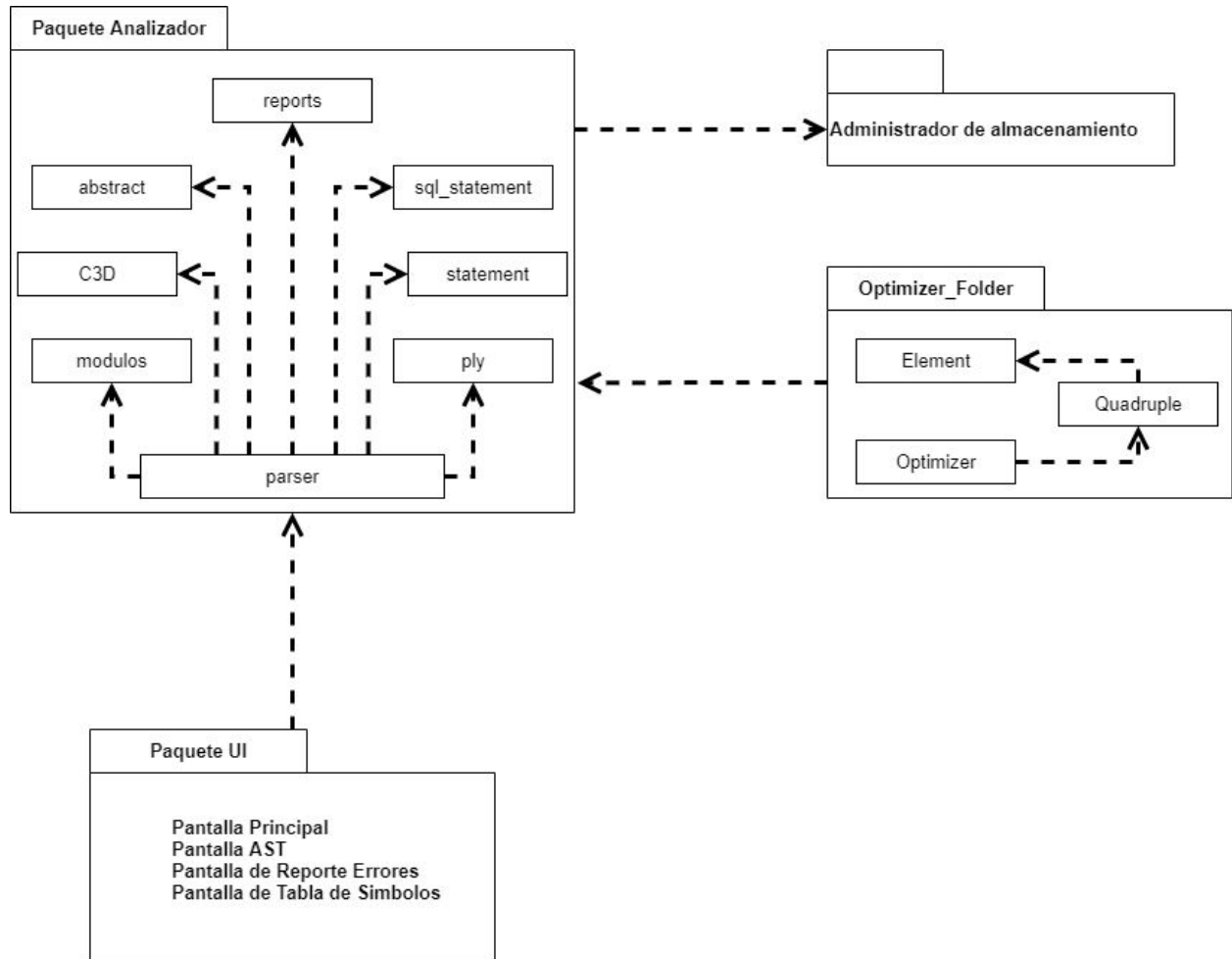


Figura 4. Diagrama de paquetes PL/PgSQL

## Diagrama de clases

El diagrama de clases está compuesto de las entidades y atributos que se crearon para el procesamiento de consultas y la manipulación de la información, así como también la generación y optimización del código 3 direcciones.

