MANUAL TÉCNICO
ANALIZADOR SQL

LENGUAJE: JAVA

JDK : 17

FRONTEND

Clase Interfaz

Descripción General

La clase Interfaz extiende JFrame y representa una interfaz gráfica de usuario (GUI) para un analizador gráfico. Contiene varios componentes Swing y los organiza dentro del marco.

Componentes Swing

JTextPane: textPane1 es un componente de texto que puede mostrar texto con varios atributos como fuentes, colores, etc.

JLabel: Usado para mostrar texto. posLabel muestra la posición actual del cursor en el panel de texto (fila y columna). errorLabel muestra mensajes de error, inicialmente vacío y configurado en texto rojo.

JButton: boton es un componente de botón que desencadena una acción cuando se hace clic en él.

Componentes del Menú

JMenuBar: La barra de menú principal del marco, que contiene tres menús: archivoMenu, generarGraficoMenu, y reportesMenu.

JMenu: Representa un menú, como "Archivo", "Generar Gráfico", y "Reportes".

JMenuItem: Representa un elemento dentro de un menú, como "Abrir", "Crear Grafico", "Reporte de errores de Token", etc.

Constructor Interfaz()

UIManager.setLookAndFeel(new FlatLightLaf()): Establece la apariencia y sensación de la GUI en FlatLightLaf.

Configuración de JMenuBar y JMenu: Configura y añade elementos de menú a la barra de menú.

Configuración de JTextPane y JScrollPane: Inicializa textPane1 y lo envuelve en un panel de desplazamiento para la capacidad de desplazamiento. También añade la numeración de líneas mediante LineNumberingTextArea.

Configuración de JLabel: Inicializa posLabel y errorLabel.

Añadir CaretListener a JTextPane: Añade un listener para actualizar la posición del cursor en posLabel.

Configuración del Layout: Usa GridBagLayout para organizar los componentes dentro del marco.

Configuración de ActionListener: Añade listeners para los elementos del menú y el botón.

Métodos y Su Propósito
ActionListener para el elemento de menú "Abrir": Abre un selector de archivos, lee el archivo seleccionado y muestra su contenido en textPane1.

ActionListener para boton: Analiza el código fuente en textPane1 usando Lexer y AnalizadorSintactico2, actualiza errorLabel, y resalta errores en el panel de texto.

ActionListener para el elemento de menú "Crear Grafico": Abre un selector de archivos para guardar el gráfico generado, analiza el código fuente y genera un gráfico.

ActionListener para el elemento de menú "Reporte de errores de Token": Analiza el código fuente en busca de errores léxicos y muestra un informe.

ActionListener para el elemento de menú "Reporte de tablas encontradas": Analiza el código fuente en busca de tablas y muestra un informe.

ActionListener para el elemento de menú "Reporte de operaciones": Analiza el código fuente en busca de operaciones y muestra un informe.

Clase LineNumberingTextArea
Descripción General
La clase LineNumberingTextArea extiende JTextPane y se utiliza para añadir numeración de líneas a un componente JTextPane. Esto facilita la visualización y el seguimiento del contenido en el panel de texto.

Componentes Swing
JTextPane: La clase extiende JTextPane y contiene una instancia de JTextPane llamada textPane para la cual se proporcionará la numeración de líneas.

Constructor LineNumberingTextArea(JTextPane textPane)
textPane: Inicializa el campo textPane con el panel de texto proporcionado.

DocumentListener: Añade un DocumentListener al documento de textPane para actualizar los números de línea cada vez que el contenido del documento cambia.

Métodos
updateLineNumbers()

Descripción: Actualiza la numeración de líneas basada en el contenido actual de textPane.

Funcionamiento:

Crea un StringBuilder para almacenar los números de línea.

Obtiene el número de líneas en el documento de textPane.

Recorre las líneas y las añade a StringBuilder.

Establece el texto del LineNumberingTextArea con la numeración de líneas generada.

BACKEND

Lexer

Clase Lexer
Descripción General
La clase Lexer es responsable de realizar el análisis léxico del código fuente. Identifica tokens en el código y clasifica los distintos componentes según sus tipos. Utiliza una serie de expresiones regulares y reglas de escaneo para reconocer diferentes elementos del lenguaje.

