COMPILADORES PL2

Grupo 23

David Bachiller Vela

Víctor Sanavia Valdeolivas

Martín Mora

Contenido

[**INTRODUCCIÓN** 3](#_Toc150244942)

[**1.Primera parte: generación de árboles sintácticos para lenguajes específicos** 4](#_Toc150244943)

[**1.1** **SQLMini** 4](#_Toc150244944)

[**1.2** **Linguine** 7](#_Toc150244945)

[**2. Segunda parte: iniciación a la tabla de símbolos (1 punto)** 8](#_Toc150244946)

**INTRODUCCIÓN**

En el ámbito de la informática, la capacidad de construir analizadores léxicos y sintácticos para lenguajes de programación es fundamental. En este proyecto, nos adentramos en el mundo de la generación de árboles sintácticos utilizando la herramienta ANTLR. A través de la definición de gramáticas, el desarrollo de lexers y parsers, y la construcción de Árboles de Sintaxis Abstracta (AST), exploramos la esencia de la interpretación de lenguajes específicos.

Esta práctica se divide en dos partes: en la primera, nos enfocamos en la creación de un analizador capaz de identificar automáticamente los elementos de un lenguaje dado, mientras que, en la segunda, nos sumergimos en la visualización y manipulación de los ASTs resultantes. Además, siempre se podrán proponer mejoras que enriquezcan la gramática y la funcionalidad del analizador.

Para entender todo este proceso ahí que entender que lo que hace el analizador léxico es analizar una cadena de entrada, a partir de una Expresión Regular o definiciones regulares. Generando un token con un cierto valor, mientras que el analizador sintáctico le llegan todos estos tokens en un determinado orden y su función es verificar si el orden en el que llegan los tokens es válido, con sus determinadas reglas gramaticales, con lo que acabamos generando un árbol en el que se puede ver toda esta secuencia.

**1.Primera parte: generación de árboles sintácticos para lenguajes específicos**

## **SQLMini**

En esta parte de la práctica se pedía construir una gramática capaz de detectar distintas consultas SQL, las cuales se encontraban en el enunciado.

Primero que todo, debemos entender cómo funciona una gramática, la cual consta de dos partes:

* **Lexer** 🡪 Es el encargado de almacenar los símbolos terminales y de pasar los tokens a la siguiente parte
* **Parser** 🡪 Se encarga de leer la entrada y asignarlos usando los tokens que le llegan del *lexer*.

Nosotros hemos decidido dividir el *lexer* y el *parser* en dos archivos distintos.

En cuanto al *lexer* tenemos el siguiente código fuente:

**lexer** **grammar** gSqlMiniLexer;

//Aquí pondremos los simbolos terminales que tiene nuestro SqlMini

SELECT: 'SELECT';

FROM: 'FROM';

WHERE: 'WHERE';

ORDER: 'ORDER';

BY: 'BY';

ASC: 'ASC';

DESC: 'DESC';

MAYORQUE: '>';

MAYORIGUALQUE: '>=';

IGUALQUE: '=';

ABREPARENTESIS: '(';

CIERRAPARENTESIS: ')';

NUMERO: [0-9]+ ('.' [0-9]+)?; //Ahora tambien evaluamos los numeros decimales

STRING: '\'' ~'\''\* '\'';

COMA: ',';

AND: 'AND';

OR: 'OR';

ID: ([a-zA-Z]+) | '\*'; // Identificadores (en este caso, nombres de columnas o tablas)

WS: [ \t\r\n]+ -> skip; // Ignorar espacios en blanco

**Palabras Clave y Operadores:**

Las palabras clave como **SELECT**, **FROM**, **WHERE**, **ORDER**, **BY**, **ASC**, **DESC**, **AND**, y **OR**, así como los operadores de comparación como **MAYORQUE**, **MAYORIGUALQUE**, y **IGUALQUE**, son elementos fundamentales en cualquier consulta SQL.

**Símbolos de Puntuación:**

Los símbolos como, (**coma**), ( **(paréntesis de apertura)** y ) **(paréntesis de cierre)** son parte esencial de la sintaxis de SQLMini. Estos símbolos se utilizan para separar elementos en la consulta y para delimitar expresiones.

**Números:**

La regla NUMERO está diseñada para reconocer tanto números enteros como números decimales. Por ejemplo, esta regla puede identificar tanto 123 como 12.34.

**Cadenas de Texto:**

La regla STRING permite reconocer cadenas de texto encerradas entre comillas simples ('). Esto es crucial para manipular texto en las consultas SQL.

**Identificadores:**

La regla ID se utiliza para reconocer identificadores, que en SQLMini suelen ser nombres de columnas o tablas. Por ejemplo, si tenemos una columna llamada nombre, en la base de datos, esta regla permitirá identificarla.

**Espacios en Blanco:**

La regla WS se encarga de ignorar los espacios en blanco, tabulaciones, retornos de carro y saltos de línea. Esto es importante para que el analizador léxico pueda omitir caracteres que no afectan la estructura de la consulta.