# SQL Parser

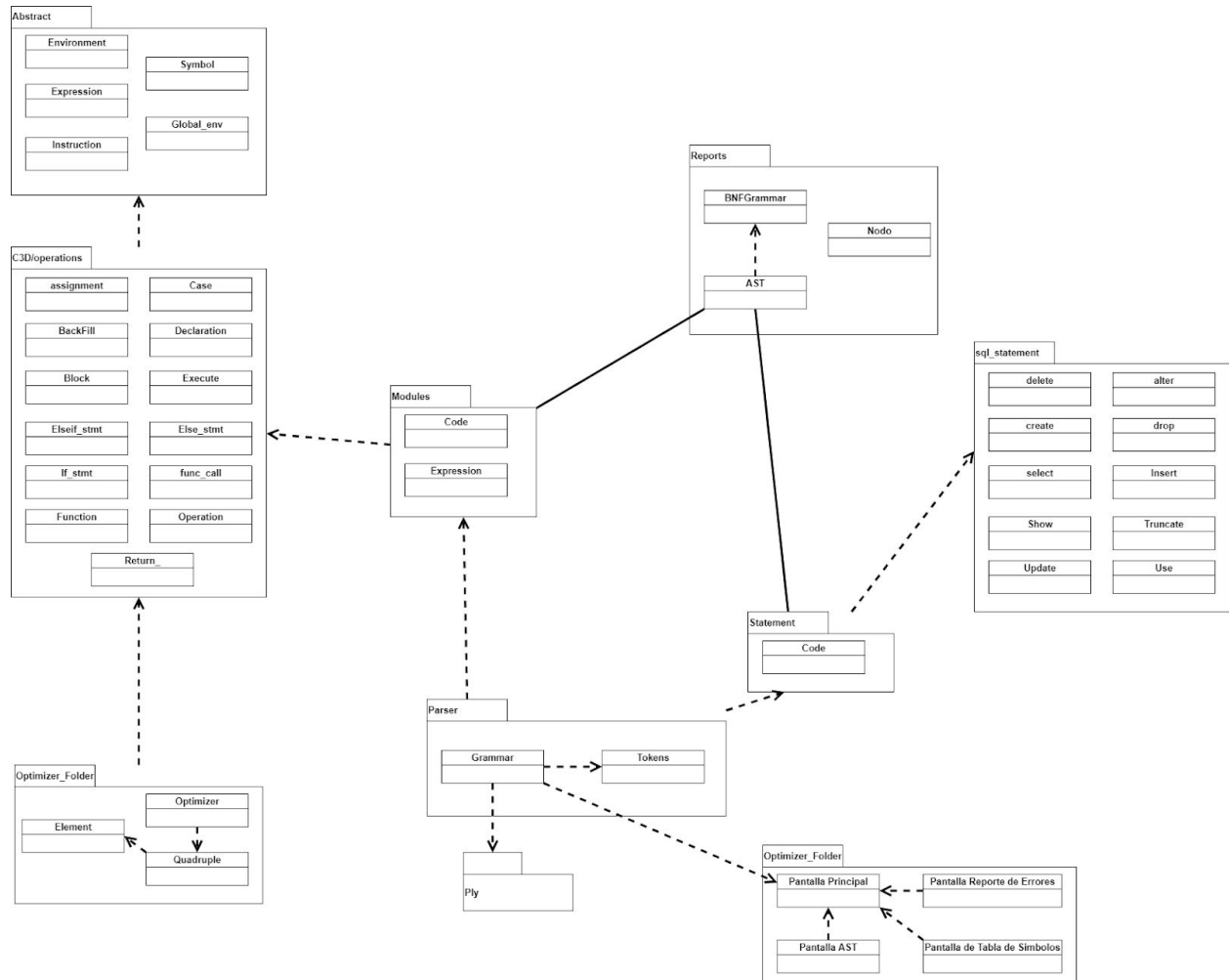


Figura 5. Diagrama de clases de PL/PgSQL

## Analizador

- Gramática
- Tokens

## Reporte

- AST
- Nodo
- Gramática BNF

## Clases Abstractas

- Expresión



# SQL Parser

- Instrucción
- Ambiente
- Ambiente Global
- Symbol

## **Código 3 Direcciones**

- Asignación
- BackFill
- Bloque
- Declaración
- Ejecutar
- Llamada a función
- Funcion
- Operación
- Retorno

