

MANUAL DE USUARIO

ANALIZADOR LEXICO Y SINTACTICO

CONSULTAS SQL

PROYECTO FINAL LENGUAJES FORMALES

APLICACIÓN DE ESCRITORIO.

CRISTIAN ALEJANDRO ROLDÁN LÓPEZ

202147280

Versión de Java: Java 21

Requisitos para su ejecución: Consola de Windows

(CMD,

GIT CMD)

CONSOLA DE LINUX

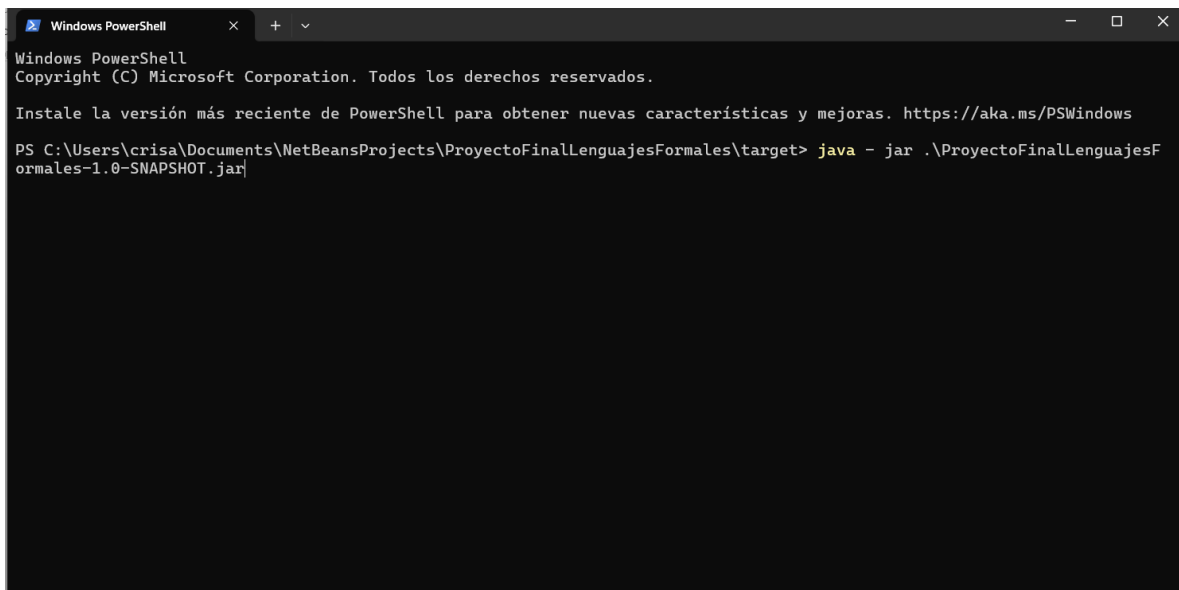
APACHE NETBEANS 21.

Descripción

La empresa LogiCode ha decidido asignarle un nuevo proyecto, dado su desempeño con los analizadores léxicos. El sistema consiste en la creación de un analizador léxico usando la herramienta JFlex, el lenguaje que debe reconocer es el de SQL, pero para esta ocasión también se le solicita que implemente un analizador sintáctico para que reconozca la estructura del lenguaje y una vez que la entrada es analizada, debe extraer la información analizada y generar diagramas de las tablas.

Como Ejecutarlo.

Para ejecutar el analizador léxico tendremos que entrar en una terminal ya sea Cmd o la terminal de Linux.

