Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Organización de Lenguajes y Compiladores 1
Segundo Semestre 2023
Catedráticos:
Ing. Mario Bautista
Ing. Manuel Castillo
Ing. Kevin Lajpop
Tutores académicos:
Walter Guerra
Kevin López
Maynor Piló
Jose Perez

QueryCrypter
Proyecto 2

1. Objetivos

Objetivo General:

Aplicar los conocimientos sobre análisis léxico y sintáctico de un compilador para desarrollar un intérprete sencillo funcional para el lenguaje especificado.

Objetivos Específicos:

- Que el estudiante aprenda a generar un analizadores léxicos y sintácticos utilizando la herramienta JISON.
- Qué el estudiante comprenda el manejo de errores léxicos y sintácticos correctamente durante la interpretación.
- Que el estudiante realicé la implementación del patrón de diseño intérprete para el manejo de acciones gramaticales en javascript.

2. Descripción General

El curso de Organización de Lenguajes y Compiladores 1, perteneciente a la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, necesita un intérprete capaz de reconocer un lenguaje basado en SQL. Por lo que le solicitan a los estudiantes implementar el intérprete en el lenguaje de programación javascript utilizando la herramienta JISON para generar el analizador léxico y sintáctico para el reconocimiento de las instrucciones.

El intérprete debe ser capaz de reconocer instrucciones en lenguaje SQL y las sentencias básicas de DDL y DML para el manejo de tablas en memoria. También se debe permitir el uso de distintos tipos de datos en las expresiones y el manejo correcto de la precedencia de las operaciones de dichas expresiones. Debido a que

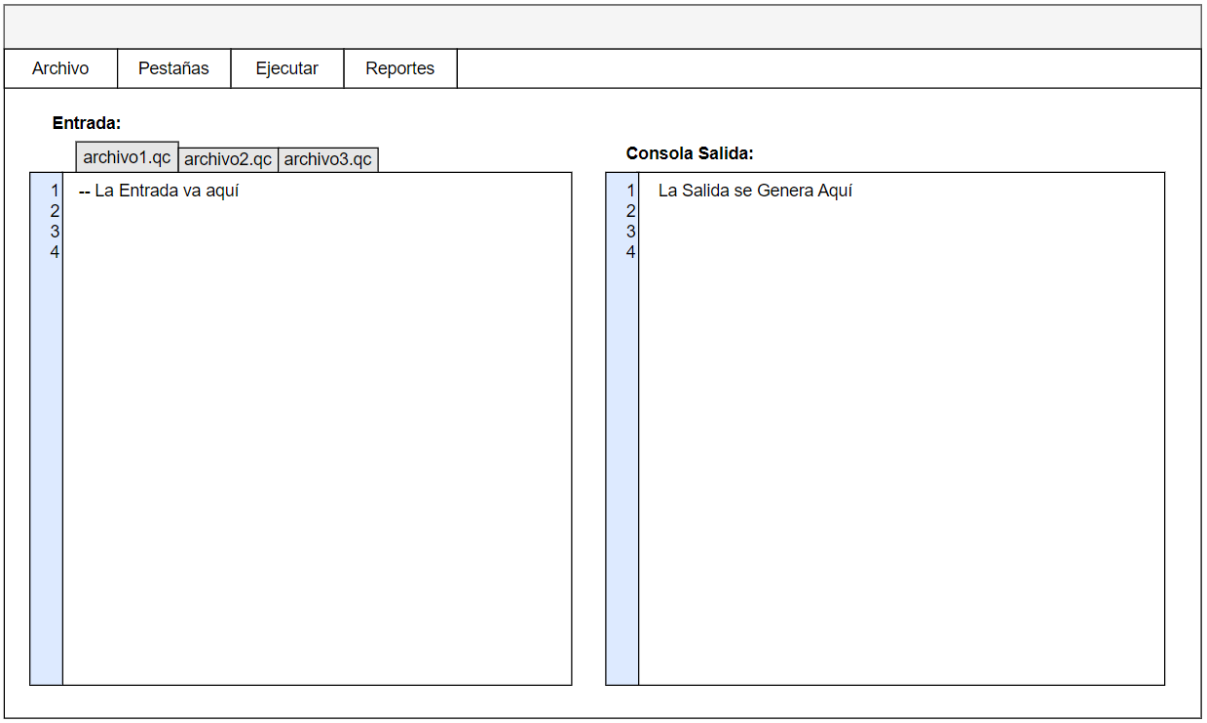
no es requerido que se diferencie entre mayúsculas y minúsculas el intérprete debe ser implementado como case insensitive.

Como cualquier procesador de lenguaje debe generar los reportes correspondientes a la fase de análisis y la interpretación. Los reportes son los siguientes: reporte de tokens, reporte de errores, reporte de tabla de símbolos y la graficación del árbol de análisis sintáctico.

