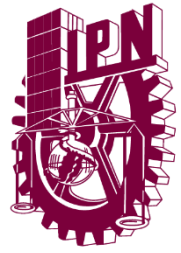




**Instituto Politécnico Nacional  
Escuela Superior de Cómputo  
Ingeniería en Sistemas Computacionales**



## **Practica 4**

**Grupo: 3CV5**

**Materia: Compiladores**

**Profesor:**

**M.C. Saucedo Delgado Rafael Norman**

**Alumno:**

**Cruz Heras Joel Antonio**

**Boleta:**

**2014080325**

## Introducción

En esta practica se realizará la implementación de un analizador léxico de un lenguaje dado con el objetivo de identificar y clasificar las diferentes clases necesarias para la implementación correcta del lenguaje que se analice, es decir, identificar el uso correcto de las palabras, operadores y/o símbolos que conformen el lenguaje de entrada. Las instrucciones dadas recomiendan que para esta práctica se use el lenguaje que se pretenda utilizar para el proyecto final de la materia, en este caso será el lenguaje de consulta estructurada (SQL en sus siglas en ingles) para lo cual se realizo la investigación de la definición de los tokens, palabras reservadas para SQL. A continuación, se realiza una breve definición del lenguaje SQL.

SQL (por sus siglas en inglés Structured Query Language) es un lenguaje de dominio específico utilizado en programación, diseñado para administrar, y recuperar información de sistemas de gestión de bases de datos relacionales. SQL pasó a ser el estándar del Instituto Nacional Estadounidense de Estándares (ANSI) en 1986 y de la Organización Internacional de Normalización (ISO) en 1987. Desde entonces, el estándar ha sido revisado para incluir más características. A pesar de la existencia de ambos estándares, la mayoría de los códigos SQL no son completamente portables entre sistemas de bases de datos diferentes sin otros ajustes.[1]

El lenguaje SQL se basa en varios elementos. Para la comodidad de los desarrolladores de SQL todos los comandos del lenguaje necesarios en los correspondientes sistemas de gestión de bases se ejecutan a través de una interfaz específica de línea de comandos SQL (command-line interface o CLI).[2] Estos elementos son.

- **Cláusulas:** las cláusulas son componentes de los estados y las queries.
- **Expresiones:** las expresiones pueden producir valores escalares o tablas, que consisten en columnas y filas de datos.
- **Predicados:** que especifican las condiciones que se utilizan para limitar los efectos de los comandos y las consultas, o para cambiar el flujo del programa.
- **Querys:** una query o consulta va a recuperar los datos, con base a un criterio dado.
- **Comandos:** con los comandos puedes controlar las operaciones, el flujo del programa, conexiones, sesiones, o diagnósticos. En los sistemas de bases de datos los comandos o sentencias SQL se utilizan para el envío de consultas desde un programa cliente a un servidor donde se almacenan las bases de datos. Como respuesta, el servidor procesa los comandos SQL y devuelve respuestas al programa cliente. Esto permite a los usuarios

ejecutar una amplia gama de maravillosas y rápidas operaciones de manipulación de datos, desde simples entradas de datos a complicadas queries.

UPDATE clause	{UPDATE country	}	statement
SET clause	{SET population =		
WHERE clause	{WHERE name =		
		expression	
		expression	
		predicate	

Ejemplo de uso de las clausulas UPDATE, SET y WHERE, juntos generan un comando

Para realizar la implementación del analizador léxico se hará uso de la herramienta FLEX en distribuciones de Linux. FLEX es, como lo dice el manual, un generador de analizadores léxicos rápidos.

flex es una herramienta para generar escáneres: programas que reconocen patrones léxicos en un texto. flex lee los ficheros de entrada dados, o la entrada estándar si no se le ha indicado ningún nombre de fichero, con la descripción de un escáner a generar. La descripción se encuentra en forma de parejas de expresiones regulares y código C, denominadas reglas. flex genera como salida un fichero fuente en C, lex.yy.c, que define una rutina yylex(). Este fichero se compila y se enlaza con la librería -lfl para producir un ejecutable. Cuando se arranca el fichero ejecutable, este analiza su entrada en busca de casos de las expresiones regulares. Siempre que encuentra uno, ejecuta el código C correspondiente.[3]