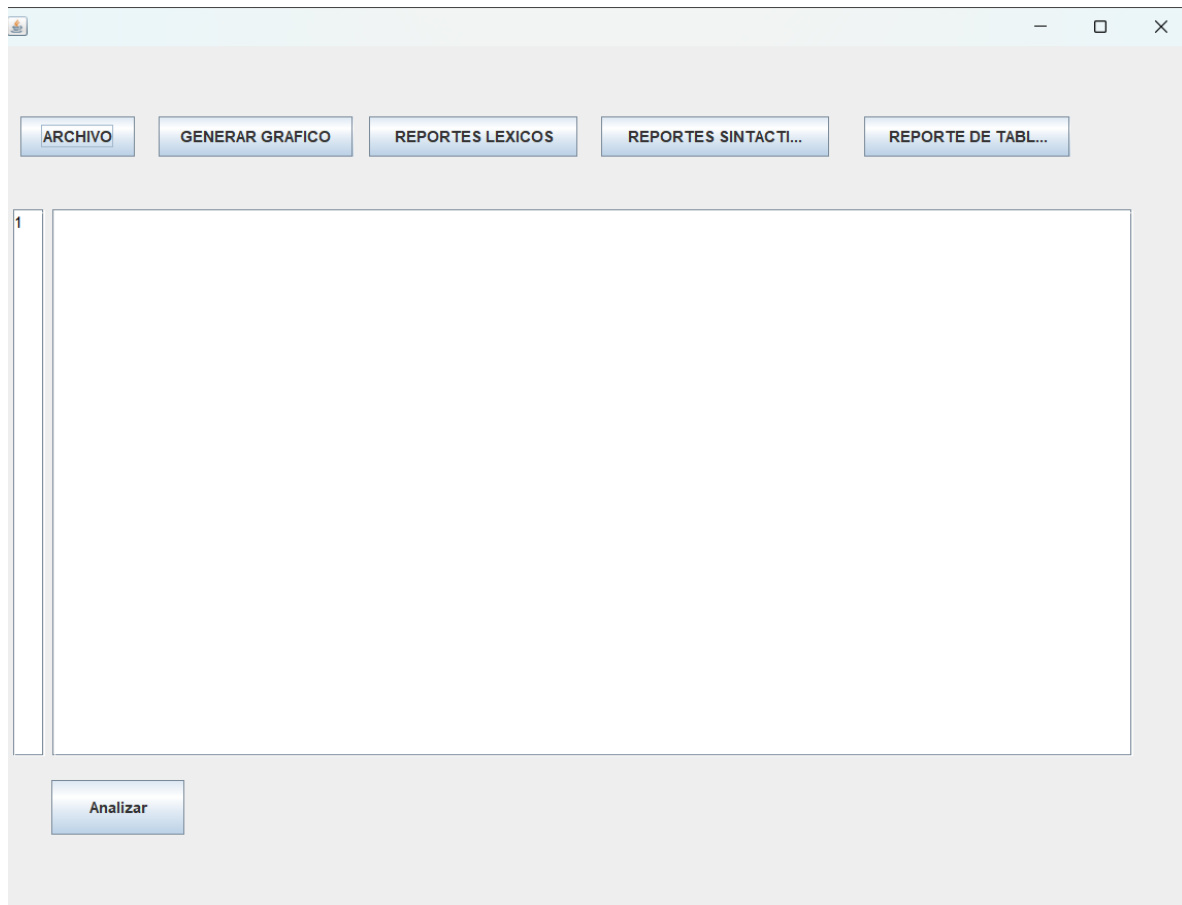
A screenshot of a Windows PowerShell terminal window. The title bar reads "Windows PowerShell" with standard window controls. The terminal text includes the Windows PowerShell header, a copyright notice for Microsoft Corporation, a link to update PowerShell, and a command prompt showing the execution of a Java JAR file. The command is: `PS C:\Users\crisa\Documents\NetBeansProjects\ProyectoFinalLenguajesFormales\target> java -jar .\ProyectoFinalLenguajesFormales-1.0-SNAPSHOT.jar`

