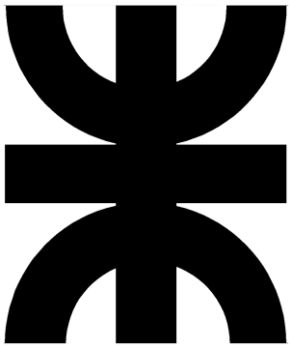

UNIVERSIDAD TECNOLÓGICA NACIONAL

FACULTAD REGIONAL RESISTENCIA

INGENIERÍA EN SISTEMAS DE INFORMACIÓN

SINTAXIS Y SEMÁNTICA DE LOS LENGUAJES

Integrantes:

- **Acevedo, Ariel Alexander.**
- **Obregón, Elías Javier.**
- **Thouzeau, Edgardo Hernán.**

Comisión: 2° “A”

AÑO 2018

Índice

Analizador SINTACTICO (2 entrega. Final)

El analizador sintáctico, también llamado parser, recibe como entrada los tokens que le pasa el Analizador Léxico (el analizador sintáctico no maneja directamente caracteres) y comprueba si esos tokens van llegando en el orden correcto (orden permitido por el lenguaje). Así pues, sus funciones son:

1. Aceptar lo que es válido sintácticamente y rechazar lo que no lo es.
2. Hacer explícito el orden jerárquico que tienen los operadores en el lenguaje de que se trate.
3. Guiar el proceso de traducción (traducción dirigida por la sintaxis).

Para esta etapa se solicita la construcción de una Gramática que genere el lenguaje a reconocer

Algunas restricciones para facilitar el trabajo e implementación:

- Una sentencia SQL se compone de al menos de la cláusula SELECT y FROM. La cláusula WHERE, ORDER BY y GROUP BY son opcionales.
- Si aparecen todas, el orden es: SELECT, FROM, WHERE, ORDER BY y GROUP BY.
- Los nombres de palabras reservadas e identificadores no son case sensitive.
- No se pueden definir dos tablas con el mismo nombre y dentro de una tabla, no puede haber dos columnas con el mismo nombre.
- Las columnas referenciadas en las cláusulas SELECT y WHERE deben cumplir:
    - Si se usó la forma tabla.columna, entonces tabla debe estar mencionada en la cláusula
- FROM y columna debe pertenecer a esa tabla.
    - Si se usó la forma columna, debe haber exactamente una tabla mencionada en la
- cláusula FROM que tenga una columna con ese nombre.
- Las comparaciones sólo se pueden hacer entre campos o constantes de igual tipo.
- En la cláusula SELECT pueden aparecer varios identificadores separados por comas, o
- funciones de agregados.
- Los identificadores pueden venir indicados mediante el atributo o bien mediante
- nombre_de_la_tabla.atributo.
- En la cláusula FROM pueden aparecer varios nombres de tablas separados por comas.
- En la cláusula WHERE pueden aparecer combinaciones de expresiones relacionales y
- operadores lógicos. Los operadores lógicos tienen mayor prioridad que los relacionales y que
- puede existir anidamiento en las expresiones.
- Supondremos que el programador puede escribir más de una sentencia SQL consecutiva. Una
- sentencia viene separada de otra por el carácter;

Consideraciones

1ra Entrega:

Para la realización del analizador léxico utilizamos el lenguaje de programación JAVA, y la herramienta JFlex con la IDE NetBeans 8.2 y JDK 8.1.7.1 en Windows 10. Durante el proceso de implementación del JFlex no se presentaron mayores inconvenientes debido a las guías y documentación que disponíamos.

Como resultado obtuvimos un archivo Analizador_Lexico.jar, el cual para su ejecución requiere de la JVM (Java Virtual Machine).

Los métodos de entrada se realizan manualmente mediante la interfaz gráfica que se ejecuta al abrir el archivo .jar. No esta implementada la funcionalidad de ingresar el directorio de un archivo de tipo texto para su analisis.

El ingreso mediante pantalla generará un archivo llamado "fichero.txt", donde es guardado el contenido ingresado y el mismo es utilizado para ejecutar el análisis; debido a que Jflex solo admite la entrada java.io.Reader.