El fichero de entrada de flex está compuesto de tres secciones, separadas por una línea donde aparece únicamente un %% en esta:

definiciones

%%

reglas

%%

código de usuario

En el bloque de definiciones se definen expresiones regulares más sencillas para reducir la carga de información en el bloque de reglas. Un ejemplo de definición es:

DIGITO [0-9] //Definición de la expresión regular para un dígito

Y este puede usarse en el bloque de reglas de la forma:

```
{DIGITO}+ {printf("Un entero");} //Expresión regular para un entero
```

En el bloque de reglas se definen las reglas en las cuales se define un patrón y la acción que se debe realizar cuando se cumpla el patrón, siguiendo con el ejemplo anterior el patrón es {DIGITO}+ el cual es una expresión regular para una palabra que contenga uno o más dígitos y la acción es lo que se encuentre entre los corchetes, en este caso imprimir la frase "Un entero".

En el bloque de código de usuario se define código de complemento, es decir, rutinas que el programador puede agregar a la implementación o puede omitir si no lo necesita. Un punto importante a destacar es que para agregar bibliotecas necesarias para la implementación de este código se debe realizar entre símbolos %% antes del bloque de definiciones.

## Desarrollo

Para comenzar la implementación se definen los pasos a seguir para el desarrollo e implementación de lo que se solicita.

1. Ejemplificar el lenguaje
2. Identificar las clases léxicas
3. Crear los patrones en lex para identificar cada clase léxica
4. Codificar en lex el punto anterior
5. Ejecutar pruebas

### 1-Ejemplo del lenguaje

El lenguaje que se propuso como entrada al ser un lenguaje existente y estandarizado conlleva que ya existen reglas y sintaxis definidas para usar el lenguaje. A continuación, se muestran unos ejemplos de la implementación de SQL.

```
SELECT [DISTINCT] select_list
```

```
FROM from_clause
```

```
[WHERE search_condition]
```

```
[GROUP BY column {, column}]
```

[HAVING search\_condition]]

[ORDER BY column {, column}]

Sintaxis para obtener información de una tabla 'select\_list'

Donde:

- SELECT, FROM, WHERE, GROUP BY y ORDER BY son las cláusulas para ejecutar una acción y dependiendo de estas requieren datos y/o condiciones para ser ejecutadas.
- [DISTINCT] puede ser el símbolo \* que selecciona todo lo que se encuentre en la lista seleccionada o se pueden escoger los campos requeridos encerrando el nombre de la columna entre comillas simples y separadas por comas.
- Las palabras en minúscula son palabras que se definen como variables para nombrar columnas, especificar información a obtener, información a insertar en las tablas, etc.

CREATE DATABASE 'nombredb'

CREATE TABLE 'nombretabla'(  
columna1      tipodato,  
columna2      tipodato,  
columna3      tipodato  
)

INSERT INTO 'nombretabla' ('columna1','columna2','columna3') VALUES  
( 'valor1','valor2','valor3')

Ejemplo de creación de una base de datos, una tabla e inserción de información en una  
tabla

## 2-Clases léxicas

Para diferenciar las clases a las que pertenecen las distintas palabras reservadas para el lenguaje SQL y el uso de palabras no reservadas se clasificaron según su uso y acciones que realiza con base en su orden sintáctico. Un ejemplo, las palabras en minúsculas y encerradas entre comillas simples siempre serán una expresión literal ya que en SQL las palabras reservadas se escriben con mayúsculas, en el caso de las palabras reservadas se les asigno clases dependiendo de su importancia para crear una instrucción. A continuación, se listan las clases léxicas utilizadas.