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

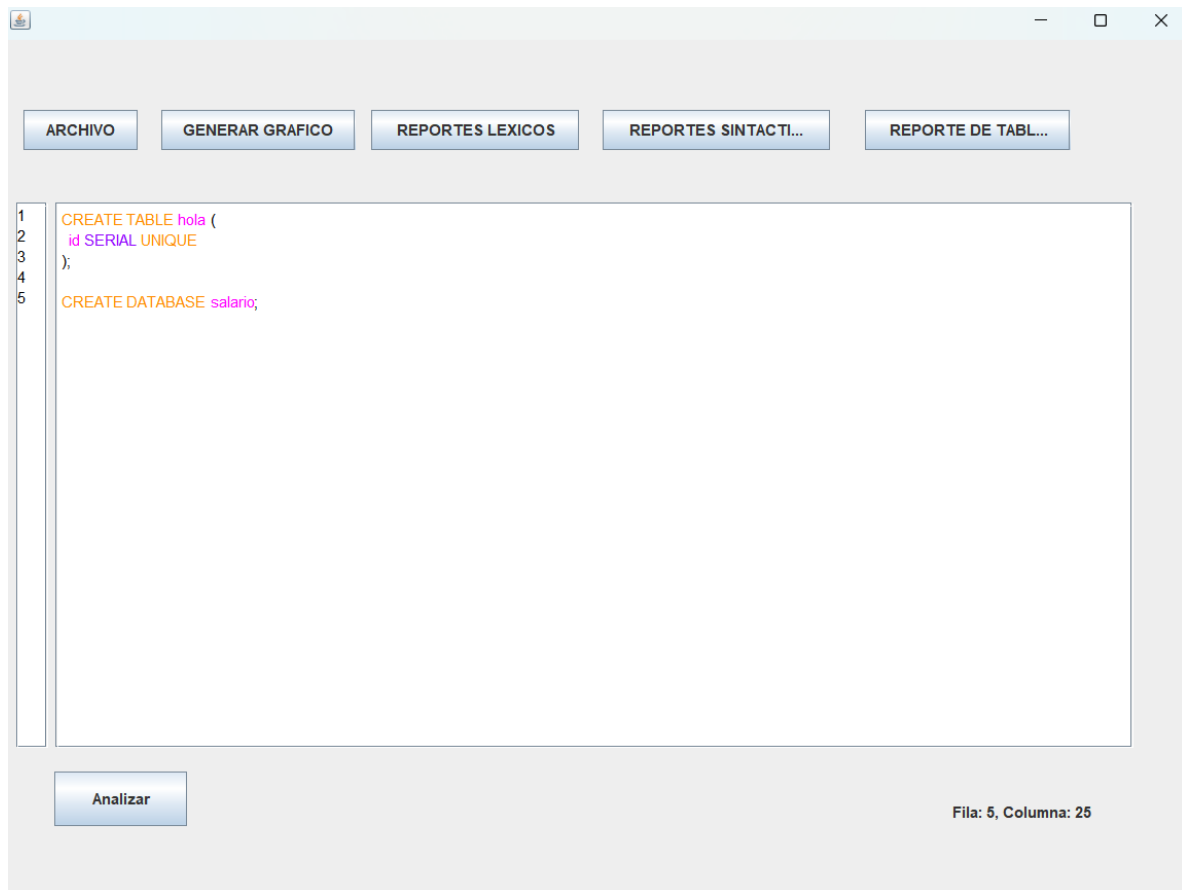
Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. https://aka.ms/PSWindows