Al definir estas reglas léxicas, establecemos la base para que el analizador léxico pueda reconocer los elementos individuales del código fuente SQLMini. Estos tokens serán utilizados posteriormente por el analizador sintáctico para construir la estructura de la consulta en forma de árbol sintáctico abstracto (AST).

Una vez introducido el *lexer* nos vamos al *parser*, el cual tiene el siguiente código fuente:

Vamos a utilizar los tokens que hemos definido en el lexer para poder declarar el parser con la sintaxis que se nos ha indicado que se define en programa.

**parser** **grammar** gSqlMiniParser;

**options**{

    tokenVocab = gSqlMiniLexer;

    language = Java;

}

programa: (consultaSelect)\*;

consultaSelect : SELECT columnas FROM nombreTabla sentenciaWhere? sentenciaOrderBy?;

columnas: nombreColumna (COMA nombreColumna)\*;

nombreColumna: ID;

nombreTabla: ID;

sentenciaWhere: WHERE expresion;

sentenciaOrderBy: ORDER BY nombreColumna metodoOrdenacion;

metodoOrdenacion: ASC | DESC;

expresion: ABREPARENTESIS expresion CIERRAPARENTESIS

         | expresion AND expresion

         | expresion OR expresion

         | nombreColumna IGUALQUE (NUMERO | STRING)

         | nombreColumna (MAYORQUE | MAYORIGUALQUE) NUMERO;

**consultaSelect**: Esta es la regla principal que representa una consulta SELECT en SQLMini. Incluye todas las posibles partes de una consulta: la **selección de columnas**, **la tabla de donde se seleccionan los datos**, la **cláusula WHERE (opcional)** y la **cláusula ORDER BY (opcional).**

**columnas**: Define la lista de columnas que se seleccionarán en la consulta. Puede haber una o más columnas separadas por comas.

**nombreColumna**: Representa el nombre de una columna. Esto es esencial para especificar qué datos se deben recuperar de la tabla.

**nombreTabla**: Define el nombre de la tabla desde donde se seleccionarán los datos.

**sentenciaWhere**: Representa la cláusula WHERE que filtra los resultados de la consulta según una condición dada.

**sentenciaOrderBy**: Representa la cláusula ORDER BY que permite ordenar los resultados de la consulta.

**metodoOrdenacion**: Define si se ordena de forma ascendente (ASC) o descendente (DESC) según una columna específica.

**expresion**: Esta regla define las diferentes expresiones que pueden aparecer en la cláusula WHERE. Pueden ser comparaciones, operaciones lógicas (AND y OR) o expresiones entre paréntesis.

Reglas alternativas en expresion: Se incluyen diferentes formas de expresiones, como comparaciones entre columnas y valores, así como operaciones lógicas.

Una vez definido el lexer y parser vamos a declarar todas las consultas de las que se va a generar el árbol en un programa el cual hemos llamado **EjemploConsulta.prog:**

Captura de pantalla de computadora

Descripción generada automáticamente

Declaramos las 8 consultas de la parte obligatoria de la practica.

Una vez creados estos tres elementos, pasamos a la compilación para poder mostrar el resultado del árbol, los pasos para esto son:

**antlr gSqlMiniLexer.g4**

**antlr gSqlMiniParser.g4**

**javac \*.java**

**grun gSqlMini programa -tokens -gui [EjemploConsulta.prog]**

Captura de pantalla de computadora

Descripción generada automáticamente

Pantalla de computadora con fondo negro

Descripción generada automáticamente con confianza mediaPantalla de computadora con fondo negro

Descripción generada automáticamente con confianza media

Una vez que se despliegan todos los tokens y se compila todo vemos como se nos generan los archivos .class, .java,

Captura de pantalla de computadora

Descripción generada automáticamente

Ahora vamos a mostrar el árbol consulta por consulta:

1. **SELECT name FROM employees**

Texto

Descripción generada automáticamente

1. **SELECT name, age FROM employees**

Texto

Descripción generada automáticamente

1. **SELECT name, age FROM employees WHERE age > 30**

Texto

Descripción generada automáticamente

1. **SELECT name, age FROM employees ORDER BY age ASC**

Texto

Descripción generada automáticamente

1. **SELECT name, age FROM employees WHERE age >= 30 ORDER BY name DESC**

Texto, Aplicación, Word

Descripción generada automáticamente

1. **SELECT \* FROM products WHERE (price >= 100 AND stock > 0) OR (category = 'Electronics')**

Imagen que contiene Diagrama

Descripción generada automáticamente



1. **SELECT name, description FROM products WHERE (price >= 10.5 AND stock > 0) OR (category = 'Electronics') ORDER BY name ASC**

Texto

Descripción generada automáticamente



1. **SELECT name, age FROM employees WHERE (age > 30 AND (department = 'HR' OR department = 'Finance')) OR (salary > 50000)**

Texto

Descripción generada automáticamente

## **1.2** **Linguine**

**2. Segunda parte: iniciación a la tabla de símbolos (1 punto)**