- Expresión literal
- Dígitos
- Expresión literal de fecha
- Tipo de dato
- Sentencia
- Expresión condicional
- Operador
- Operador lógico
- Clausula
- Característica de dato
- Función
- Función de tiempo
- Llave

### **3-Creación de los patrones de flex**

Los patrones que se utilizaron para flex se crearon con base a la definición del lenguaje SQL dada en la pagina de la referencia [4] y las funciones descritas en el libro “flex & bison” usando las funciones especificadas dentro de estos y omitiendo algunas funciones a las cuales el funcionamiento no me es familiar y por lo tanto no los pude clasificar. Para optimizar la clasificación, como lo sugiere el manual de flex, se utilizo el bloque de definiciones para definir los caracteres, símbolos y números para tener mas clara la identificación de las palabras reservadas. Debajo se muestran ejemplos de las definiciones y una clase léxica como será programada en flex.

digito [0-9]

letras [a-zA-Z]

espacios [ ]

salto [\n]

separador [,]

operadores [\*+-/]

%%

```
{letras}*{digito}*    { printf("<Expresion literal>");}
{digito}*              {printf("<Digito>");}
```

#### 4-Codificación de los patrones de flex

La parte de la codificación se realizó con base al manual de flex y a la clase y ejemplos dados sobre el uso de flex que se dieron en el curso de la materia de compiladores. La codificación completa no se mostrara en este documento, ya que no se apreciara completo. En el archivo adjunto "léxico.l" se muestra a detalle la codificación.

#### 5-Pruebas

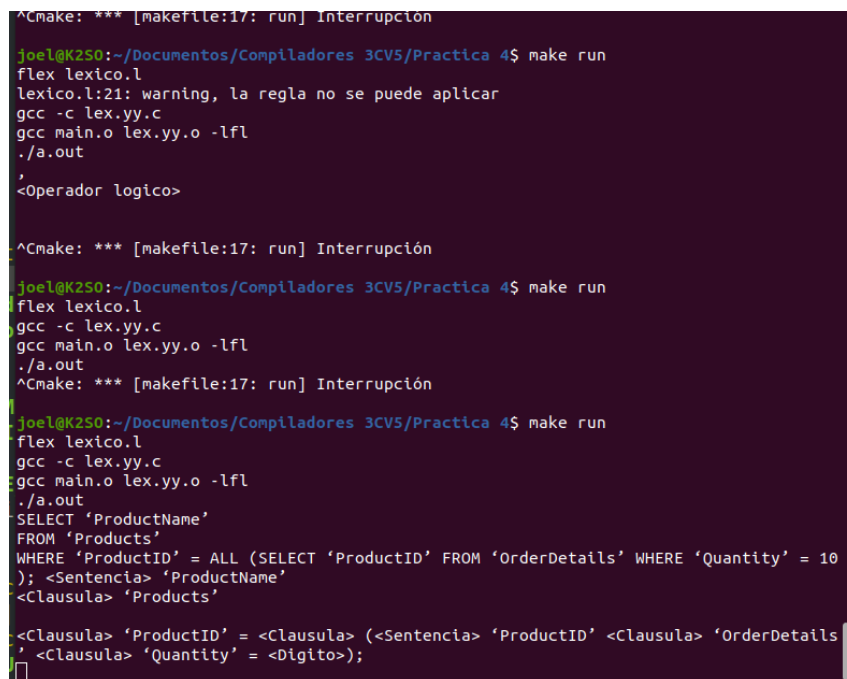
Se realizaron 4 pruebas de uso del código generado, la entrada así como la salida se muestran debajo.

Prueba 1

SELECT 'ProductName'

FROM 'Products'

WHERE 'ProductID' = ALL (SELECT 'ProductID' FROM 'OrderDetails' WHERE 'Quantity' = 10);



```
^Cmake: *** [makefile:17: run] Interrupción
joel@K250:~/Documentos/Compiladores 3CV5/Practica 4$ make run
flex lexico.l
lexico.l:21: warning, la regla no se puede aplicar
gcc -c lex.yy.c
gcc main.o lex.yy.o -lfl
./a.out
<Operador logico>
^Cmake: *** [makefile:17: run] Interrupción
joel@K250:~/Documentos/Compiladores 3CV5/Practica 4$ make run
flex lexico.l
gcc -c lex.yy.c
gcc main.o lex.yy.o -lfl
./a.out
^Cmake: *** [makefile:17: run] Interrupción
joel@K250:~/Documentos/Compiladores 3CV5/Practica 4$ make run
flex lexico.l
gcc -c lex.yy.c
gcc main.o lex.yy.o -lfl
./a.out
SELECT 'ProductName'
FROM 'Products'
WHERE 'ProductID' = ALL (SELECT 'ProductID' FROM 'OrderDetails' WHERE 'Quantity' = 10
); <Sentencia> 'ProductName'
<Clausula> 'Products'
<Clausula> 'ProductID' = <Clausula> (<Sentencia> 'ProductID' <Clausula> 'OrderDetails'
' <Clausula> 'Quantity' = <Digito>);
```