PS C:\Users\crisa\Documents\NetBeansProjects\ProyectoFinalLenguajesFormales\target> java -jar .\ProyectoFinalLenguajesFormales-1.0-SNAPSHOT.jar
```

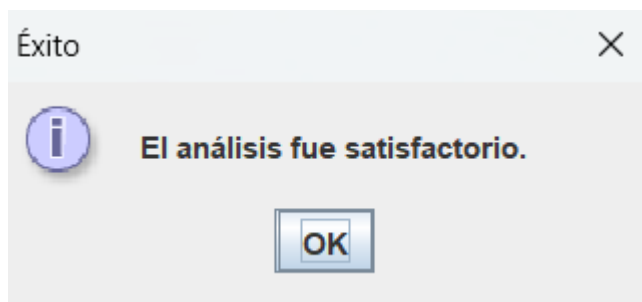
Estando dentro de ella vamos a colocar el comando `java -jar .\ProyectoFinalLenguajesFormales-1.0-SNAPSHOT.jar` que se le entregara en la carpeta del analizador.



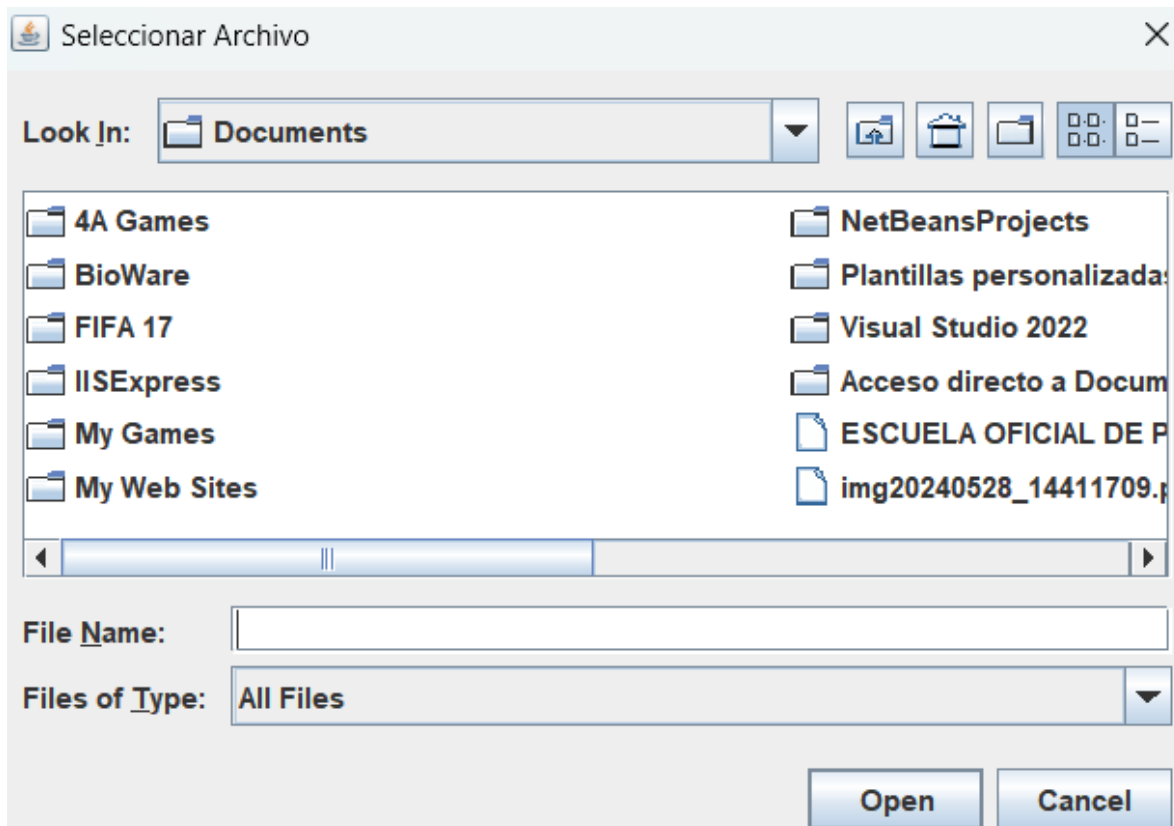
- **Archivo:** Cuando se presiona se abre un apartado para poder cargar un archivo de texto que se desee analizar.
- **Generar Grafico:** Cuando se presiona crea una imagen jpg de las tablas encontradas.
- **Reportes Lexico:** Muestra un reporte de errores Lexicos encontrados durante el análisis.
- **Reporte Sintactico:** Muestra un reporte de errores Sintacticos encontrados durante el análisis.
- **Reporte Tablas:** Muestra un reporte de tablas encontradas y otros reportes como número de operación por sección.
- **Analizar:** Realiza el análisis del archivo o texto que se ingrese.



De esta forma se pueden ir ingresando las palabras que se deseen analizar, cada token valido tendrá un color específico para distinguirse.



Si el análisis es correcto muestra el mensaje de análisis satisfactorio.



Cuando se carga un archivo se abre la siguiente ventana y se busca el archivo de texto que se desea analizar.

Si el análisis tiene errores léxicos se mostrara en el apartado de reportes léxicos.


Reporte de Errores Léxicos			
Token	Línea	Columna	Descripción
TBL	1	1	Token no recon...

Si el texto ingresado cuenta con errores sintácticos se mostrara en el reporte de esta manera y también se mostrara un mensaje que contiene errores sintácticos.

Reporte de Errores Sintácticos			
Token	Línea	Columna	Descripción
soso	1	8	Se esperaba 'DATABASE'.
soso	1	8	Instrucción no reconocida.

DATABASE

Errores Sintácticos

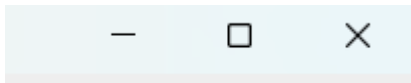
 Se encontraron errores sintácticos:
Error: Instrucción no reconocida. en 'DATABASE' (Línea: 1, Columna: 1)

OK

Analizar

Fila: 1, Columna: 9