Para el reconocimiento de las palabras reservadas en SQL solo se admiten mayusculas, de lo contrario serán reconocidas como ID.

En el caso de no reconocer alguna sentencia se mostrará un mensaje de error continuando con las siguientes sentencias.

2da Entrega:

Para la realización del analizador léxico utilizamos el lenguaje de programación JAVA, y la herramienta JFlex 1.6.1 y CUP 11b con la IDE NetBeans 8.2 y JDK 8.1.7.1 en Windows 10. Durante el proceso de implementación del JFlex y CUP no se presentaron mayores inconvenientes debido a las guías y documentación que disponíamos.

Como resultado obtuvimos un archivo Analizador_Sintactico.jar, el cual para su ejecución requiere de la JVM (Java Virtual Machine).

Los métodos de entrada se realizan manualmente mediante la interfaz gráfica que se ejecuta al abrir el archivo .jar. Ya se encuentra implementada la funcionalidad de ingresar un archivo de tipo texto (mediante el file chooser) para su análisis, el cual escribe el contenido del archivo en la ventana de ingreso de texto. En el caso de intentar abrir un archivo distinto a *.txt el programa informará que el archivo no es compatible.

El ingreso mediante pantalla generará un archivo llamado "entrada.txt", donde es guardado el contenido ingresado y el mismo es utilizado para ejecutar el análisis; debido a que Jflex solo admite la entrada java.io.Reader.

Para el reconocimiento de las palabras reservadas en SQL ya se admite la posibilidad de mezclar Mayúsculas y Minúsculas dado que CUP contiene una función llamada %ignorecase que permite que esto se pueda hacer.

En el caso de no reconocer alguna sentencia se mostrará un mensaje de error mostrando donde se encontró el error.

Algunas consideraciones: para el ingreso de texto de la siguiente forma 'Juan pedro' en las sentencias WHERE solo se admite la utilización de apostrofes y no comillas simples.

Intentamos contemplar la mayor cantidad de opciones posibles que pudiesen tener las estructuras de CREATE y SELECT, no teniendo control sobre contenidos repetibles como ID, las sentencias PRIMARY KEY, y otros.

Se adjunta una serie de ejemplos utilizados y creados para conocer el comportamiento del analizador sintáctico.

Tokens

```
//---> Palabras Reservadas
"CREATE"        ----------Token.TKN_CREATE
"TABLE"         ----------Token.TKN_TABLE
"SELECT"        ----------Token.TKN_SELECT
"FROM"          ----------Token.TKN_FROM
"WHERE"         ----------Token.TKN_WHERE
"GROUP"         ----------Token.TKN_GROUP
"ORDER"         ----------Token.TKN_ORDER
"BY"    ----------Token.TKN_BY
"NOT"   ----------Token.TKN_NOT
"*"     ----------Token.TKN_TODO
"PRIMARY"       ----------
Token.TKN_PRIMARY
"KEY"   ----------Token.TKN_KEY
"NULL"  ----------Token.TKN_NULL

//----> Funciones SQL
"COUNT"         ----------Token.TKN_COUNT
"AVG"   ----------Token.TKN_AVG
"SUM"   ----------Token.TKN_SUM
"MIN"   ----------Token.TKN_MIN
"MAX"   ----------Token.TKN_MAX

//----> Tipos de Datos
"VARCHAR"       ----------Token.TKN_VARCHAR
"CHAR"          ----------Token.TKN_CHAR
"STRING"        ----------Token.TKN_STRING
"INTEGER"       ----------Token.TKN_INTEG
"INT"   ----------Token.TKN_INTEG
```

```
//----> Simbolos
"("     ----------Token.TKN_APAR
")"     ----------Token.TKN_CPAR
";"     ----------Token.TKN_PTOCOMA
"."     ----------Token.TKN_PTO
","     ----------Token.TKN_COMA
"<"     ----------Token.TKN_MEN
">"     ----------Token.TKN_MAY
">="    ----------Token.TKN_MAYIGU
"<="    ----------Token.TKN_MENIGU
"="     ----------Token.TKN_IGUAL
"!="    ----------Token.TKN_DIST
"and"   ----------Token.TKN_AND
"or"    ----------Token.TKN_OR
"'"     ----------Token.TKN_APOST
"+"     ----------Token.TKN_PLUS
"-"     ----------Token.TKN_MINUS
"/"     ----------Token.TKN_DIVID

//----> Expresiones Regulares
{Letras} ({Letras}|{Digito})* ---------Token.TKN_ID,
 [-+]?{Digito}+    ----------
Token.TKN_INT
```