Salida

## Prueba 2

ALTER TABLE 'Customers'

ADD 'Email' VARCHAR(255);

```
joel@K250: ~/Documentos/Compiladores 3CV5/Practica 4
joel@K250:~/Documentos/Compiladores 3CV5/Practica 4$ make clear
make: *** No hay ninguna regla para construir el objetivo 'clear'. Alto.
joel@K250:~/Documentos/Compiladores 3CV5/Practica 4$ ls
a.out lexico.l lex.yy.c lex.yy.o main.c main.o makefile
joel@K250:~/Documentos/Compiladores 3CV5/Practica 4$ make clear
make: *** No hay ninguna regla para construir el objetivo 'clear'. Alto.
joel@K250:~/Documentos/Compiladores 3CV5/Practica 4$ make clean
rm -f a.out main.o lex.yy.o lex.yy.c
joel@K250:~/Documentos/Compiladores 3CV5/Practica 4$ make run
gcc -c main.c
main.c: In function 'main':
main.c:2:2: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
    2 | yylex();/*Inicia el autómata del analizador lexico*/
      |
flex lexico.l
gcc -c lex.yy.c
gcc main.o lex.yy.o -lfl
./a.out
ALTER TABLE 'Customers'
ADD 'Email' VARCHAR(255);<Sentencia> <Clausula> 'Customers'

<Funcion> 'Email' <Tipo de dato>(<Digito>);

^Cmake: *** [makefile:17: run] Interrupción
joel@K250:~/Documentos/Compiladores 3CV5/Practica 4$
```

Salida



### Prueba 3

SELECT TIMESTAMPDIF

(SQL\_TSI\_DAY, TIMESTAMP'1998-07-31 23:35:00',TIMESTAMP'2000-04-01 14:24:00')

FROM Employee WHERE employeeid = 2;

```
joel@K250: ~/Documentos/Compiladores 3CV5/Practica 4
joel@K250:~/Documentos/Compiladores 3CV5/Practica 4$ make run
gcc -c main.c
main.c: In function 'main':
main.c:2:2: warning: implicit declaration of function 'yylex' [-Wimplicit-function-declaration]
   2 | yylex();/*Inicia el autómata del analizador lexico*/
     | ^~~~~~
flex lexico.l
gcc -c lex.yy.c
gcc main.o lex.yy.o -lfl
./a.out
ALTER TABLE 'Customers'
ADD 'Email' VARCHAR(255);<Sentencia> <Clausula> 'Customers'

<Funcion> 'Email' <Tipo de dato>(<Digito>);

^Cmake: *** [makefile:17: run] Interrupción
joel@K250:~/Documentos/Compiladores 3CV5/Practica 4$ make run
./a.out
SELECT TIMESTAMPDIF
(SQL_TSI_DAY, TIMESTAMP'1998-07-31 23:35:00',TIMESTAMP'2000-04-01 14:24:00')
FROM Employee WHERE employeeid = 2;<Sentencia> <Funcion de tiempo>DIFF
(SQL_TSI_DAY<Operador logico> <Funcion de tiempo>'<Digito><Operador logico><Digito><Operador logico><Digito>:<Digito>:<Digito>'<Operador logico><Funcion de tiempo>'<Digito><Operador logico><Digito><Operador logico><Digito> <Digito>:<Digito>:<Digito>')
<Clausula> Employee <Clausula> employeeid = <Digito>;
```

Salida

## Prueba 4

USE AdventureWorks2012;