Componentes y Atributos
Lista de Tokens: private List<Token> lista almacena los tokens identificados.

Lista de Errores: private List<Token> listaErrores almacena los tokens que son considerados errores léxicos.

Métodos
addList(Token token)
Descripción: Añade un token a la lista de tokens.
addListaErrores(Token token)
Descripción: Añade un token a la lista de errores.
getLista()
Descripción: Devuelve la lista de tokens.
getListaErrores()
Descripción: Devuelve la lista de errores.
Reglas de Palabras Reservadas
Define las palabras reservadas que reconoce el lexer:

PALABRAS_RESERVADAS: Incluye palabras como "BETWEEN", "USE", "CREATE", "DATABASE", etc.

Reglas de Tipos de Datos
Define los tipos de datos que reconoce el lexer:

TIPO_DATO: Incluye "INT", "VARCHAR", "DATE", etc.

Expresiones Regulares
Define patrones para identificar diferentes elementos del código:
ENTERO: [0-9]+
DECIMAL: {ENTERO}("."{ENTERO}+)
IDENTIFICADOR: [a-z]+([a-z0-9_]*)
ESPACIOS: [" "\r\t\b\n]
FECHA: """{ENTERO}{4}"-"{ENTERO}{2}"-"{ENTERO}{2}"""
CADENA: """[^']*"""
Reglas de Escaneo de Expresiones
Define cómo se identifican y clasifican los tokens en el código:

Palabras Reservadas: {PALABRAS_RESERVADAS} { ... }
Tipos de Datos: {TIPO_DATO} { ... }
Identificadores: {IDENTIFICADOR} { ... }
Números Enteros: {ENTERO} { ... }
Números Decimales: {DECIMAL} { ... }
Literals de Fecha: {FECHA} { ... }
Cadenas de Texto: {CADENA} { ... }
Booleanos: {BOOLEANO} { ... }
Funciones de Agrupación: {FUNCION_AGRUPACION} { ... }
Signos: {SIGNOS} { ... }
Operadores Aritméticos: {ARITMETICOS} { ... }
Operadores Relacionales: {RELACIONALES} { ... }
Operadores Lógicos: {LOGICOS} { ... }
Comentarios de Línea: {COMENTARIO_LINEA} { ... }
Errores Léxicos: .

Propósito de los Métodos y Componentes
addList(Token token): Añade un token válido a la lista de tokens.
addListaErrores(Token token): Añade un token inválido a la lista de errores.
getLista(): Devuelve la lista de tokens válidos.
getListaErrores(): Devuelve la lista de errores léxicos.

Clase Token
Descripción General
La clase Token representa una unidad léxica en el análisis de código fuente. Cada instancia de esta clase contiene información sobre un token específico, incluyendo su tipo, valor, posición en el código y atributos adicionales como el color y el lexema.

Atributos
TokenType type: Tipo del token, que puede ser una palabra reservada, un identificador, un número, etc.
String value: Valor del token.

int linea: Línea en la que se encuentra el token en el código fuente.
int columna: Columna en la que se encuentra el token en el código fuente.
String color: Color asociado al token, usado para resaltar el token en la interfaz de usuario.
String lexema: Parte del código fuente correspondiente al token.
String tipo: Tipo de dato del token.

Constructores
Token(String lexema, int linea, int columna, String tipo, String color)

Descripción: Constructor que inicializa un token con su lexema, línea, columna, tipo y color.

Parámetros:

lexema: Parte del código correspondiente al token.
linea: Línea del token en el código.
columna: Columna del token en el código.
tipo: Tipo de dato del token.
color: Color usado para resaltar el token.
Token(TokenType type, String value, int linea, int columna, String color)
Descripción: Constructor que inicializa un token con su tipo, valor, línea, columna y color.
Parámetros:
type: Tipo del token.
value: Valor del token.
linea: Línea del token en el código.
columna: Columna del token en el código.
color: Color usado para resaltar el token.