Lexema

```
package Analizador_Sintactico;
import java_cup.runtime.*;
import java.util.LinkedList;

/*------------- Area2: Opciones y Declaraciones -------------*/
%%
%{
//----> Codigo de Usuario
   public static LinkedList<Tabla_Error> Tabla_ERROR_LEXICO = new LinkedList<Tabla_Error>();
   public String resultado ="";
%}
//----> Directivas de Jflex
%public
%class Lexer
%cupsym Token
%cup
%char
%column
%full
%ignorecase
%line
%unicode
//----> %type Token VER ESTA SENTENCIA llama al conjunto ENUM TOKEN
//----> Expresiones Regulares
Letras = [a-zA-z_ñÑ]
Digito = [0-9]
White = [ \t\r\n]

//----> Estados
%%

/*------------------ Area3: Reglas Lexicas ------------------*/
//----> Contexto General

   {White} {/*Ignore*/}
   "//".* {/*Ignore*/}

//----> Palabras Reservadas

<YYINITIAL>   "CREATE" {return new Symbol(Token.TKN_CREATE, yycolumn, yyline, yytext());}
<YYINITIAL>   "TABLE" {return new Symbol(Token.TKN_TABLE, yycolumn, yyline, yytext());}
<YYINITIAL>   "SELECT" {return new Symbol(Token.TKN_SELECT, yycolumn, yyline, yytext());}
<YYINITIAL>   "FROM" {return new Symbol(Token.TKN_FROM, yycolumn, yyline, yytext());}
<YYINITIAL>   "WHERE" {return new Symbol(Token.TKN_WHERE, yycolumn, yyline, yytext());}
<YYINITIAL>   "GROUP" {return new Symbol(Token.TKN_GROUP, yycolumn, yyline, yytext());}
```

```
<YYINITIAL>    "ORDER" {return new Symbol(Token.TKN_ORDER, yycolumn, yyline, yytext());}
<YYINITIAL>    "BY" {return new Symbol(Token.TKN_BY, yycolumn, yyline, yytext());}
<YYINITIAL>    "NOT" {return new Symbol(Token.TKN_NOT, yycolumn, yyline, yytext());}
<YYINITIAL>    "*" {return new Symbol(Token.TKN_TODO, yycolumn, yyline, yytext());}
<YYINITIAL>    "PRIMARY" {return new Symbol(Token.TKN_PRIMARY, yycolumn, yyline, yytext());}
<YYINITIAL>    "KEY" {return new Symbol(Token.TKN_KEY, yycolumn, yyline, yytext());}
<YYINITIAL>    "NULL" {return new Symbol(Token.TKN_NULL, yycolumn, yyline, yytext());}

//----> Funciones SQL

<YYINITIAL>    "COUNT" {return new Symbol(Token.TKN_COUNT, yycolumn, yyline, yytext());}
<YYINITIAL>    "AVG" {return new Symbol(Token.TKN_AVG, yycolumn, yyline, yytext());}
<YYINITIAL>    "SUM" {return new Symbol(Token.TKN_SUM, yycolumn, yyline, yytext());}
<YYINITIAL>    "MIN" {return new Symbol(Token.TKN_MIN, yycolumn, yyline, yytext());}
<YYINITIAL>    "MAX" {return new Symbol(Token.TKN_MAX, yycolumn, yyline, yytext());}

//----> Tipos de Datos