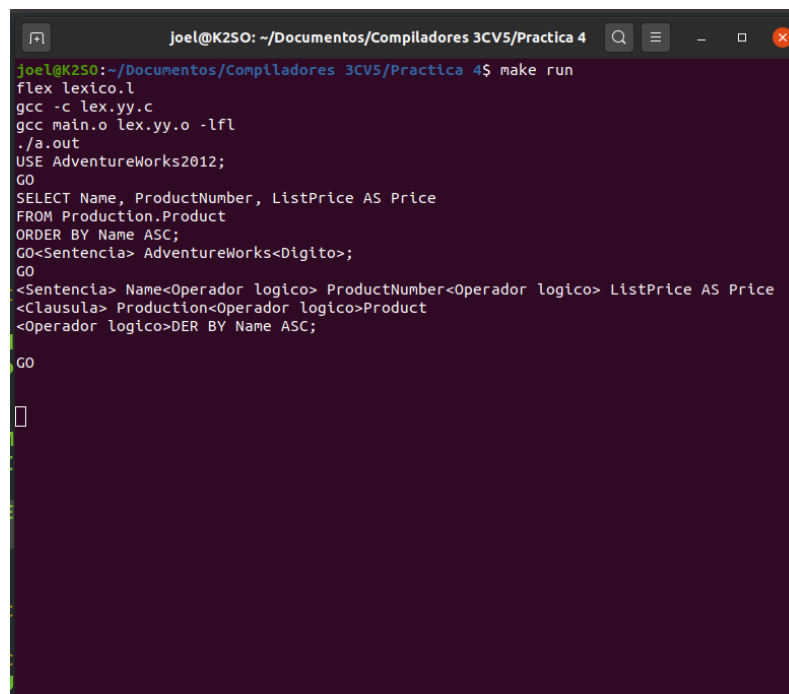
GO

SELECT Name, ProductNumber, ListPrice AS Price

FROM Production.Product

ORDER BY Name ASC;

GO



```
joel@K250: ~/Documentos/Compiladores 3CV5/Practica 4
joel@K250:~/Documentos/Compiladores 3CV5/Practica 4$ make run
flex lexico.l
gcc -c lex.yy.c
gcc main.o lex.yy.o -lfl
./a.out
USE AdventureWorks2012;
GO
SELECT Name, ProductNumber, ListPrice AS Price
FROM Production.Product
ORDER BY Name ASC;
GO<Sentencia> AdventureWorks<Digito>;
GO
<Sentencia> Name<Operador logico> ProductNumber<Operador logico> ListPrice AS Price
<Clausula> Production<Operador logico>Product
<Operador logico>DER BY Name ASC;
GO
[]
```

Salida

## Conclusiones

Para esta práctica tuve que desempolvar mis conocimientos de SQL y además investigar sobre las capacidades y limitaciones de la herramienta flex para conseguir realizar los puntos que se requerían. En cuanto a la implementación estoy consciente que faltan funciones o palabras reservadas que se encuentran especificadas dentro del lenguaje SQL, pero dado nunca había usado esas funciones no me fue posible asignarles una clasificación. Como implementación no es la más optima, pero para comenzar a familiarizarme y desarrollar mis habilidades con flex fue un buen ejercicio. Espero mejorar mis implementaciones en el futuro

## Referencias

- [1] Anonimo. (2020). SQL. 2020-11-12, de Wikipedia Sitio web: <https://es.wikipedia.org/wiki/SQL>
- [2] Expertos, E. (2018, 21 marzo). Lenguaje SQL, historia y conceptos básicos | VIU. Universidad Internacional de Valencia. <https://www.universidadviu.com/lenguaje-sql-historia-conceptos-basicos/>
- [3] C. (2019). Ubuntu Manpage: flex - generador de analizadores léxicos rápidos. Ubuntu.com. <http://manpages.ubuntu.com/manpages/bionic/es/man1/flex.1.html>
- [4] Oracle. (s. f.). Sintaxis y Semántica de SQL. [exlibrisgroup.com](http://exlibrisgroup.com). Recuperado 11 de diciembre de 2020, de [https://analytics-na01.alma.exlibrisgroup.com/analytics/olh/l\\_es/sql\\_syntax\\_semantics.htm](https://analytics-na01.alma.exlibrisgroup.com/analytics/olh/l_es/sql_syntax_semantics.htm)
- [5] Levine, J. R. (2009). Flex & Bison (1.a ed.) [Libro electrónico]. Oreilly & Associates Inc. [http://web.iitd.ac.in/~sumeet/flex\\_bison.pdf](http://web.iitd.ac.in/~sumeet/flex_bison.pdf), pag. 89