Métodos
getType(): Devuelve el tipo del token.
setType(TokenType type): Establece el tipo del token.
getValue(): Devuelve el valor del token.
setValue(String value): Establece el valor del token.
getLinea(): Devuelve la línea del token.
setLinea(int linea): Establece la línea del token.
getColumna(): Devuelve la columna del token.
setColumna(int columna): Establece la columna del token.
getColor(): Devuelve el color del token.
setColor(String color): Establece el color del token.
getLexema(): Devuelve el lexema del token.
toString()
Descripción: Devuelve una representación en cadena del token.
Salida: String

Clase TokenType

Descripción General

El enumerador TokenType define una lista de tipos de tokens que el lexer puede identificar en el código fuente. Cada tipo de token representa una categoría específica de elementos del lenguaje, tales como palabras reservadas, identificadores, operadores, y más.

Uso y Funcionamiento

El TokenType se utiliza en la clase Lexer para clasificar los diferentes tokens encontrados durante el análisis léxico. Cada token identificado es categorizado de acuerdo a estos tipos, lo que facilita su manejo y procesamiento posterior en el análisis sintáctico y semántico.

Tipos de Tokens

Palabras Reservadas

PALABRA_RESERVADA: Representa palabras clave del lenguaje como CREATE, SELECT, UPDATE, etc.

Tipos de Datos

TIPO_DATO: Incluye tipos de datos como INT, VARCHAR, BOOLEAN.

Identificadores

IDENTIFICADOR: Representa nombres de variables, tablas, y otros identificadores.

Números

NUMERO_ENTERO: Representa números enteros.

NUMERO_DECIMAL: Representa números decimales.

Literales

LITERAL_CADENA: Cadenas de texto.

LITERAL_FECHA: Representa fechas.

LITERAL_BOOLEANO: Representa valores booleanos (TRUE, FALSE).

Funciones de Agrupación

FUNCION_AGRUPACION: Funciones como SUM, COUNT.

Signos y Operadores

Signos:

SIGNO: Paréntesis, comas, puntos, punto y coma, etc.

Operadores Aritméticos:

OPERADOR_ARITMETICO, OPERADOR_ARITMETICO_SUMA, OPERADOR_ARITMETICO_RESTA, OPERADOR_ARITMETICO_MULTIPLICACION, OPERADOR_ARITMETICO_DIVISION.

Operadores Relacionales:

OPERADOR_RELACIONAL, OPERADOR_RELACIONAL_IGUAL, OPERADOR_RELACIONAL_DIFERENTE, OPERADOR_RELACIONAL_MENOR, OPERADOR_RELACIONAL_MENOR_IGUAL, OPERADOR_RELACIONAL_MAYOR, OPERADOR_RELACIONAL_MAYOR_IGUAL.

Operadores Lógicos:

OPERADOR_LOGICO, OPERADOR_LOGICO_AND, OPERADOR_LOGICO_OR, OPERADOR_LOGICO_NOT.

Comentarios
COMENTARIO_LINEA: Comentarios de una sola línea.
Espacios
ESPACIO: Espacios y caracteres en blanco.
Errores
ERROR: Cualquier token no identificado o erróneo.
Fin de Archivo

YYEOF: Representa el final del archivo.

Propósito en el Análisis Léxico
Cada TokenType proporciona una forma estructurada de clasificar y manejar los diferentes elementos del código fuente. Esto permite que el lexer distinga entre palabras reservadas, identificadores, operadores, etc., y categorice correctamente cada parte del código, facilitando el análisis sintáctico y la generación de reportes precisos.

Clase Grafica
Uso y Funcionamiento
La clase Grafica se utiliza para representar y manejar estructuras de tablas en un contexto gráfico. Aquí se describen sus principales funciones y cómo se utilizan:

Funcionalidades Principales
Representación de Tablas:

La clase permite crear objetos que representan tablas con un nombre y una lista de columnas.

Constructor: Grafica(String nombre) inicializa una nueva tabla con el nombre proporcionado y una lista vacía de columnas.

Gestión de Columnas:

Método: agregarColumna(String nombreColumna) añade una columna a la lista de columnas de la tabla.

Esto es útil para construir la estructura de una tabla basándose en una definición de base de datos.

Generación de Diagramas con Graphviz:

Método: generarGraphviz(String codigoFuente, String outputFilePath) analiza el código fuente en busca de definiciones de tablas y genera un archivo .dot para su conversión en un gráfico de imagen mediante Graphviz.

Patrón de Búsqueda: Utiliza expresiones regulares para encontrar definiciones de tablas en el código fuente.

