Manual Técnico para Analizador Léxico y Sintáctico SQL

Introducción

Este documento describe el funcionamiento y la implementación de un analizador léxico y sintáctico en Java, diseñado con JDK 17 y utilizando JFlex para la generación del analizador léxico. El sistema procesa estructuras SQL básicas para la creación, modificación y eliminación de tablas en una base de datos.

Objetivo

El objetivo del analizador es identificar y validar tokens y estructuras de comandos SQL de tipo DDL y DML, generando diagramas gráficos de tablas y proporcionando retroalimentación sobre errores léxicos y sintácticos en las entradas de SQL.

Tecnologías Utilizadas

- **Java JDK 17**: Versión de Java utilizada para la implementación.
- **JFlex**: Herramienta generadora de analizadores léxicos en Java.
- **Graphviz**: Librería de visualización para mostrar diagramas de relaciones de las tablas SQL.

Analizador Léxico (Jflex)

El archivo JFlex define el analizador léxico en Java, el cual se utiliza para procesar tokens de un lenguaje SQL. Este analizador genera una lista de tokens clasificados en diferentes tipos, permitiendo el procesamiento posterior en el analizador sintáctico.

Estructura del Archivo

El archivo se organiza en tres secciones principales:

1. **Sección de Código Java y Variables Globales**
2. **Declaración de Expresiones Regulares y Configuración de JFlex**
3. **Reglas de Tokenización**

Sección de Código Java y Variables Globales

En esta sección, se definen variables y métodos auxiliares en Java que serán utilizados por JFlex para almacenar y categorizar los tokens generados.

Métodos:

- `addList(Token token)`: Agrega un token a `lista` y `listaVista`.
- `addListaErrores(Token token)`: Agrega tokens que corresponden a errores a `lista` y `listaErrores`.
- `addListaVista(Token token)`: Agrega un token solo a `listaVista`.

3. Reglas de Tokenización

Esta sección especifica expresiones regulares (regex) para identificar tokens y las acciones correspondientes para cada tipo de token. Cada patrón regex está asociado a una acción en la que se crea un nuevo token y se añade a las listas definidas.

- **N**: Define los caracteres numéricos (`0-9`).
- **LMINS**: Define letras minúsculas (`a-z`).
- **IDENTIFICADOR**: Define un identificador SQL, compuesto de letras, guiones bajos o números.
- **FECHA**: Define el patrón de una fecha (`YYYY-MM-DD`).
- **RESERVED**: Palabras reservadas en SQL como `CREATE`, `TABLE`, `SELECT`, etc.
- **DATO**: Tipos de datos SQL (`INTEGER`, `VARCHAR`, etc.).
- **BOOLEANO**: Valores booleanos (`TRUE` y `FALSE`).
- **AGREGACION**: Funciones de agregación SQL (`SUM`, `AVG`, etc.).
- **LOGICOS**: Operadores lógicos (`AND`, `OR`, `NOT`).
- **SIGNOS**: Símbolos usados en SQL (`(`, `)`, `;`, `=`, etc.).
- **ARITMETICOS**: Operadores aritméticos (`+`, `-`, `*`, `/`).
- **RELACIONALES**: Operadores relacionales (`<=`, `>=`, `<`, `>`).

Estructura de la Clase `AnalizadorSintactico`

La clase `AnalizadorSintactico` es la principal en el sistema y se encarga de procesar una lista de tokens. Identifica las operaciones de DDL y DML y delega su procesamiento a analizadores secundarios (`AnalizadorDDL` y `AnalizadorDML`). A continuación, se explica cada sección de la clase.

Atributos

- `FIN_BLOQUE`: Define el delimitador `;` que marca el fin de cada bloque SQL.
- `INICIO`: Lista de palabras reservadas que representan el inicio de comandos SQL válidos.
- `tokens`: Lista de tokens generados previamente por el analizador léxico.
- `currentIndex`: Índice actual en el arreglo de tokens.
- `tokensError`: Objeto para almacenar y gestionar errores sintácticos.

- `analizadorDDL`: Analizador para comandos SQL de tipo DDL.
- `analizadorDML`: Analizador para comandos SQL de tipo DML.
- `operaciones`: Clase auxiliar para operaciones.