3. Entorno de Trabajo

3.1 Editor:

El editor será parte del entorno de trabajo, cuya finalidad será proporcionar ciertas funcionalidades, características y herramientas que serán de utilidad para el usuario. La función principal del editor será el ingreso del código fuente que será analizado. En este se podrán abrir diferentes archivos al mismo tiempo y deberá mostrar la línea actual. El editor de texto se tendrá que mostrar en el navegador. Queda a discreción del estudiante el diseño.



3.2 Funcionalidades:

- **Nuevo Archivo:** El editor debe tener la capacidad de crear archivos en blanco el cual podrá ser editado en una pestaña que tiene el nombre del archivo.

- **Abrir Archivo:** El editor debe tener la capacidad de abrir archivos con las extensiones .qc cuyo contenido se mostrará en el área de entrada en una nueva pestaña con el nombre del archivo.
- **Guardar:** El editor debe tener la capacidad de guardar el estado del archivo en el que se estará trabajando.
- **Eliminar Pestaña:** Cada pestaña puede ser cerrada en cualquier momento. Si los cambios no se han guardado se descartan.

3.3 Características:

- **Múltiples Pestañas:** Se podrán crear nuevas pestañas con la finalidad de ver y abrir los archivos de prueba en la aplicación. Para cada pestaña corresponde un archivo.

3.4 Herramientas:

- **Ejecutar:** Se envía la entrada de la pestaña actualmente seleccionada al intérprete con la finalidad de realizar el análisis léxico, sintáctico y la ejecución de las instrucciones.

3.5 Reportes:

- **Reporte de Tokens:** Se mostrarán todos los tokens reconocidos por el analizador léxico..
- **Reporte de Errores:** Se mostrarán todos los errores léxicos y sintácticos encontrados.
- **Reporte de Tabla de Símbolos:** Se mostrarán todas las variables almacenadas en la tabla de símbolos.
- **Generar Árbol de Análisis Sintáctico:** Se deberá generar una imagen del árbol de análisis sintáctico resultante del análisis.

3.6 Area de Consola:

En la consola de salida se mostrarán los resultados, mensajes y todo lo que sea indiciado en el lenguaje. Tiene como restricción el no ser editable por el usuario y únicamente puede mostrar información.

4. Descripción del lenguaje

El lenguaje Query Crypter basado en SQL, o Structured Query Language, es un lenguaje de consulta de bases de datos relacionales. Se utiliza para realizar tareas como la creación, modificación y consulta de datos en una base de datos.

El lenguaje Query Crypter se divide en dos categorías principales:

- **Lenguaje de definición de datos (DDL):** Se utiliza para crear, modificar y eliminar tablas de una base de datos.
- **Lenguaje de manipulación de datos (DML):** Se utiliza para insertar, actualizar, eliminar y seleccionar datos de las tablas de una base de datos.

4.1 Case insensitive

El lenguaje QueryCrypter es case insensitive, por lo que las siguientes sentencias son equivalentes:

```
SELECT * FROM users WHERE name = 'John Doe';

SELECT * FROM USERS WHERE NAME = 'John Doe';
```

En ambos casos, el intérprete devolverá todas las filas de la tabla users donde el valor de la columna name sea John Doe

4.2 Comentarios

El lenguaje QueryCrypter admite dos tipos de comentarios: comentarios de una línea y comentarios de varias líneas.

4.2.1 Comentarios de una línea

Los comentarios de una línea comienzan con el símbolo **--** y terminan con el final de la línea. Por ejemplo, el siguiente código es un ejemplo de un comentario de una línea:

```
-- Este es un comentario de una línea
```

4.2.1 Comentarios de varias líneas

Los comentarios de varias líneas comienzan con los símbolos **/*** y se extienden hasta los símbolos ***/**. Por ejemplo, el siguiente código es un ejemplo de un comentario de varias líneas

```
/*
  Este es un comentario
  de varias líneas
*/
```

4.3 Tipos de datos

El lenguaje QueryCrypter admite una variedad de tipos de datos, que se pueden utilizar para almacenar diferentes tipos de datos.

Tipo	Definición	Descripción	Ejemplo	observaciones
Enteros	int	Los tipos de datos enteros se utilizan para almacenar números enteros.	123, 568, 10, etc	Se maneja cualquier cantidad de dígitos.
Reales	double	Los tipos de datos reales se utilizan para almacenar números reales.	1.2, 50.23, 00.34, etc.	Se maneja cualquier cantidad de decimales
Date	date	Los tipos de datos de fecha se utilizan para almacenar valores de fecha.	2023-07-20	El formato de la fecha es YYYY-MM-DD
Cadena de caracteres	varchar	Los tipos de datos de cadena de caracteres se utilizan para almacenar cadenas de caracteres	"Hola mundo"	Cadena de caracteres de longitud variable,Se permitirá cualquier carácter entre las comillas dobles,incluyendo las secuencias de escape: \" comilla doble \\ barra invertida \n salto de linea \r retorno de carro \t tabulación
BOOLEAN	TRUE / FALSE.	El tipo de datos BOOLEAN se utiliza para almacenar valores	TRUE, FALSE	

		lógicos.		
NULL	NULL	El tipo de datos NULL se utiliza para representar un valor desconocido o no aplicable.	NULL	

4.4 Secuencias de Escape

El lenguaje QueryCrypter admite secuencias de escape para evitar que los caracteres especiales se interpreten incorrectamente. Las secuencias de escape se utilizan para definir ciertos caracteres especiales dentro de cadenas de texto.