Salida: Crea un archivo de gráfico visual que muestra las tablas y sus columnas.

Uso: Llamar a este método con el código fuente y la ruta de salida deseada genera el gráfico visual de las tablas encontradas.

Visualización de Reportes de Tablas:

Método: mostrarReporteTablas(List<Grafica> tablas) genera una ventana que muestra un reporte tabular de las tablas y sus columnas.

Interfaz Gráfica: Usa JFrame, DefaultTableModel, y JTable para presentar la información en una tabla gráfica.

Uso: Proporcionar una lista de objetos Grafica al método para visualizar un reporte detallado en una interfaz gráfica de usuario.

Análisis de Tablas en el Código Fuente:

Método: analizarTablas(String codigoFuente) analiza el código fuente para identificar todas las definiciones de tablas y devuelve una lista de objetos Grafica.

Patrón de Búsqueda: Utiliza expresiones regulares similares a generarGraphviz para extraer la información de las tablas.

Salida: Devuelve una lista de tablas (List<Grafica>) encontradas en el código fuente.

Uso: Ideal para extraer y manipular información estructural de las tablas definidas en el código fuente.

Con estas funcionalidades, la clase Grafica permite tanto la manipulación y visualización de estructuras de tablas en una base de datos, como la generación de gráficos visuales mediante Graphviz.

Clase Operaciones

Uso y Funcionamiento

La clase Operaciones se centra en el análisis y conteo de diferentes tipos de operaciones en el código fuente. A continuación, se describen sus principales funciones y cómo se utilizan:

Funcionalidades Principales
Inicialización de Operaciones:

Constructor: Operaciones(String tipoOperacion) crea una nueva instancia con el tipo de operación especificado y un contador inicializado en 0.

Uso: Cada operación (como CREATE, DELETE, etc.) se representa por una instancia de esta clase.

Conteo de Operaciones:

Método: incrementarConteo() incrementa el contador de la operación en uno.

Uso: Cada vez que se encuentra una operación en el código fuente, se llama a este método para actualizar su conteo.

Acceso a los Atributos:

Método: getTipoOperacion() devuelve el tipo de operación.

Método: getConteo() devuelve el conteo actual de la operación.

Uso: Estos métodos son útiles para obtener información sobre el tipo de operación y cuántas veces ha sido encontrada en el código fuente.

Análisis de Operaciones en el Código Fuente:

Método: analizarOperaciones(String codigoFuente) analiza el código fuente para encontrar diferentes tipos de operaciones y cuenta cuántas veces aparece cada una.

Proceso:

Inicializa una lista con los tipos de operaciones más comunes (CREATE, DELETE, UPDATE, SELECT, ALTER).

Utiliza expresiones regulares para encontrar y contar cada operación en el código fuente.

Salida: Devuelve una lista de Operaciones con el conteo actualizado para cada tipo de operación.

Uso: Ideal para realizar un análisis detallado de las operaciones en el código fuente y generar reportes.

Visualización de Reportes de Operaciones:

Método: mostrarReporteOperaciones(List<Operaciones> operaciones) genera una ventana que muestra un reporte tabular de las operaciones y sus conteos.

Interfaz Gráfica: Usa JFrame, DefaultTableModel, y JTable para presentar la información en una tabla gráfica.

Uso: Proporcionar una lista de objetos Operaciones al método para visualizar un reporte detallado en una interfaz gráfica de usuario.

Con estas funcionalidades, la clase Operaciones facilita el análisis y la visualización de las diferentes operaciones realizadas en un código fuente, permitiendo un mejor entendimiento y documentación del mismo.

Clase ReporteErrores

Uso y Funcionamiento

La clase ReporteErrores se encarga de generar una interfaz gráfica para mostrar un reporte detallado de los errores léxicos encontrados en el código fuente. Este reporte se presenta en una tabla, facilitando la identificación y corrección de errores.

Funcionalidades Principales

Inicialización del Reporte de Errores:

Constructor: ReporteErrores(List<Token> listaErrores) crea una nueva ventana que muestra los errores léxicos en una tabla.

Uso: Proporcionar una lista de tokens con errores al constructor para generar el reporte.

Configuración de la Interfaz Gráfica:

Diseño: Utiliza BorderLayout para organizar los componentes dentro de la ventana.