## **Sentencias de Control-Código 3 direcciones**

- Sentencia If
- Sentencia Else
- Sentencia Else if
- Sentencia Case

## **Módulos**

- Módulo Code
- Módulo Expression

## **SQL Statement**

- Alter
  - Database
  - Table
- Create
  - Database
  - Table
  - Index
  - Type
- Drop
  - Database



# SQL Parser

- Index
- Table
- Select
- Delete
- Insert
- Show
- Truncate
- Update
- Use

## **Statement**

- Code

## **Optimizer\_Folder**

- Element
- Quadruple
- Optimizer

## **UI**

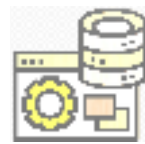
- Pantalla AST
- Pantalla Reporte de Errores
- Pantalla Tabla de Símbolos
- Pantalla Principal

## **Optimizador**

Para la optimización del código 3 direcciones se utilizan las reglas de optimización por mirilla, en donde se toman las instrucciones dentro de la mirilla y se sustituyen en una secuencia equivalente que sea de menor longitud y lo más rápido posible que el bloque original. El proceso de mirilla permite que por cada optimización realizada con este método se puedan obtener mejores beneficios.

Los tipos de transformación para realizar la optimización por mirilla serán los siguientes:

- Eliminación de instrucciones redundantes de carga y almacenamiento.
- Eliminación de código inalcanzable.



# SQL Parser

- Optimizaciones de Flujo de control.
- Simplificación algebraica y reducción por fuerza

La optimización se realiza gracias a la ayuda de Cuadruplas, el algoritmo realiza varias pasadas y en cada una aplica un tipo de transformación.

## Diseño de reportes y pantallas

### Pantalla Principal

La pantalla principal cuenta con 3 secciones las cuales son:

- Barra de menús
- Área de Consultas
- Área de Visualización de Resultados y Errores

### Barra de menús

Esta sección contiene 3 menús:

- Tabla de Símbolos: Se redirigirá a la pantalla de Tabla de símbolos
- AST: Se mostrará el AST generado
- Reporte de errores: Se mostrará la pantalla con los errores léxicos, sintácticos y semánticos

### Área de Consultas y Visualización de Resultados y Errores

Contiene un panel de entrada en la cual se ingresa la consulta, y con el botón consultar se enviará la consulta al Analizador para que este posteriormente envíe los resultados y estos se visualizarán en el área de visualización de resultados y errores.