<YYINITIAL>    "VARCHAR" {return new Symbol(Token.TKN_VARCHAR, yycolumn, yyline, yytext());}
<YYINITIAL>    "CHAR" {return new Symbol(Token.TKN_CHAR, yycolumn, yyline, yytext());}
<YYINITIAL>    "STRING" {return new Symbol(Token.TKN_STRING, yycolumn, yyline, yytext());}
<YYINITIAL>    "INTEGER" {return new Symbol(Token.TKN_INTEG, yycolumn, yyline, yytext());}
//<YYINITIAL>    "FLOAT" {return new Symbol(Token.TKN_FLOAT, yycolumn, yyline, yytext());}

//----> Simbolos

<YYINITIAL>    "(" {return new Symbol(Token.TKN_APAR, yycolumn, yyline, yytext());}
<YYINITIAL>    ")" {return new Symbol(Token.TKN_CPAR, yycolumn, yyline, yytext());}
<YYINITIAL>    ";" {return new Symbol(Token.TKN_PTOCOMA, yycolumn, yyline, yytext());}
<YYINITIAL>    "." {return new Symbol(Token.TKN_PTO, yycolumn, yyline, yytext());}
<YYINITIAL>    "," {return new Symbol(Token.TKN_COMA, yycolumn, yyline, yytext());}
<YYINITIAL>    "<" {return new Symbol(Token.TKN_MEN, yycolumn, yyline, yytext());}
<YYINITIAL>    ">" {return new Symbol(Token.TKN_MAY, yycolumn, yyline, yytext());}
<YYINITIAL>    ">=" {return new Symbol(Token.TKN_MAYIGU, yycolumn, yyline, yytext());}
<YYINITIAL>    "<=" {return new Symbol(Token.TKN_MENIGU, yycolumn, yyline, yytext());}
<YYINITIAL>    "=" {return new Symbol(Token.TKN_IGUAL, yycolumn, yyline, yytext());}
<YYINITIAL>    "!=" {return new Symbol(Token.TKN_DIST, yycolumn, yyline, yytext());}
<YYINITIAL>    "and" {return new Symbol(Token.TKN_AND, yycolumn, yyline, yytext());}
<YYINITIAL>    "or" {return new Symbol(Token.TKN_OR, yycolumn, yyline, yytext());}
<YYINITIAL>    "'" {return new Symbol(Token.TKN_APOST, yycolumn, yyline, yytext());}
//<YYINITIAL>    "+" {return new Symbol(Token.TKN_PLUS, yycolumn, yyline, yytext());}
//<YYINITIAL>    "-" {return new Symbol(Token.TKN_MINUS, yycolumn, yyline, yytext());}
//<YYINITIAL>    "/" {return new Symbol(Token.TKN_DIVID, yycolumn, yyline, yytext());}

//----> Expresiones Regulares

<YYINITIAL>    {Letras} ({Letras}|{Digito})* {return new Symbol(Token.TKN_ID, yycolumn, yyline, yytext());}
<YYINITIAL>    [-+]?{Digito}+ {return new Symbol(Token.TKN_INT, yycolumn, yyline, yytext());}
```

```
//----> Error Lexico
   . {System.out.println("Error, el simbolo o sentencia "+yytext() +"no fue recnocido \n");
      Tabla_Error datos = new Tabla_Error(yytext(), yyline, yycolumn, "Error Lexico, ", "Simbolo no pertenece al lenguaje");
      Tabla_ERROR_LEXICO.add(datos);}
```