Componentes:

DefaultTableModel: Modelo de datos para la tabla de errores.

JTable: Tabla que muestra los errores léxicos.

JScrollPane: Panel de desplazamiento que contiene la tabla, permitiendo visualizar un gran número de errores.

Agregar Errores a la Tabla:

Método: agregarErroresTabla(List<Token> listaErrores) añade los errores léxicos a la tabla.

Proceso:

Recorre la lista de errores y añade cada error como una fila en el modelo de la tabla.

Cada fila contiene el lexema, línea, columna y una descripción del error ("Error en token").

Uso: Este método es llamado dentro del constructor para inicializar la tabla con los errores proporcionados.

Visualización del Reporte:

Ventana: La clase extiende JFrame para crear una ventana independiente que muestra el reporte.

Tamaño y Comportamiento:

Configura el tamaño de la ventana (600x400 píxeles).

Establece el comportamiento de cierre (DISPOSE_ON_CLOSE), que cierra solo la ventana actual sin afectar la aplicación principal.

Visibilidad: La ventana se hace visible al final del constructor (setVisible(true)).

Clase ReporteErroresSintacticos

Uso y Funcionamiento

La clase ReporteErroresSintacticos se encarga de generar una interfaz gráfica para mostrar un reporte detallado de los errores sintácticos encontrados en el código fuente. Este reporte se presenta en una tabla, facilitando la identificación y corrección de errores sintácticos.

Funcionalidades Principales

Inicialización del Reporte de Errores Sintácticos:

Constructor: ReporteErroresSintacticos(List<SyntaxError> listaErrores) crea una nueva ventana que muestra los errores sintácticos en una tabla.

Uso: Proporcionar una lista de errores sintácticos al constructor para generar el reporte.

Configuración de la Interfaz Gráfica:

Diseño: Utiliza BorderLayout para organizar los componentes dentro de la ventana.

Componentes:

DefaultTableModel: Modelo de datos para la tabla de errores.

JTable: Tabla que muestra los errores sintácticos.

JScrollPane: Panel de desplazamiento que contiene la tabla, permitiendo visualizar un gran número de errores.

Agregar Errores a la Tabla:

Método: agregarErroresTabla(List<SyntaxError> listaErrores) añade los errores sintácticos a la tabla.

Proceso:

Recorre la lista de errores y añade cada error como una fila en el modelo de la tabla.

Cada fila contiene el lexema, el tipo de token, línea, columna y una descripción del error.

Uso: Este método es llamado dentro del constructor para inicializar la tabla con los errores proporcionados.

Visualización del Reporte:

Ventana: La clase extiende JFrame para crear una ventana independiente que muestra el reporte.

Tamaño y Comportamiento:

Configura el tamaño de la ventana (600x400 píxeles).

Establece el comportamiento de cierre (DISPOSE_ON_CLOSE), que cierra solo la ventana actual sin afectar la aplicación principal.

Visibilidad: La ventana se hace visible al final del constructor (setVisible(true)).

Clase SyntaxError

Uso y Funcionamiento

La clase SyntaxError se utiliza para representar errores sintácticos detectados durante el análisis de código fuente. Cada instancia de esta clase contiene información detallada sobre un error específico, incluyendo el token asociado y una descripción del error.

Funcionalidades Principales

Inicialización del Error Sintáctico:

Constructor: SyntaxError(Token token, String descripcion) crea una nueva instancia con el token problemático y una descripción del error.

Uso: Utilizar este constructor para encapsular información sobre errores sintácticos detectados.

Acceso a los Atributos:
Método: getToken() devuelve el token asociado al error.
Método: getDescripcion() devuelve la descripción del error.

Uso: Estos métodos permiten obtener información detallada sobre el error sintáctico para fines de diagnóstico y corrección.

Clase AnalizadorSintactico
Uso y Funcionamiento
La clase AnalizadorSintactico se encarga de realizar el análisis sintáctico del código fuente proporcionado. Utiliza una lista de tokens para verificar la corrección de la estructura gramatical del código y detectar errores sintácticos.

Funcionalidades Principales
Inicialización del Analizador Sintáctico:

Constructor: AnalizadorSintactico(List<Token> tokens) crea una nueva instancia con la lista de tokens proporcionada, inicializando el índice del token actual en 0 y la lista de errores en vacía.

