MANUAL TÉCNICO

ANALIZADOR LÉXICO Y SINTACTICO
PRACTICA 1 LENGUAJES FORMALES
APLICACIÓN DE ESCRITORIO

CRISTIAN ALEJANDRO ROLDÁN LÓPEZ
202147280

Especificaciones

Versión de Java utilizado: Java 21.

Versión de JDK: 21

Sistema Operativo Utilizado: Windows 11.

IDE utilizado: Apache NetBeans IDE 21.

Parse para CREATE

```
private void parseCreate() {
        avanzarToken();
        if (comprobaReservada(Token.DATABASE)) {
            parseCreacionBase();
        } else if (comprobaReservada(Token.TABLE)) {
            parseCreacionTabla();
        } else {
            Token token = getTokenActual();
            agregarErrorSintactico(token.getLexema(),
token.getTipoToken(), token.getFila(), token.getColumna(), "Se
esperaba 'DATABASE' o 'TABLE' después de 'CREATE'");
        }
    }
```

Gramatica para CREATE

G = (N, T, P, S)

N (No terminales):

{ Escritura }

T (Terminales):

{ DATABASE, TABLE, ; }

S (Símbolo inicial):

{ Escritura }

P (Producciones):

A → DATABASE| TABLE

Parse para Creacion de base de datos

```
private void parseCreacionBase() {
        Token token = getTokenActual();

        if (token == null) {
            System.out.println("Falta nombre de la base");
            return;
        }

        if (comproIdentificador()) {
            if (comprobaSigno(Token.PUNTO_COMA)) {
                System.out.println("Base de datos creada");
            }else{
                agregarErrorSintactico(token.getLexema(),
token.getTipoToken(), token.getFila(), token.getColumna(), "No
tiene punto y coma");
                System.out.println("No tiene punto y coma");
            }
        }else{
            agregarErrorSintactico(token.getLexema(),
token.getTipoToken(), token.getFila(), token.getColumna(), "No
tiene nombre la base de datos");
```

```
            System.out.println("No tiene nombre");
        }
    }
```

Gramática para CREATE DATABASE

G = (N, T, P, S)

N (Sin terminales) :

{ Escritura, NombreBase, Terminador }

T (Terminales) :

{ CREATE, DATABASE, <identificador>, ; }

S (Símbolo inicial) :

{ Escritura }

P (Producciones) :

Escritura → CREATE DATABASE NombreBase Terminador

NombreBase → <identificador>

Terminador → ;

Parse para creación de TABLE

```
private void parseCreacionTabla() {
    Token token = getTokenActual();  // Obtener el token actual
    if (token == null) {
        agregarErrorSintactico(token.getLexema(), token.getTipoToken(), token.getFila(), token.getColumna(), "No tiene nombre la tabla");
        System.out.println("Error: Falta el nombre de la tabla.");
        return;
    }

    String nombreTabla = token.getLexema();
    avanzarToken();

    List<Columna> columnas = new ArrayList<>();

    if (comprobaSigno(Token.PARENTESIS_APERTURA)) {
        boolean errorEnColumna = false;
        do {
            if (!parseColumna()) {
                agregarErrorSintactico(token.getLexema(), token.getTipoToken(), token.getFila(), token.getColumna(), "Error en la columna");
                System.out.println("Error en la declaración de una columna.");
```

```
                errorEnColumna = true;
                break;
            }
        } while (comprobaSigno(Token.COMA));


        if (!errorEnColumna &&
comprobaSigno(Token.PARENTESIS_CIERRE) &&
comprobaSigno(Token.PUNTO_COMA)) {
            System.out.println("Tabla creada correctamente.");
        } else {
            agregarErrorSintactico(token.getLexema(),
token.getTipoToken(), token.getFila(), token.getColumna(), "Falta ')'
o ';' en la declaración de la tabla.");
            System.out.println("Error: Falta ')' o ';' en la declaración de
la tabla.");
        }
    } else {
        agregarErrorSintactico(token.getLexema(),
token.getTipoToken(), token.getFila(), token.getColumna(), " Falta
abrir paréntesis '(' en la declaración de la tabla");
        System.out.println("Error: Falta abrir paréntesis '(' en la
declaración de la tabla.");
    }
}
```