Secuencia	Descripción	Ejemplo
\n	Salto de línea	“Hola\nMundo”
****	Barra invertida	“C:\\miCarpeta\\Personal”
\“	Comilla doble	“\”Esto es una cadena\””
\t	Tabulación	“\tEsto es una tabulación”
\’	Comilla Simple	“\’Estas son comillas simples\’”

4.5 Operadores Aritméticos

4.5.1 Suma

Es la operación aritmética que consiste en realizar la suma entre dos o más valores. El símbolo a utilizar es el signo más **+**.

Especificaciones de la operación suma

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

+	Int	Double	Date	Varchar	Boolean	Null
Int	int	Double	Date	Int		
Double	Double	Double	Date	Double		
Date	Date	Date		Date		
Varchar	Int	Double	Date	Varchar		

Boolean						
Null						

Nota:

- Se puede utilizar el operador de suma para sumar días a valores de fecha

- Al momento de sumar tipos de datos varchar e Int, el dato de tipo varchar deberá de ser válido(numéricos)

4.5.2 Resta

Es la operación aritmética que consiste en realizar la resta entre dos o más valores. El símbolo por utilizar es el signo menos **-**

Especificaciones de la operación resta

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

-	**Int**	**Double**	**Date**	**Varchar**	**Boolean**	**Null**
Int	int	Double	Date	Int		
Double	Double	Double	Date	Double		
Date	Date	Date		Date		
Varchar	Int	Double	Date	Varchar		
Boolean						
Null						

Nota:

- Se puede utilizar el operador de resta para restar días a valores de fecha

- Al momento de restar tipos de datos varchar e Int, el dato de tipo varchar deberá de ser válido(numéricos)

4.5.3 Multiplicación

Operación aritmética que consiste en sumar un número (multiplicando) tantas veces como indica otro número (multiplicador). El sino para representar la operación es el asterisco *****.

Especificaciones de la operación multiplicación

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

*	Int	Double	Date	Varchar	Boolean	Null
Int	int	Double				
Double	Double	Double				
Date						
Varchar						
Boolean						
Null						

4.5.4 División

Operación aritmética que consiste en partir un todo en varias partes, al todo se le conoce como dividendo, al total de partes se le llama divisor y el resultado recibe el nombre de cociente. El operador de la división es la diagonal **/**.

Especificaciones de la operación División

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

/	Int	Double	Date	Varchar	Boolean	Null
Int	int	Double				
Double	Double	Double				

Date						
Varchar						
Boolean						
Null						

4.5.5 Módulo

Es una operación aritmética que obtiene el resto de la división de un número entre otro. El signo a utilizar es el porcentaje %..

Especificaciones de la operación Módulo

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

%	Int	Double	Date	Varchar	Boolean	Null
Int	int	Double				
Double	Double	Double				
Date						
Varchar						
Boolean						
Null						

4.5.6 Negación Unaria

Es una operación que niega el valor de un número, es decir que devuelve el contrario del valor original.

Especificaciones de la operación negación

A continuación, se especifica en una tabla los resultados que se deberán obtener con esta operación.

-num	Int
Int	int
Double	Double
Date	

Varchar	
Boolean	
Null	

Nota: Todas las operaciones matemáticas retornan "null" en caso de error.

4.6 Operadores Relacionales

Los operadores relacionales en SQL son símbolos o palabras clave utilizados para comparar valores en una consulta o condición, y así determinar si se cumple o no una relación o comparación entre esos valores.

Estos operadores se utilizan principalmente en la cláusula WHERE de una consulta SELECT para filtrar los resultados y obtener registros que cumplan con ciertas condiciones específicas.

A continuación, se definen los símbolos que serán aceptados dentro del lenguaje:

Secuencia	Descripción	Ejemplo
=	Igualación: Compara ambos valores y verifica si son iguales.	SELECT * FROM productos WHERE precio = 100;
!=	Diferenciación: Compara ambos lados y verifica si son distintos.	SELECT * FROM productos WHERE precio != 100;
<	Menor que: Compara ambos lados y verifica si el derecho es mayor que el izquierdo.	SELECT * FROM productos WHERE precio < 100;
<=	Menor o igual que: Compara ambos lados y verifica si el derecho es mayor o igual que el izquierdo.	SELECT * FROM productos WHERE precio <= 100;
>	Mayor que: Compara ambos lados y verifica si el izquierdo es mayor que el derecho.	SELECT * FROM productos WHERE precio > 100;
>=	Mayor o igual que: Compara ambos lados y verifica si el	SELECT * FROM productos WHERE precio >= 100;

	izquierdo es mayor o igual que el derecho.	

4.7 Operadores Lógicos

Los operadores lógicos sirven para combinar dos valores booleanos y dar un retorno del resultado, sea verdadero o falso.