Uso: Proporcionar la lista de tokens obtenidos del análisis léxico al constructor para comenzar el análisis sintáctico.

Proceso de Análisis Sintáctico:

Método: parse() realiza el análisis sintáctico del código fuente.

Ciclo Principal: Recorre la lista de tokens y trata de identificar el tipo de sentencia según el primer token.

Manejo de Errores: Si se encuentra una sentencia no reconocida, lanza una excepción ParserException y avanza hasta el siguiente punto y coma (;) para evitar ciclos infinitos.

Reporte de Errores: Imprime los errores encontrados al final del análisis, si los hay.

Uso: Llamar a este método para iniciar el análisis sintáctico del código fuente.

Avance de Sentencia:

Método: avanzarAlFinalDeSentencia() avanza el índice del token actual hasta el próximo punto y coma (;) para continuar con el análisis después de un error.

Uso: Se utiliza internamente para manejar errores y evitar que el análisis se detenga ante un error.

Análisis de Sentencias:

Método: parseSentencia() analiza una sentencia CREATE TABLE.

Estados: Utiliza una máquina de estados para validar la estructura de la sentencia.

Llamadas a Métodos Específicos: Llama a parseEstructuraDeTabla para analizar la estructura interna de una tabla.
Uso: Internamente, para descomponer y analizar una sentencia específica.
Análisis de Estructura de Tabla:
Método: parseEstructuraDeTabla() analiza la estructura de una tabla en una sentencia CREATE TABLE.

Estados: Utiliza una máquina de estados para validar la declaración de la estructura de la tabla.

Llamadas a Métodos Específicos: Llama a parseEstructuraDeDeclaracion para analizar las declaraciones de columnas dentro de una tabla.

Uso: Internamente, para descomponer y analizar la estructura de una tabla.

Análisis de Estructura de Declaración:

Método: parseEstructuraDeDeclaracion() analiza las declaraciones de columnas dentro de una estructura de tabla.

Estados: Utiliza una máquina de estados para validar las declaraciones de columnas, tipos de datos y restricciones opcionales.

Uso: Internamente, para validar y analizar las declaraciones de columnas en una tabla.

Métodos Auxiliares
Validación de Tipos de Datos:

Método: esTipoDeDatoValido(Token token) verifica si un token corresponde a un tipo de dato válido.

Uso: Internamente, para asegurar que los tipos de datos de las columnas sean correctos.

Verificación del Token Siguiente:

Método: verificarTokenSiguiente(String valorEsperado) verifica si el siguiente token coincide con el valor esperado.

Uso: Internamente, para validar la sintaxis y avanzar el índice del token si la verificación es exitosa.

Con estas funcionalidades, la clase AnalizadorSintactico facilita el análisis y la corrección de la estructura sintáctica del código fuente, asegurando que siga las reglas gramaticales del lenguaje definido.

Automatas

Clase AnalizadorSintactico2
Uso y Funcionamiento
La clase AnalizadorSintactico2 extiende las capacidades del análisis sintáctico al utilizar varios autómatas para procesar diferentes tipos de sentencias SQL. Estos autómatas se encargan de analizar tokens específicos y detectar errores sintácticos.

Funcionalidades Principales
Inicialización del Analizador Sintáctico:

Constructor: AnalizadorSintactico2() crea una nueva instancia inicializando autómatas para diferentes operaciones SQL (CREATE, INSERT, SELECT, UPDATE, DELETE, ALTER, DROP) y una lista de errores vacía.

Uso: Esta inicialización permite que diferentes tipos de operaciones sean procesadas adecuadamente por sus respectivos autómatas.

Análisis de Tokens:

Método: analizarTokens(List<Token> tokens) analiza y procesa una lista de tokens, utilizando autómatas específicos según el tipo de operación.

Proceso:

Itera sobre la lista de tokens.

Determina el autómata apropiado basado en el valor del token.

Procesa el token con el autómata correspondiente.

Agrega el token procesado a la lista de tokens validados o registra un error si el token no es reconocido.

Salida: Devuelve una lista de tokens validados y actualiza la lista de errores si se encuentran tokens no reconocidos.

Uso: Llamar a este método con la lista de tokens para realizar el análisis inicial de cada token y clasificarlos según el tipo de operación.