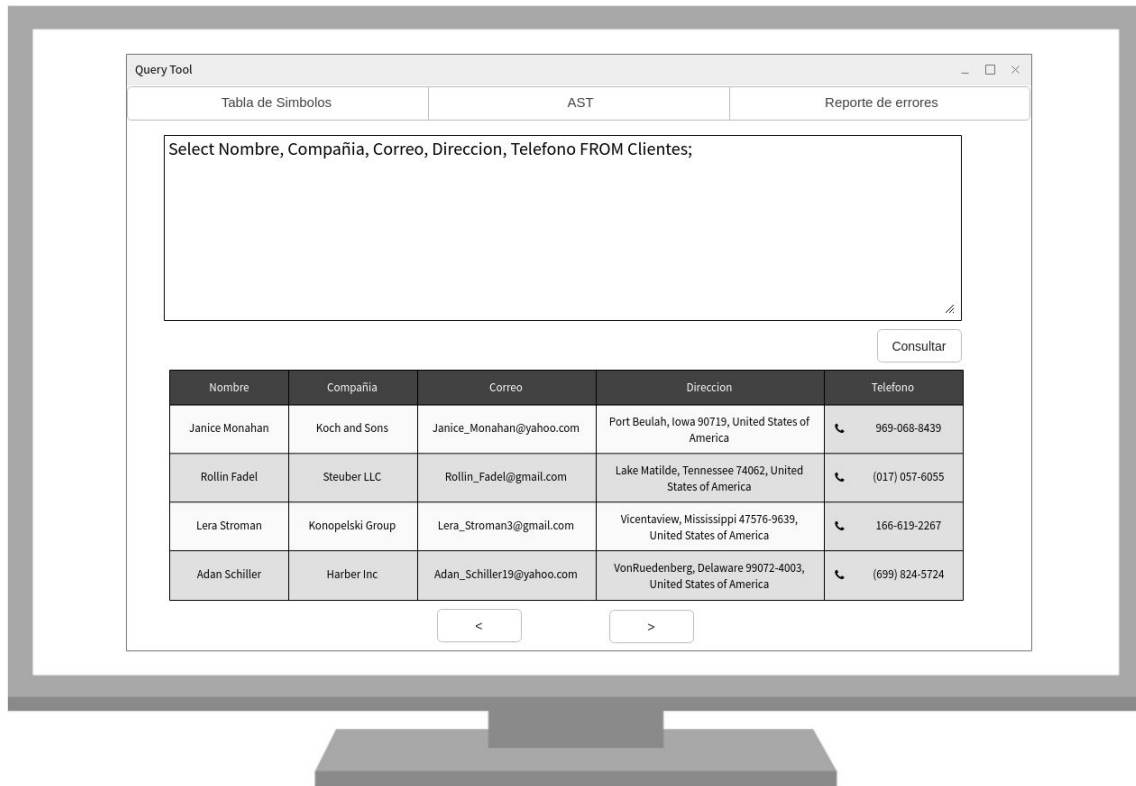


Figura 6. Pantalla principal

## Pantalla de Reporte de Errores

Los errores léxicos, semánticos y sintácticos se mostrarán en 3 tablas diferentes con una breve descripción y la línea en donde fueron encontrados

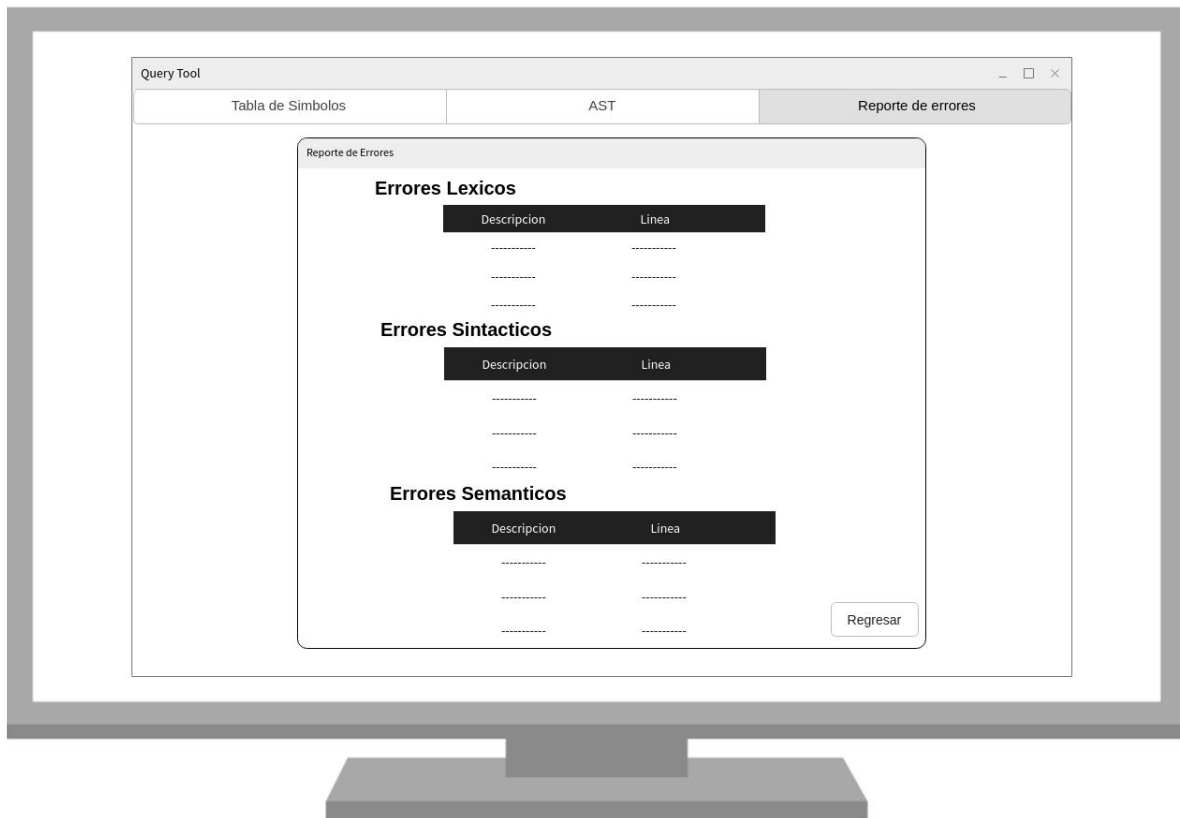
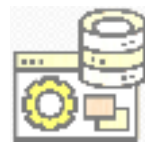