Presionando la “X” se cerrara y finalizara la ejecución de nuestro programa.



Tokens validos en el analizador.

Lenguaje SQL

Nombre	Token	Color	Observación
Palabras Reservadas	<ul style="list-style-type: none">• CREATE• DATABASE• TABLE	Naranja	
	<ul style="list-style-type: none">• KEY• NULL• PRIMARY• UNIQUE• FOREIGN• REFERENCES• ALTER• ADD• COLUMN• TYPE• DROP• CONSTRAINT• IF• EXIST• CASCADE• ON• DELETE• SET• UPDATE• INSERT• INTO• VALUES• SELECT• FROM• WHERE• AS• GROUP• ORDER• BY• ASC• DESC• LIMIT• JOIN		

TIPO DE DATO	<ul style="list-style-type: none"> • INTEGER, BIGINT • VARCHAR • DECIMAL • NUMERIC • DATE • TEXT • BOOLEAN • SERIAL 	MORADO	
Entero	Ejemplo: 100, 50, 2, 3, 25	Azul	
Decimal	Ejemplo:	Azul	
	65.50, 52.02, 6000.558		
Fecha	Ejemplo: '2024-10-15' '1999-02-05' '2555-12-22'	Amarillo	La fecha solo puede aceptar el formato: 'YYYY-MM-DD', y debe estar dentro de comillas simples
Cadena	Ejemplo: 'abcd', 'saludo 321' 'camino'	Verde	Una cadena debe estar dentro de comillas simples y puede aceptar cualquier carácter.
identificador	Ejemplos: empleados, departamento_publico categoria_por_rol	Fucsia	Se usa la nomenclatura de snake_case , es decir que solo se aceptan letras minúsculas, guión bajo y números enteros.
Booleano	TRUE, FALSE	Azul	
Funciones de Agregación	<ul style="list-style-type: none"> • SUM • AVG • COUNT • MAX • MIN 	Azul	
Signos	<ul style="list-style-type: none"> • (•) • , • ; • . • = 	Negro	
Aritméticos	+, -, *, /	Negro	
Relacionales	<, >, <=, >=	Negro	
Lógicos	<ul style="list-style-type: none"> • AND • OR • NOT 	Naranja	

Aritméticos	+, -, *, /	Negro	
Relacionales	<, >, <=, >=	Negro	
Lógicos	<ul style="list-style-type: none"> • AND • OR • NOT 	Naranja	
Comentario de línea	-- Este es un comentario -- Una Línea	Gris	Los comentarios empiezan por dos guiones medio, ambos separados por un espacio, luego puede venir cualquier carácter. Y son de una línea los comentarios.

Estructuras correctas en SQL.

Para los DDL

Creación de Base de Datos

Estructura