A continuación, se definen los símbolos que serán aceptados dentro del lenguaje:

Secuencia	Descripción	Ejemplo
AND	Es el “y” lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.	SELECT * FROM productos WHERE precio > 100 AND nombre = 'Producto 1';
OR	Es el “o” lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.	SELECT * FROM productos WHERE precio > 100 OR nombre = 'Producto 1';
NOT	Es el “o” lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta..	SELECT * FROM productos WHERE NOT nombre = 'Producto 1';

4.8 Signos de Agrupación

Los signos de agrupación en SQL se utilizan para definir el orden en el que se evalúan las expresiones. Los signos de agrupación serán utilizados para agrupar operaciones aritméticas, lógicas o relacionales. Los símbolos de agrupación están dados por (y).

Los signos de agrupación también se pueden utilizar para agrupar subconsultas.

```
SELECT * FROM productos WHERE precio > (
  SELECT AVG(precio) FROM productos
);
```

4.9 Signos de Agrupación

La precedencia de operadores nos indica la importancia en que una operación debe realizarse por encima del resto. A continuación, se define la misma.

Nivel	Operador	Asociatividad
0	-	Derecha
1	/, *	Izquierda
2	+, -	Izquierda
3	==, !=, <, <=, >, >=	Izquierda
4	NOT	Derecha
5	AND	Izquierda
6	OR	Izquierda

NOTA: el nivel 0 es el nivel de mayor importancia.

4.10 Caracteres de finalización y encapsulamiento de sentencias

El lenguaje se verá restringido por dos reglas que ayudan a finalizar una instrucción y encapsular sentencias:

- **Finalización de instrucciones:** para finalizar una instrucciones se utilizará el signo **;**.

```
<SENTENCIA>;

SELECT * FROM productos;
```

- **Encapsular sentencias:**Las sentencias QueryCrypter también se pueden encapsular en bloques. Esto se hace utilizando la palabra clave BEGIN y la palabra clave END. Los bloques de sentencias SQL se utilizan para agrupar sentencias SQL relacionadas.

```
BEGIN
  <SENTENCIAS>;
END;

BEGIN
  SELECT * FROM productos WHERE precio > 100;
END;
```

4.11 Declaración y asignación de variables

Las variables nos permiten almacenar un valor y recuperarlo más adelante para emplear en otras sentencias. Pueden utilizarse para almacenar valores temporales, para realizar cálculos complejos o para simplificar la escritura de consultas.

Nota:

- Una variable debe ser declarada antes de usarse
- Para declarar una variable en SQL, se utiliza la sentencia DECLARE
- Las variables comienzan con "@" (arroba) seguido del nombre (sin espacios), dicho nombre puede contener cualquier carácter.
- Puede declarar varias variables en una misma sentencia
- No existen variables globales
- Una variable declarada existe dentro del entorno en que se declara; debemos declarar y emplear la variable en el mismo lote de sentencias, porque si declaramos una variable y luego, en otro bloque de sentencias pretendemos emplearla, dicha variable ya no existe.
- Una variable a la cual no se le ha asignado un valor contiene "null", o cuando se le asigna un valor predeterminado se utiliza la sentencia DEFAULT antes del valor.

```
DECLARE @variable_name [DATA_TYPE] [DEFAULT value];

-- Declaración de una variable
declare @nombre varchar;

-- Declaración de varias variables
 declare @nombre varchar, @edad int
```

Una vez declarada una variable, se puede asignar un valor a la variable utilizando la sentencia SET. El valor que se asigna a una variable debe ser del mismo tipo de datos que la variable.

```
SET @variable_name = value;

-- Declara una variable de tipo `VARCHAR`
DECLARE @nombre VARCHAR;

-- Declara una variable de tipo `INT` con un valor predeterminado de 10
DECLARE @edad INT DEFAULT 10;

-- Asigna el valor 'Juan Pérez' a la variable @nombre
SET @nombre = 'Juan Pérez';

-- Asigna el valor 25 a la variable @edad
SET @edad = 25;

-- Imprime el valor de la variable @nombre
SELECT @nombre;

-- Imprime el valor de la variable @edad
SELECT @edad;
```

4.9 Ejecución DDL(Data Definition Language):

4.9.1 Create Table

Dado que QueryCrypter es un lenguaje basado en SQL el proyecto deberá de ser capaz de crear estructuras de datos simulando tablas de n dimensiones para almacenar datos ejemplo de la estructura para crear una tabla de datos

```
CREATE TABLE nombre_de_la_tabla
(
    columna1 tipo_de_dato,
    columna2 tipo_de_dato2,
    ...
    columnaN tipo_de_datoN
);
```

Ejemplo

```
CREATE TABLE Clientes
(
    ID_Cliente INT,
    Nombre VARCHAR,
    CorreoElectronico VARCHAR
);
```

4.9.2 Alter Table

Con Alter table QueyCrypter debe ser capaz de modificar la estructura de una tabla existente, como agregar, eliminar o modificar el nombre de las columnas y cambiar el nombre de la tabla. Ejemplo de estructura

```
ALTER TABLE nombre_de_la_tabla

<ACCION> ;
```