Proceso de Análisis Sintáctico:
Método: parse(List<Token> tokens) realiza el análisis sintáctico completo.
Proceso:
Llama a analizarTokens para obtener la lista de tokens validados.
Imprime los errores sintácticos si se encuentran.
Informa sobre el éxito del análisis si no hay errores.
Uso: Este método es el punto de entrada principal para realizar el análisis sintáctico del código fuente.
Acceso a los Errores:
Método: getErrores() devuelve la lista de errores sintácticos encontrados durante el análisis.
Uso: Permite acceder a los errores sintácticos para su visualización y corrección posterior.

Clase AutomataActualizacion
Uso y Funcionamiento
La clase AutomataActualizacion se encarga de procesar tokens específicos relacionados con sentencias UPDATE en SQL. Utiliza una máquina de estados para validar la estructura sintáctica de estas sentencias y reconocer errores.

Funcionalidades Principales
Inicialización del Autómata:

Constructor: AutomataActualizacion() crea una nueva instancia del autómata y inicializa el estado actual en Q0 y un conjunto de datos primitivos (TRUE, FALSE).
Uso: La inicialización permite al autómata comenzar desde el estado inicial y reconocer datos primitivos como enteros, decimales, cadenas y fechas.
Validación de Datos Primitivos:
Método: esDatoPrimitivo(String token) verifica si un token es un dato primitivo válido.
Uso: Este método se utiliza internamente para validar valores en las sentencias UPDATE.
Procesamiento de Tokens:
Método: procesarToken(Token token) procesa un token utilizando la máquina de estados.

Estados:

Q0: Espera la palabra clave UPDATE.

Q1: Espera un identificador (nombre de la tabla).

Q2: Espera la palabra clave SET.

Q3: Espera un identificador (nombre de una columna).

Q4: Espera el signo =.

Q5: Espera un dato primitivo.

Q6: Maneja múltiples columnas (WHERE, ,, ;).

Q7: Espera un identificador (condición WHERE).

Q8: Espera el signo =.

Q9: Espera un dato primitivo (condición WHERE).

QF: Estado final, sentencia válida.

ERROR: Estado de error, sentencia no válida.

Salida: Devuelve un token con el tipo apropiado o un token de error si la sentencia no es válida.

Uso: Este método se utiliza para analizar sentencias UPDATE y verificar su validez sintáctica.

Máquina de Estados
La máquina de estados definida en AutomataActualizacion sigue una secuencia lógica para validar las sentencias UPDATE:

Comienza esperando la palabra UPDATE.

A continuación, espera el nombre de la tabla y la palabra SET.
Luego, espera el nombre de una columna, seguido de = y un valor.
Maneja múltiples columnas y condiciones WHERE.
Termina cuando encuentra un ; o una condición de WHERE válida.

Ejemplo de Flujo de Estados
Estado Q0: Se espera UPDATE.

Estado Q1: Se espera el nombre de la tabla.
Estado Q2: Se espera SET.
Estado Q3: Se espera el nombre de una columna.
Estado Q4: Se espera =.
Estado Q5: Se espera un valor.
Estado Q6: Maneja WHERE, , o ;.
Estado Q7: Se espera el nombre de una columna para WHERE.
Estado Q8: Se espera = en WHERE.
Estado Q9: Se espera un valor en WHERE.
Estado QF: Sentencia válida.

Clase AutomataCREATE
Uso y Funcionamiento
La clase AutomataCREATE se encarga de procesar tokens específicos relacionados con sentencias CREATE TABLE en SQL. Utiliza una máquina de estados para validar la estructura sintáctica de estas sentencias y reconocer errores.

Funcionalidades Principales
Inicialización del Autómata:

Constructor: AutomataCREATE() crea una nueva instancia del autómata e inicializa el estado actual en Q0.

Uso: La inicialización permite al autómata comenzar desde el estado inicial para procesar sentencias CREATE TABLE.

Procesamiento de Tokens:

Método: procesarToken(Token token) procesa un token utilizando la máquina de estados.

Estados:
Q0: Espera la palabra clave CREATE.
Q1: Espera la palabra clave TABLE.
Q2: Espera un identificador (nombre de la tabla).
Q3: Espera el signo (.
Q4: Estado final, sentencia válida.
ERROR: Estado de error, sentencia no válida.