Gramática para CREATE TABLE

G = (N, T, P, S)

N (Sin terminales) :

{ Escritura, NombreTabla, DeclaracionColumnas, Columna, Terminador }

T (Terminales) :

{ CREATE, TABLE, <identificador>, (,), ,, ; }

S (Símbolo inicial) :

{ Escritura }

P (Producciones) :

Escritura → CREATE TABLE NombreTabla DeclaracionColumnas Terminador

NombreTabla → <identificador>

DeclaracionColumnas → (Columna { , Columna })

Columna → <identificador> Tipo

Tipo → INTEGER | VARCHAR | DATE | DECIMAL | TEXT | BOOLEAN | SERIAL

Terminador → ;

Parse para ALTER

```
private void parseAlterTable() {
        avanzarToken();
        if (comprobaReservada(Token.TABLE)) {
            if (comproIdentificador()) {
                if (comprobaReservada(Token.ADD)) {
                    parseAlterAdd();
                } else if (comprobaReservada(Token.DROP)) {
                    parseAlterDrop();
                } else if (comprobaReservada(Token.ALTER)) {
                    parseAlterColumnType();
                } else {
                    agregarErrorSintactico(getTokenActual().getLexema(),
getTokenActual().getTipoToken(), getTokenActual().getFila(),
getTokenActual().getColumna(), "Error: Instrucción ALTER
TABLE no válida.");
                }
            } else {
                agregarErrorSintactico(getTokenActual().getLexema(),
getTokenActual().getTipoToken(), getTokenActual().getFila(),
getTokenActual().getColumna(), "Error: Falta identificador de tabla
en ALTER TABLE.");
            }
        }
```

}

Gramática para ALTER TABLE

G = (N, T, P, S)

N (No terminales):

{ Escritura, NombreTabla, Alteracion, AddColumn, DropColumn, AlterColumnType, Tipo }

T (Terminales):

{ ALTER, TABLE, ADD, DROP, COLUMN, <identificador>, TYPE }

S (Símbolo inicial):

{ Escritura }

P (Producciones):