Donde las acciones que pueden ser utilizadas son las siguientes:

Agregar una nueva columna:

```
ALTER TABLE nombre_de_la_tabla

ADD nombre_de_la_columna tipo_de_dato;
```

Eliminar una columna:

```
ALTER TABLE nombre_de_la_tabla

DROP COLUMN nombre_de_la_columna;
```

Cambiar el nombre de la tabla:

```
ALTER TABLE nombre_de_la_tabla

RENAME TO nuevo_nombre_de_la_tabla;
```

Cambiar el nombre de la columna:

```
ALTER TABLE nombre_de_la_tabla

RENAME COLUMN nombre_anterior_de_la_columna TO
nuevo_nombre_de_la_columna;
```

4.9.3 Drop Table

Con drop table QueyCrypter debe ser capaz de eliminar una tabla y todos sus datos de la estructura de datos

```
DROP TABLE nombre_de_la_tabla;
```

4.10 Ejecución DML(Data Manipulation Language):

4.10.1 insert

Con insert QueyCrypter debe ser capaz de agregar nuevos registros (filas) a una tabla existente. Puedes especificar los valores que se deben insertar en cada columna. Ejemplo de la estructura:

```
INSERT INTO nombre_de_la_tabla (columna1, columna2,
..., columnaN)
VALUES (valor1, valor2, ..., valorN);
```

```
INSERT INTO Clientes (Nombre, CorreoElectronico)

VALUES ('Juan Pérez', 'juan@example.com');
```

4.10.2 select

Con select deberá recuperar datos de alguna tabla, también será utilizado para obtener valores de alguna variable a su vez podrá filtrar datos basándose en la cláusula while queda a criterio y creatividad de cada uno como presentar el resultado en consola, a continuación se presenta las diferentes estructuras del select:

Estructura simple:

```
SELECT columna1, columna2, …

FROM nombre_de_la_tabla;
```

Estructura corta para traer todas las columnas:

```
SELECT * FROM nombre_de_la_tabla;
```

Estructura con clausula where:

```
SELECT columna1, columna2, ...
FROM nombre_de_la_tabla
WHERE condición;
```

Donde ‘condición’ puede ser una expresión lógica que involucre operadores como =, !=, <, >, <=, >=, AND, OR, NOT.
Nota: solo se podrá comparar la columna con un valor constante o con una variable.

Estructura para imprimir variables:

```
DECLARE @nombre_variable tipo_de_dato;  -- Declaración de la variable
SET @nombre_variable = valor;  -- Asignación de valor a la variable

-- Utilizando SELECT para mostrar el valor de la variable
SELECT @nombre_variable AS nombre_de_columna_resultante;
```

Ejemplos de Select:

```
SELECT Nombre, Edad
```

```
FROM Empleados
WHERE Departamento = 'Ventas';

DECLARE @precio1 INT;
SET @precio1 = 50.00;

SELECT Nombre, Precio
FROM Productos
WHERE Precio = @precio1;

Select @precio1 AS valor_precio;
```

4.10.3 update

Se utiliza para modificar datos existentes en una tabla. Se podrá actualizar los valores de una o varias columnas en función de una condición específica. Ejemplo de estructura

```
UPDATE nombre_de_la_tabla
SET columna1 = nuevo_valor1, columna2 = nuevo_valor2,
...
WHERE condición;
```

```
UPDATE Empleados
SET Salario = 55000
WHERE Departamento = 'Ventas';
```

4.10.4 truncate

se utiliza para eliminar todos los registros de una tabla, pero no elimina la estructura de la tabla en sí. Ejemplo de estructura

```
TRUNCATE TABLE nombre_de_la_tabla;
```

4.10.5 delete

Se utiliza para eliminar registros de una tabla que cumplan con ciertas condiciones. Ejemplo de la estructura:

```
DELETE FROM nombre_de_la_tabla

WHERE condición;
```

Ejemplo:

```
DELETE FROM Clientes

WHERE Estado = 'Inactivo';
```

4.11 Casteos

En QueryCrypter la operación de casteo se utiliza para convertir un tipo de dato en otro.

```
CAST (<EXPRESIÓN> AS tipo_de_dato)
```

Ejemplos:

```
SELECT CAST(salario AS VARCHAR) FROM Empleados;

SELECT nombre, CAST(precio AS DOUBLE) FROM Productos;
```

4.12 Sentencias de Control

Las sentencias de control en QueryCrypter se utilizan para controlar el flujo de ejecución de las consultas o instrucciones SQL, y permiten tomar decisiones basadas en condiciones.