Salida: Devuelve un token con el tipo apropiado o un token de error si la sentencia no es válida.

Uso: Este método se utiliza para analizar sentencias CREATE TABLE y verificar su validez sintáctica.

Máquina de Estados
La máquina de estados definida en AutomataCREATE sigue una secuencia lógica para validar las sentencias CREATE TABLE:

Comienza esperando la palabra CREATE.
A continuación, espera la palabra TABLE.
Luego, espera el nombre de la tabla.
Termina cuando encuentra el signo (.
Ejemplo de Flujo de Estados
Estado Q0: Se espera CREATE.
Estado Q1: Se espera TABLE.
Estado Q2: Se espera el nombre de la tabla.
Estado Q3: Se espera (.
Estado Q4: Sentencia válida.

Clase AutomataEliminacion
Uso y Funcionamiento
La clase AutomataEliminacion se encarga de procesar tokens específicos relacionados con sentencias DELETE en SQL. Utiliza una máquina de estados para validar la estructura sintáctica de estas sentencias y reconocer errores.

Funcionalidades Principales
Inicialización del Autómata:

Constructor: AutomataEliminacion() crea una nueva instancia del autómata e inicializa el estado actual en Q0.

Uso: La inicialización permite al autómata comenzar desde el estado inicial para procesar sentencias DELETE.

Procesamiento de Tokens:

Método: procesarToken(Token token) procesa un token utilizando la máquina de estados.

Estados:

Q0: Espera la palabra clave DELETE.
Q1: Espera la palabra clave FROM.
Q2: Espera un identificador (nombre de la tabla).
Q3: Maneja las condiciones WHERE y el fin de la sentencia ;.
Q4: Espera un identificador en la condición WHERE.
Q5: Espera un operador relacional (=, <, >).
Q6: Espera un valor primitivo.
QF: Estado final, sentencia válida.
ERROR: Estado de error, sentencia no válida.

Salida: Devuelve un token con el tipo apropiado o un token de error si la sentencia no es válida.

Clase AutomataEstructuraDeclaracion
Uso y Funcionamiento
La clase AutomataEstructuraDeclaracion se encarga de procesar tokens relacionados con las declaraciones de estructura en SQL, validando su sintaxis mediante una máquina de estados. Este autómata maneja principalmente la declaración de columnas dentro de una tabla.

Funcionalidades Principales
Inicialización del Autómata:

Constructor: AutomataEstructuraDeclaracion() inicializa el estado actual en Q0 y los tipos de datos válidos que la declaración puede incluir.

Uso: Este constructor permite al autómata comenzar en un estado inicial preparado para validar declaraciones de estructura.

Tipos de Datos:

Método: inicializarTiposDeDato() añade al conjunto los tipos de datos que se pueden usar en las declaraciones, como SERIAL, INTEGER, VARCHAR, etc.

Método: esTipoDeDato(String token) verifica si un token es un tipo de dato válido.

Uso: Estos métodos aseguran que solo los tipos de datos definidos sean reconocidos durante el análisis.

Procesamiento de Tokens:

Método: procesarToken(Token token) analiza un token usando la máquina de estados para verificar la validez de la declaración de la estructura.

Estados:
Q0: Espera un identificador (nombre de la columna).
Q1: Espera un tipo de dato.
Q2: Maneja restricciones (PRIMARY, NOT, UNIQUE).
Q3: Espera las claves (KEY, NULL).
QF: Estado final, declaración válida.
ERROR: Estado de error, declaración no válida.
Salida: Devuelve un token validado o un token de error si la declaración no es válida.

Uso: Este método se utiliza para analizar la sintaxis de las declaraciones de columnas dentro de una tabla y asegurar que sigan las reglas gramaticales del lenguaje SQL.

Propósito de los Estados
Q0 a Q1: Identificación y validación del nombre de la columna y tipo de dato.

Q2 a Q3: Manejo de restricciones y validación de la estructura.

QF: Indica una declaración válida.

ERROR: Indica una declaración no válida.

Con estas funcionalidades, la clase AutomataEstructuraDeclaracion facilita el análisis y la validación de las declaraciones de columnas en una tabla SQL, asegurando que las declaraciones sigan las reglas sintácticas establecidas.