Escritura → ALTER TABLE NombreTabla Alteracion

NombreTabla → <identificador>

Alteracion → AddColumn | DropColumn | AlterColumnType

AddColumn → ADD COLUMN <identificador> Tipo

DropColumn → DROP COLUMN <identificador>

AlterColumnType → ALTER COLUMN <identificador> TYPE Tipo

Tipo → INTEGER | VARCHAR | DATE | DECIMAL | TEXT | BOOLEAN | SERIAL

Parse ALTER ADD- ALTERCOLUMN- ALTER DROP

```
private void parseAlterAdd() {
        if (comprobaReservada(Token.COLUMN)) {
            if (comproIdentificador()) {
                String tipoDato = parseTipoDeDato();
                if (tipoDato != null) {
                    if (comprobaSigno(Token.PUNTO_COMA)) {
                        System.out.println("Columna añadida
correctamente.");
                    } else {

agregarErrorSintactico(getTokenActual().getLexema(),
getTokenActual().getTipoToken(), getTokenActual().getFila(),
getTokenActual().getColumna(), "Error: Falta ';' al final de la
instrucción.");
                    }
                } else {
                    agregarErrorSintactico(getTokenActual().getLexema(),
getTokenActual().getTipoToken(), getTokenActual().getFila(),
getTokenActual().getColumna(), "Error: Tipo de dato no válido.");
                }
```

```
            } else {
                agregarErrorSintactico(getTokenActual().getLexema(), getTokenActual().getTipoToken(), getTokenActual().getFila(), getTokenActual().getColumna(), "Error: Falta el nombre de la columna en ADD COLUMN.");
            }
        } else if (comprobaReservada(Token.CONSTRAINT)) {
            if (comproIdentificador()) {
                parseConstraintType();
            } else {
                agregarErrorSintactico(getTokenActual().getLexema(), getTokenActual().getTipoToken(), getTokenActual().getFila(), getTokenActual().getColumna(), "Error: Falta el nombre de la constraint en ADD CONSTRAINT.");
            }
        } else {
            agregarErrorSintactico(getTokenActual().getLexema(), getTokenActual().getTipoToken(), getTokenActual().getFila(), getTokenActual().getColumna(), "Error: Se esperaba 'COLUMN' o 'CONSTRAINT' después de 'ADD'.");
        }
    }

    private void parseAlterColumnType() {
        if (comprobaReservada(Token.COLUMN) && comproIdentificador()) {
```

```
            if (comprobaReservada(Token.TYPE)) {
                String tipoDato = parseTipoDeDato();
                if (tipoDato != null &&
comprobaSigno(Token.PUNTO_COMA)) {
                    System.out.println("Tipo de columna alterado
correctamente.");
                } else {
                    agregarErrorSintactico(getTokenActual().getLexema(),
getTokenActual().getTipoToken(), getTokenActual().getFila(),
getTokenActual().getColumna(), "Error: Falta ';' al final de la
instrucción.");
                }
            } else {
                agregarErrorSintactico(getTokenActual().getLexema(),
getTokenActual().getTipoToken(), getTokenActual().getFila(),
getTokenActual().getColumna(), "Error: Se esperaba 'TYPE' en
ALTER COLUMN.");
            }
        } else {
            agregarErrorSintactico(getTokenActual().getLexema(),
getTokenActual().getTipoToken(), getTokenActual().getFila(),
getTokenActual().getColumna(), "Error en ALTER COLUMN: falta
identificador de columna.");
        }
    }
```

```
    private void parseAlterDrop() {
        if (comprobaReservada(Token.COLUMN) &&
comproIdentificador()) {
            if (comprobaSigno(Token.PUNTO_COMA)) {
                System.out.println("Columna eliminada correctamente.");
            } else {
                agregarErrorSintactico(getTokenActual().getLexema(),
getTokenActual().getTipoToken(), getTokenActual().getFila(),
getTokenActual().getColumna(), "Error: Falta ';' al final de la
instrucción.");
            }
        } else if (comprobaReservada(Token.CONSTRAINT) &&
comproIdentificador()) {
            if (comprobaSigno(Token.PUNTO_COMA)) {
                System.out.println("Restricción eliminada
correctamente.");
            } else {
                agregarErrorSintactico(getTokenActual().getLexema(),
getTokenActual().getTipoToken(), getTokenActual().getFila(),
getTokenActual().getColumna(), "Error: Falta ';' al final de la
instrucción.");
            }
        } else {
            agregarErrorSintactico(getTokenActual().getLexema(),
getTokenActual().getTipoToken(), getTokenActual().getFila(),
getTokenActual().getColumna(), "Error en DROP COLUMN o
CONSTRAINT.");
```

}

}

Gramática para ALTER TABLE - Subcomandos ADD, ALTER COLUMN, DROP

G = (N, T, P, S)

N (No terminales):

{ Escritura, NombreTabla, Alteracion, AddColumn, DropColumn, AlterColumnType, Tipo, Constraint }

T (Terminales):

{ ALTER, TABLE, ADD, DROP, COLUMN, CONSTRAINT, <identificador>, TYPE, ; }

S (Símbolo inicial):

{ Escritura }

P (Producciones):

Escritura → ALTER TABLE NombreTabla Alteracion

NombreTabla → <identificador>

Alteracion → AddColumn | DropColumn | AlterColumnType

AddColumn → ADD COLUMN <identificador> Tipo Terminador

AddConstraint → ADD CONSTRAINT <identificador> Constraint Terminador

DropColumn → DROP COLUMN <identificador> Terminador

DropConstraint → DROP CONSTRAINT <identificador> Terminador

AlterColumnType → ALTER COLUMN <identificador> TYPE Tipo Terminador

Tipo → INTEGER | VARCHAR | DATE | DECIMAL | TEXT | BOOLEAN | SERIAL (y otros tipos de datos posibles en SQL)

Constraint → PRIMARY KEY | FOREIGN KEY | UNIQUE | NOT NULL (y otros tipos de restricciones en SQL)

Terminador → ;