4.12.1 If

La sentencia if permite ejecutar una acción si se cumple una condición y otra acción si no se cumple.

```
IF <EXPRESIÓN> THEN
BEGIN
        -- Acción si se cumple la condición
        <SENTENCIAS>
END;
```

```
IF <EXPRESIÓN> THEN
        -- Acción si se cumple la condición
        <SENTENCIAS>
ELSE
        -- Acción si no se cumple la condición
        <SENTENCIAS>
END IF;
```

Ejemplo:

```
DECLARE @nota INT;
SET @nota = 70;

IF @nota >= 61 THEN
        PRINT 'Ganó el laboratorio';
ELSE
        PRINT 'Perdió el laboratorio';
END IF;
```

4.12.2 Case

La sentencia case se utiliza para realizar múltiples acciones basadas en una condición. Puede ser utilizado de dos formas: CASE simple y CASE buscado.

CASE Simple

El CASE simple se utiliza cuando quieres comparar una expresión con varios valores y retornar un resultado específico cuando se cumple una de las condiciones.

```
CASE valor
        WHEN valor1 THEN resultado1
        WHEN valor2 THEN resultado2
        …
        ELSE resultado_predeterminado
```

```
END;
```

Ejemplo:

```
DECLARE @nota INT;
SET @nota = 70;

CASE nota
        WHEN 100 THEN 'Sobresaliente'
        WHEN 99 THEN 'Muy bueno'
        WHEN 98 THEN 'Bueno'
        ELSE 'Aprobado'
END AS resultado;
```

CASE Buscado

El CASE buscado se utiliza cuando deseas realizar evaluaciones condicionales basadas en múltiples condiciones independientes.

```
CASE
        WHEN condición1 THEN resultado1
        WHEN condición2 THEN resultado2
        ...
        ELSE resultado_predeterminado
END;
```

Ejemplos:

```
-- Ejemplo 1
DECLARE @nota INT;
SET @nota = 70;

CASE
        WHEN nota > 85 THEN 'Excelente'
        WHEN nota >= 61 AND nota <= 85 THEN 'Aprobado'
        ELSE 'No Aprobado'
END;
```

4.13 Sentencias Cíclicas

Las sentencias ciclicas en QueryCrypter se utilizan para realizar bucles y repetir un conjunto de instrucciones hasta que se cumpla una condición específica.

4.12.1 While

El bucle While se utiliza para ejecutar un bloque de código mientras una condición sea verdadera. La condición se verifica antes de cada ejecución del bloque, y si es verdadera, el bloque se ejecuta; de lo contrario, el bucle se detiene.

```
WHILE <EXPRESIÓN>
BEGIN
        -- Código a ejecutar mientras se cumple la condición
        <SENTENCIAS>
END;
```

Ejemplo:

```
DECLARE @contador INT = 0;

WHILE @contador < 10
BEGIN
        PRINT 'Contador: ' + CAST(@contador AS VARCHAR);
        SET @contador = @contador + 1;
END;
```

4.12.1 For

El bucle FOR se utiliza para iterar a través de un conjunto predefinido de valores o elementos, como una secuencia de números. A diferencia de WHILE, FOR tiene un número fijo de iteraciones basadas en el conjunto de valores especificados.

```
FOR <ID> IN <RANGO>
BEGIN
        -- Código a ejecutar
        <SENTENCIAS>
END;
```

Ejemplo:

```
FOR contador IN 1..10
BEGIN
        PRINT 'Contador: ' + CAST(contador AS VARCHAR);
END LOOP;
```

Nota:

- <ID> es la variable que se utiliza para llevar un registro del valor actual en cada iteración.
- El <RANGO> especifica el rango de valores numéricos sobre el cual el bucle itera, y está compuesto por “inicio…fin” donde "inicio" es el valor inicial del rango y "fin" es el valor final del rango .

4.14 Sentencias de transferencia

Las sentencias de transferencia nos permiten manipular el comportamiento de los bucles, ya sea para detenerlo o para saltarse algunas iteraciones. El lenguaje soporta las siguientes sentencias.

4.14.1 Break

La sentencia break hace que se salga del ciclo inmediatamente, es decir que el código que se encuentre después del break en la misma iteración no se ejecutará y este se saldrá del ciclo, se define por la Palabra reservada BREAK seguido de un ;.

4.14.2 Continue

La sentencia continue puede detener la ejecución de la iteración actual y saltar a la siguiente. La sentencia continue siempre debe de estar dentro de un ciclo, de lo contrario será un error, se define por la Palabra reservada CONTINUE seguido de un ;.

Ejemplo:

```
DECLARE @contador INT = 1;

WHILE @contador <= 10
BEGIN
        -- BREAK: Salir del bucle cuando se cumple una condición
        IF @contador = 5
        BEGIN
        BREAK;
        END;

        -- CONTINUE: Saltar una iteración cuando se cumple una condición
        IF @contador = 3
        BEGIN
```

```
        SET @contador = @contador + 1;
        CONTINUE;
        END;

        PRINT 'Contador: ' + CAST(@contador AS VARCHAR);
        SET @contador = @contador + 1;
END;
```