Métodos

Constructor: `public AnalizadorSintactico(List<Token> tokens)`

Inicializa el analizador con una lista de tokens. Si la lista está vacía, el analizador no se inicializa. Crea instancias de `ListSyntaxError`, `AnalizadorDDL`, y `AnalizadorDML`.

`void next()`

Avanza el índice actual para procesar el siguiente token en la lista.

`boolean isEnd()`

Retorna `true` si el índice actual ha alcanzado o superado el final de la lista de tokens.

`void analizar()`

1. **Función**: Es el núcleo de la lógica sintáctica.
2. **Flujo de Trabajo**:
    - Valida que cada token sea un inicio válido.
    - En función de la palabra reservada (por ejemplo, `CREATE`, `ALTER`, `DROP`), envía el bloque de tokens correspondiente al analizador adecuado (`analizadorDDL` o `analizadorDML`).
    - Si un token no coincide con un inicio válido, se registra un error en `tokensError`.
    - Al finalizar, muestra errores de sintaxis y genera un diagrama con las tablas creadas, modificadas o eliminadas.

`Pila getTokensBloque()`

Obtiene un bloque de tokens hasta encontrar el delimitador `FIN_BLOQUE` (`;`). Este bloque se envía luego a los analizadores `DDL` y `DML`.

`void printTables()`

Muestra los diagramas de tablas creadas, modificadas o eliminadas en un `JFrame`. Cada tabla es representada como una imagen generada a partir de un diagrama `DOT` de Graphviz.

`ListSyntaxError getTokensError()`

Retorna el objeto `tokensError` para acceso externo.

Estructura de la Clase `AnalizadorDDL`

La clase AnalizadorDDL es una parte central del analizador sintáctico para sentencias de manipulación de datos en SQL (DDL - Data Definition Language). Este analizador se encarga de procesar instrucciones de creación, modificación y eliminación de bases de datos y tablas, validando la sintaxis y generando estructuras correspondientes en el sistema. A continuación se explica su estructura:

`Atributos de la clase`

1. **`ACEPTED`**: Arreglo de palabras reservadas aceptadas para el análisis (`DATABASE` y `TABLE`), que define los tipos de sentencias válidas.
2. **`DBCreadas`**: Lista de bases de datos creadas.
3. **`tablasCreadas`**: Lista de tablas creadas.
4. **`tablasModificadas`**: Lista de modificaciones a tablas.
5. **`tablasBorradas`**: Lista de tablas eliminadas.
6. **`tokensError`**: Lista de errores sintácticos encontrados durante el análisis.
7. **`helper`**: Instancia de la clase `AnalizadorHelper` que facilita la validación de tokens.
8. **`operaciones`**: Instancia de la clase `Operaciones` para registrar el número de operaciones de creación, alteración y eliminación.

Constructor

El constructor inicializa las listas de bases de datos y tablas, así como las estructuras de ayuda y control de errores.

Métodos principales

1. Método `analizarCreate(Pila pila)`

Este método identifica y analiza sentencias `CREATE` para bases de datos y tablas, extrayendo el nombre de la base de datos o tabla y validando la sintaxis. Dependiendo de si se crea una base de datos o una tabla, delega el análisis a métodos específicos como `analizarCreacionDB` o `analizarCreacionTabla`.

2. Método `analizarCreacionDB(Pila pila)`

Procesa la creación de una base de datos con el formato `CREATE DATABASE <identificador>;`. Verifica que el identificador esté presente y termina la sentencia con `;`. Si la sintaxis es correcta, registra la base de datos en `DBCreadas`.

3. Método `analizarCreacionTabla(Pila pila)`

Procesa la creación de tablas siguiendo el formato `CREATE TABLE <identificador> (<estructura_declaracion>, [<estructura_llaves>?]);`. Extrae el nombre de la tabla, declaraciones de campos y posibles restricciones de llave (`PRIMARY` o `FOREIGN KEY`). Luego, registra la tabla en `tablasCreadas` si no hay errores.