Figura 7. Pantalla de Reporte de Errores



## Pantalla de Tabla de Símbolos

En esta pantalla se muestran los datos de identificador, tipo, dimensión, ámbito de declaración y la referencia de las variables, funciones y procedimientos.



Figura 8. Pantalla de Tabla de Símbolos



## Pantalla AST

En esta pantalla se muestra una imagen del árbol de sintaxis abstracta, realizado mediante la herramienta de Graphviz

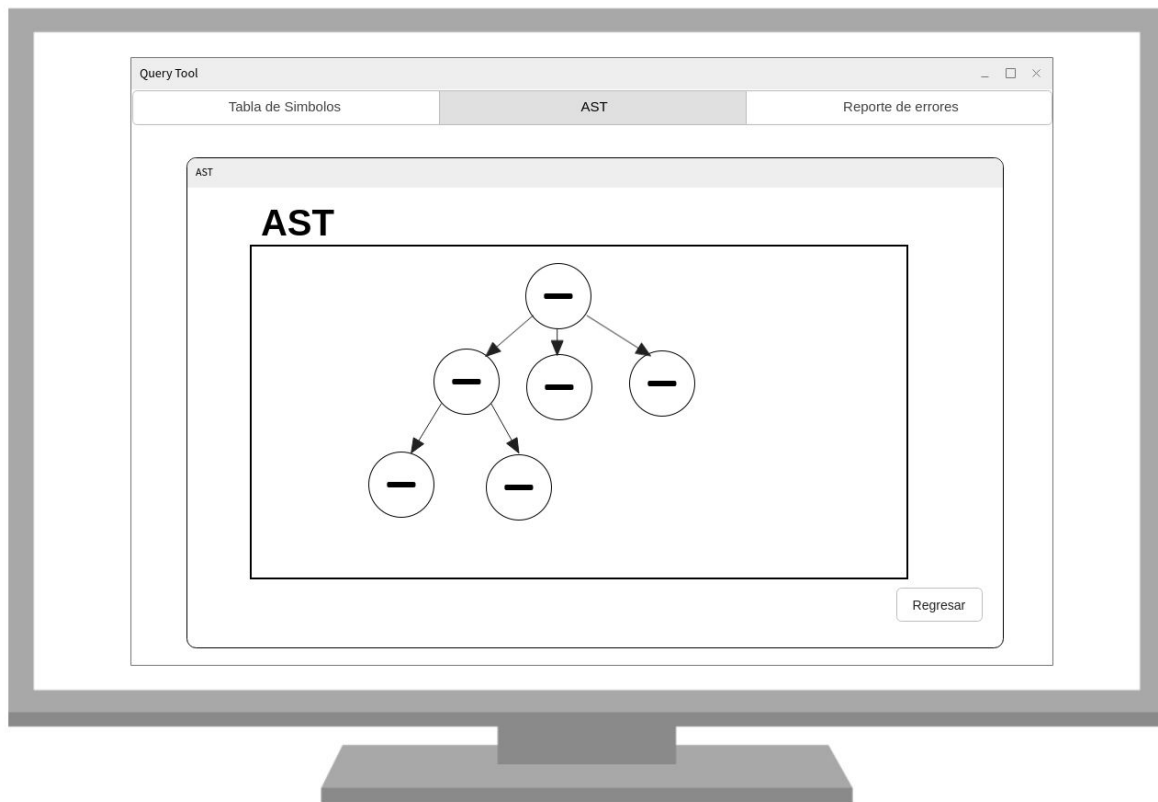


Figura 9. Pantalla AST