4.15 Funciones

- Una función en QueryCrypter es un objeto de SQL que acepta uno o más valores de entrada, realiza algún procesamiento y devuelve un valor como resultado.
- Puedes utilizar una función en una consulta QueryCrypter para calcular valores, transformar datos o realizar operaciones específicas.
- Las funciones se utilizan en expresiones SQL y se pueden llamar desde consultas SELECT, WHERE, y otras partes de una consulta SQL.
- Los parámetros en las funciones son valores de entrada que la función utilizará en sus operaciones internas para calcular un resultado, en caso de que sean varios parámetros se debe utilizar comas para separarlos.

```
CREATE FUNCTION nombre_funcion
(
        parametro1 tipo_de_dato1,
        parametro2 tipo_de_dato2,
        ...
)
RETURNS tipo_de_dato_retorno
BEGIN
   -- Código de la función aquí
   -- Puede incluir declaraciones, cálculos y lógica
   RETURN valor_retorno;
END;
```

Ejemplo:

```
-- Función que devuelve el área del círculo y recibe como parámetros el valor de pi y el radio.

CREATE FUNCTION CalcularAreaCirculo(@pi FLOAT, @radio FLOAT)
RETURNS FLOAT
BEGIN
        DECLARE @area FLOAT;
        -- Fórmula para calcular el área del círculo: π * radio^2
        SET @area = @pi * @radio * @radio;
```

```
        RETURN @area;
END;
```

4.16 Métodos

Los métodos en QueryCrypter serán tratados como Procedimientos almacenados, un objeto de SQL que realiza algún procesamiento y a diferencia de una función no devuelve un valor como resultado.

```
CREATE PROCEDURE nombre_procedimiento
        -- Parámetros de entrada (si los hay)
        @parametro1 tipo_de_dato,
        @parametro2 tipo_de_dato
AS
BEGIN
       -- Código del procedimiento aquí
       <SENTENCIAS>
END;
```

Ejemplo:

```
CREATE PROCEDURE SumarNumeros
        @numero1 INT,
        @numero2 INT,
        @resultado INT
AS
BEGIN
        SET @resultado = @numero1 + @numero2;
        PRINT 'La suma es:  ' + CAST(@resultado AS VARCHAR);
END;
```

4.17 LLamadas

La llamada a una función o método se realiza como parte de una consulta SQL o como parte de una instrucción. Los parámetros se asocian normalmente por posición, aunque opcionalmente, también se pueden asociar por nombre.

```
SELECT nombre_funcion_metodo(parametro1, parametro2, ...);

o

SET @nombre_variable = nombre_funcion_metodo(parametro1, parametro2, ...);
```

Ejemplo:

```
-- Llamar a la función para calcular el área de un círculo con radio 5.
DECLARE @radioCirculo FLOAT = 5.0;
DECLARE @pi FLOAT = 3.14159265359;
DECLARE @areaCirculo FLOAT;

SET @areaCirculo = CalcularAreaCirculo(@pi, @radioCirculo);

PRINT 'El área del círculo con radio ' + CAST(@radioCirculo AS
VARCHAR) + ' es ' + CAST(@areaCirculo AS VARCHAR);
```

4.18 Funciones Nativas

Estas funciones nativas son herramientas útiles para manipular y transformar datos en consultas SQL y para realizar operaciones matemáticas, de texto y de tipo de datos.

4.18.1 Print

Se utiliza para imprimir expresiones.

```
PRINT<EXPRESIÓN>;
```

4.18.2 Lower

Convierte una cadena de texto a minúsculas.

```
SELECT LOWER('HOLA MUNDO');
```

4.18.3 Upper

Convierte una cadena de texto a mayúsculas.

```
SELECT UPPER('hola mundo');
```

4.18.4 Round

Redondea un número decimal al número entero más cercano o a una cantidad específica de decimales.

```
SELECT ROUND(num_redondear, cantidad_decimales);
SELECT ROUND(5.678, 2);  -- Redondea a 2 decimales
```

4.18.5 Length

Devuelve la longitud (número de caracteres) de una cadena de texto.

```
SELECT LEN('Hola mundo');  --10
```

4.18.6 Truncate

Trunca un número decimal para eliminar los decimales.

```
SELECT TRUNCATE(num_truncar, cantidad_decimales);
SELECT TRUNCATE(8.945, 1);  -- Trunca a 1 decimal
```

4.18.7 TypeOf

Devuelve el tipo de datos de una expresión o valor.

```
SELECT TYPEOF(123);  -- Devuelve el tipo INT
```

5. Reportes

Los reportes son una parte fundamental de QueryCrypter, ya que muestra de forma visual las herramientas utilizadas para realizar la ejecución del código.

A continuación, se muestran ejemplos de estos reportes. (Queda a discreción del estudiante el diseño de estos, solo se pide que sean totalmente legibles).

5.1 Tabla de símbolos

Este reporte mostrará la tabla de símbolos después de la ejecución. Se deberán mostrar todas las variables, funciones y métodos declarados, así como su tipo, entorno y toda la información que se considere necesaria.