4. Método `analizarAlter(Pila pila)`

Interpreta sentencias `ALTER TABLE`, como agregar (`ADD`), alterar (`ALTER`), o eliminar (`DROP`) columnas y restricciones. Valida que se usen correctamente las palabras reservadas y los identificadores en las modificaciones. Si todo es correcto, la modificación se agrega a `tablasModificadas`.

5. Método `analizarDrop(Pila pila)`

Interpreta y valida sentencias de eliminación de tablas con el formato `DROP TABLE IF EXISTS <identificador> CASCADE;`. Si la sintaxis es válida, añade la tabla a `tablasBorradas`.

Métodos de soporte

1. **`analizarDato(Pila pila)`**: Valida el tipo de dato en una declaración de columna, asegurando que sea un tipo de dato permitido.
2. **`analizarDeclaracion(Pila pila)`** y **`analizarLlave(Pila pila, Llave llave)`**: Procesan declaraciones de columnas y llaves en tablas, verificando que la sintaxis y los tipos sean correctos.

Esta clase es clave en el procesamiento de sentencias DDL, encargándose de interpretar instrucciones SQL específicas y construir estructuras en memoria para su posterior uso o persistencia en la base de datos.

Estructura de la Clase AnalizadorDML

La clase AnalizadorDML se encarga de analizar y procesar instrucciones DML (Data Manipulation Language) para un lenguaje SQL, principalmente para los comandos INSERT, SELECT, UPDATE y DELETE. Utiliza una serie de métodos y estructuras para verificar la validez de cada instrucción en función de los tokens y sus tipos. Aquí está la estructura principal y las funciones clave:

Atributos de la Clase

- **`operaciones`**: Una instancia de `Operaciones` que posiblemente realiza operaciones estadísticas o contables sobre los comandos ejecutados (por ejemplo, contar las veces que se ha ejecutado un `SELECT` o `UPDATE`).
- **`tokensError`**: Una instancia de `ListSyntaxError`, que guarda los errores de sintaxis encontrados durante el análisis de las instrucciones.
- **`helper`**: Una instancia de `AnalizadorHelper`, utilizada como apoyo para validar valores y tipos de tokens.
- **`inserts`**: Una lista de objetos `Insert` donde se almacenan las operaciones de inserción validadas.

Principales Métodos

1. **Métodos para `INSERT`**:

- `analizarInsert(Pila pila)`: Valida y analiza una instrucción `INSERT`. Extrae la tabla de destino y las columnas, luego valida los valores entre paréntesis.
- `getIdentificadores(Pila pila)`: Obtiene los identificadores de columnas entre paréntesis.
- `getDato(Pila pila)`: Extrae un dato válido para ser insertado, que puede ser un entero, decimal, fecha, cadena, o valor booleano.
- `getOperador(Pila pila)`: Verifica si el token es un operador válido (ej., `+`, `-`, `AND`, `OR`).

2. **Métodos para `SELECT`**:

    - `analizarSelect(Pila pila)`: Valida y procesa una instrucción `SELECT`, verificando que incluya `FROM` y opcionalmente otros componentes como `WHERE`, `GROUP BY`, y `ORDER BY`.
    - `analizarSeleccionColumna(Pila pila)`: Procesa las columnas a seleccionar, permitiendo el uso de funciones de agregación (`SUM`, `AVG`, etc.) y alias.
    - `analizarSentencia(Pila pila)`: Procesa las posibles cláusulas adicionales (`JOIN`, `WHERE`, `GROUP BY`, `ORDER BY`, `LIMIT`) dentro de la instrucción `SELECT`.

3. **Métodos para `UPDATE`**:

    - `analizarUpdate(Pila pila)`: Analiza una instrucción `UPDATE`, procesando los valores de las columnas que serán modificadas y la cláusula `WHERE`.
    - `analizarSet(Pila pila)`: Procesa los valores que serán asignados en la instrucción `UPDATE`.

4. **Métodos para `DELETE`**:

    - `analizarDelete(Pila pila)`: Procesa y valida una instrucción `DELETE`, asegurándose de que siga la estructura correcta y opcionalmente que contenga una cláusula `WHERE`.