```
CREATE DATABASE <identificador> ;
```

Ejemplo

```
CREATE DATABASE store;
CREATE DATABASE shop_and_store;
```

Creación de Tablas

La Estructura_de_llaves es opcional puede o no venir, y puede tener múltiples Estructura_de_declaracion separadas por coma.

Estructura

```
CREATE TABLE <identificador> (
[Estructura_de_declaracion] ,
[Estructura_de_llaves]
);
```

Estructura de declaración

Pueden tener múltiples estructuras de declaración separadas por comas.

Estructura

```
<identificador> [Tipo_de_dato] PRIMARY KEY
<identificador> [Tipo_de_dato] NOT NULL
<identificador> [Tipo_de_dato] UNIQUE
```

Tipo de Dato

Puede ser alguno de los siguientes

```
SERIAL | INTEGER | BIGINT | VARCHAR(<entero>) | DECIMAL(<entero>, <entero>) | NUMERIC(<entero>, <entero>) | DATE | TEXT | BOOLEAN
```

Estructura De Llaves

Estructura

```
CONSTRAINT <identificador>
    FOREIGN KEY (<identificador>)
    REFERENCE <identificador>(<identificador>)
```

Ejemplo

```
CREATE TABLE empleados (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    puesto VARCHAR(50),
    salario DECIMAL(10, 2),
    fecha_contratacion DATE,
    departamento_id INTEGER,
    email VARCHAR(100) UNIQUE,
    CONSTRAINT fk_departamento
        FOREIGN KEY (departamento_id)
        REFERENCES departamentos(id)
);
```

Modificadores

Estructura

```
ALTER TABLE <identificador> ADD COLUMN <identificador> [Tipo_de_dato];

ALTER TABLE <identificador> ALTER COLUMN <identificador> TYPE
[Tipo_de_dato];

ALTER TABLE <identificador> DROP COLUMN <identificador>;

ALTER TABLE <identificador> ADD CONSTRAINT <identificador>
[Tipo_de_dato];

ALTER TABLE <identificador> ADD CONSTRAINT <identificador> UNIQUE
(<identificador>);

ALTER TABLE <identificador> ADD CONSTRAINT <identificador> FOREIGN KEY
(<identificador>) REFERENCES <identificador>(<identificador>);

DROP TABLE IF EXISTS <identificador> CASCADE;
```

Ejemplo

```
ALTER TABLE empleados ADD COLUMN fecha_nacimiento DATE;

ALTER TABLE empleados ALTER COLUMN salario TYPE NUMERIC(12, 2);

ALTER TABLE empleados DROP COLUMN puesto;

ALTER TABLE empleados ADD CONSTRAINT uc_email UNIQUE (email);

DROP TABLE IF EXISTS empleados CASCADE;

ALTER TABLE empleados
ADD CONSTRAINT fk_departamento
```

Para los DML

Inserción

Estructura

```
INSERT INTO <identificador> (<identificador>,...) VALUES ([DATO],...);

INSERT INTO <identificador> (<identificador>,...) VALUES ([DATO],...),
...([DATO],...);
```

Dato

Para el datos puede ser alguno de los datos primitivos o alguna relación de expresión aritmética, racional o lógico, y puede aceptar la agrupación de instrucciones entre paréntesis.

<entero> | <decimal> | <fecha> | <cadena> | TRUE | FALSE

[DATO] + [DATO] * [DATO] / [DATO]

[DATO] < [DATO] OR ([DATO] + [DATO]) < [DATO]

Actualización

[WHERE] estructura opcional.

Estructura

```
UPDATE <identificador> SET <identificador> = [DATO] [WHERE];

UPDATE <identificador> SET <identificador> = [DATO], <identificador> =
[DATO],... [WHERE];
```

Ejemplo

```
UPDATE empleados
SET salario = 4000.00
WHERE id = 1;

UPDATE empleados
SET puesto = 'Supervisor', salario = salario * 1.10
WHERE departamento_id = 2;

UPDATE empleados
SET salario = salario + 500
WHERE salario < 3000 AND departamento_id = 1;
```

Eliminación

[WHERE] estructura opcional.

Estructura

```
DELETE FROM <identificador> [WHERE];
```