Identificador	Tipo	Tipo	Entorno	Línea	Columna
numero1	Variable	Entero	Función sumar	15	4
numero2	Variable	Decimal	Función sumar	16	1
SumarNumeros	Funcion	Decimal	-	50	6

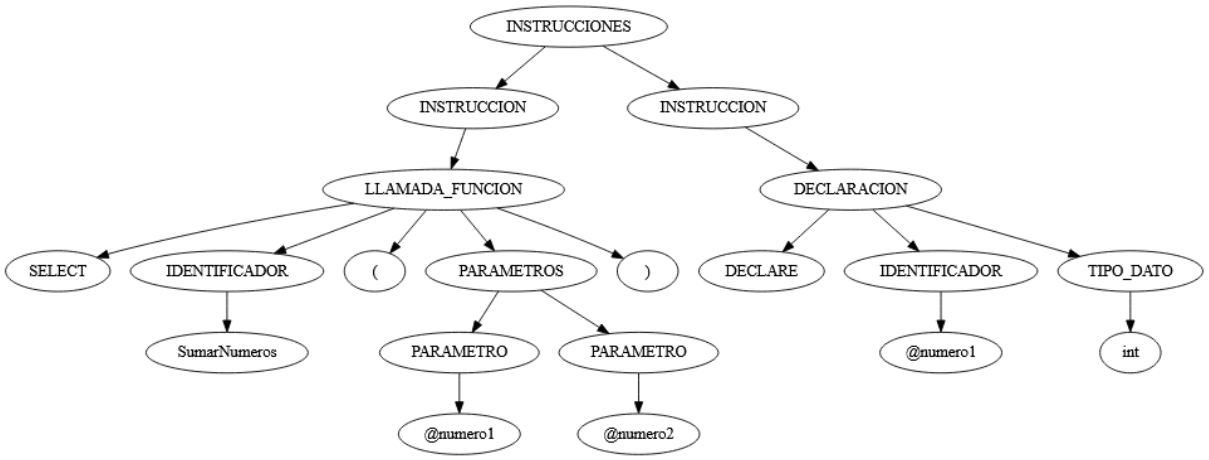
5.2 Tabla de errores

El reporte de errores debe contener la información suficiente para detectar y corregir errores en el código fuente, además en la descripción se debe de agregar la instrucción que se descartó y el carácter con el que se sincronizó al entrar en modo pánico.

Tipo de error	Descripción	Línea	Columna
Léxico	El carácter “$” no pertenece al lenguaje.	26	17
Sintáctico	Variable encontrada “@nota”, se esperaba Palabra Reservada “DECLARE” Entrada descartada: "@nota varchar;" carácter de sincronización: “;”	170	8

5.3 AST

Este reporte muestra el árbol de sintaxis abstracta producido al analizar los archivos de entrada, debe ser representado como un grafo y mostrar los nodos que el estudiante considere necesarios para describir el flujo realizado para analizar e interpretar sus archivos de entrada.



Nota: Se sugiere a los estudiantes utilizar la herramienta Graphviz para graficar su AST

5.4 Salidas en consola

La consola es el área de salida del intérprete. Por medio de esta herramienta se podrán visualizar las salidas generadas, así como los errores léxicos y sintácticos.

```
>Este es un mensaje desde mi interprete de OLC1
>
>
>Error Lexico: El carácter '#' no reconocido en la linea 7 y columna 28.
```

6. Requerimientos Mínimos

Para que el estudiante tenga derecho a calificación, deberá cumplir con lo siguiente:

1. Interfaz gráfica funcional
2. Analizador léxico y sintáctico
3. Implementación del patrón intérprete
4. Operaciones aritméticas, lógicas y relacionales
5. Declaración y asignación de variables.
6. Función Print
7. Ejecución de instrucciones DDL y DML.
8. Reportes
9. Manual de Usuario
10. Manual Técnico
11. Archivo de Gramáticas

7. Entregables:

- Código Fuente del proyecto.
- Manual de Usuario.
- Manual Técnico
- Archivo de Gramática para la solución en formato **BNF** (El archivo debe de ser limpio, entendible y no debe ser una copia del archivo de jison).
- Link al repositorio privado de Github en donde se encuentra su proyecto (Se debe de agregar como colaborador al auxiliar que le califique).

8. Restricciones

1. Lenguajes de programación a usar: **Javascript/Typescript.**
2. Puede utilizar frameworks como Angular, React, Vuejs, etc para generar su entorno gráfico. Queda a discreción del estudiante.
3. Puede utilizar herramientas como Nodejs para Javascript.
4. Herramientas de análisis léxico y sintáctico: **Jison**
5. **El Proyecto es individual.**
6. Para graficar se puede utilizar cualquier librería (Se recomienda Graphviz)
7. **Copias completas/parciales** de: código, gramática, etc. serán merecedoras de una **nota de 0 puntos**, los responsables serán reportados al catedrático de la sección y a la Escuela de Ciencias y Sistemas.
8. La calificación tendrá una duración de 30 minutos, acorde al programa del laboratorio.

9. Fecha de Entrega

Domingo 22 de Octubre de 2023 a las 23:59. La entrega será por medio de la plataforma UEDI. Entregas fuera de la fecha indicada, no se calificarán.

SE LE CALIFICARA DEL ÚLTIMO COMMIT REALIZADO ANTERIOR A ESTA FECHA.