Gramática

```
package Analizador_Sintactico;

import java_cup.runtime.Symbol;
import java.util.ArrayList;
import java.util.LinkedList;

//------> Codigo contenedor del parser, metodos y variables
parser code
{:
   public static LinkedList<Tabla_Error> Tabla_ERROR_SINTAX = new LinkedList<Tabla_Error>();

   //Metodo llamado automaticamente al encontrar algun error Sintactico
   public void syntax_error(Symbol s)
   {
      String lexema = s.value.toString();
      int fila = s.right;
      int columna = s.left;

      System.out.println("Error Sintáctico, La Sentencia de Entrada no está completa");
      System.out.println("\tLexema: "+lexema);
      System.out.println("\tFila: "+fila);
      System.out.println("\tColumna: "+columna);

      Tabla_Error datos = new Tabla_Error(lexema, fila, columna, "Error Sintactico", "Caracter no esperado");

      /*-- Tabla_ERROR_SINTAX-> tabla de errores sintácticos --*/
      Tabla_ERROR_SINTAX.add(datos);
   }

   //Metodo llamado en el momento en el que ya no es posible la recuperacion de errores
   public void unrecovered_syntax_error(Symbol s) throws java.lang.Exception
   {
      String lexema = s.value.toString();
      int fila = s.right;
      int columna = s.left;
```

```
      Interface.Area_de_Resultado.setText("Resultado del Análisis: \n \n");
      Interface.Area_de_Resultado.append("Error Sintáctico, No es posible continuar con el Análisis! \n");
      Interface.Area_de_Resultado.append(" Hint:\n");
      Interface.Area_de_Resultado.append("\tLexema: "+ lexema +" \n");
      Interface.Area_de_Resultado.append("\tFila: "+ fila +" \n");
      Interface.Area_de_Resultado.append("\tColumna: "+ columna +" \n");

      Tabla_Error datos = new Tabla_Error(lexema, fila, columna, "Error Sintactico", "Caracter no
esperado");
      Tabla_ERROR_SINTAX.add(datos);

    }
:}

/*------------- Area2: Declaraciones -------------*/

//-----> Declaracion de TERMINALES

   //----> Terminales de Palabras Reservadas
      terminal TKN_CREATE, TKN_TABLE, TKN_SELECT, TKN_FROM, TKN_WHERE,
         TKN_GROUP, TKN_ORDER, TKN_BY, TKN_NOT, TKN_TODO, TKN_PRIMARY,
         TKN_KEY, TKN_NULL;

   //----> Terminales de Funciones de SQL

   terminal TKN_COUNT, TKN_AVG, TKN_SUM, TKN_MIN, TKN_MAX;

   //----> Terminales de Tipos de Datos
      terminal TKN_VARCHAR, TKN_CHAR, TKN_STRING, TKN_INTEG;
      //terminal TKN_FLOAT;

   //----> Terminales de Simbolos
      terminal TKN_APAR, TKN_CPAR, TKN_PTOCOMA, TKN_PTO, TKN_COMA, TKN_MEN,
         TKN_MAY, TKN_MAYIGU, TKN_MENIGU, TKN_IGUAL, TKN_DIST, TKN_AND, TKN_OR,
         TKN_APOST;
      //terminal TKN_PLUS, TKN_MINUS, TKN_DIVID;

   //----> Terminales de Expresiones Regulares
      terminal TKN_ID, TKN_INT;

//-----> Declaracion de NO TERMINALES

   //----> NO Terminales de Palabras Reservadas y Simbolos
      non terminal String N_TKN_APAR, N_TKN_CPAR, N_TKN_ID, N_TKN_INT, N_TKN_INTEG,
         N_TKN_STRING, N_TKN_COMA, N_TKN_WHERE, N_TKN_NOT, N_TKN_ORDER, N_TKN_GROUP,
         N_TKN_BY, N_TKN_IGUAL, N_TKN_APOST, N_TKN_SELECT, N_TKN_AND, N_TKN_OR,
N_TKN_MEN,
         N_TKN_MAY, N_TKN_MAYIGU, N_TKN_MENIGU, N_TKN_DIST, N_TKN_CREATE, N_TKN_TABLE,
```

```
        N_TKN_FROM, N_TKN_PTO, N_TKN_COUNT, N_TKN_AVG, N_TKN_SUM, N_TKN_MIN,
N_TKN_MAX,
        N_TKN_TODO, N_TKN_VARCHAR, N_TKN_CHAR, N_TKN_PRIMARY, N_TKN_KEY, N_TKN_NULL;

   //----> NO Terminales de referencia de estructura
      non terminal String INICIO, X, Cre, Y, P_c;
      non terminal String E_T, E_STR, E_NUM, E_VCH, E_CHR, S, Z, S_E, F, R_Col, R_Tab, A, B, C,
         W, O_B, G_B, Cond, R_Camp, N_AO, N_OP, Head, T_ASIG, T_INT, F_ESTRUCTURA, F_SQL
         , C_INT, C_ID, P_KEY, N_NULL;

//-----> PRECEDENCIAS

start with INICIO;

/*----------------- Area3: Reglas Semanticas ------------------*/

INICIO::= X;

//-----> Producciones NO TERMINALES de estructura

X::= Cre Y P_c | S P_c | S Z P_c | Cre Y P_c X | S P_c X | S Z P_c X;
   Cre::= N_TKN_CREATE N_TKN_TABLE N_TKN_ID;

   Y::= N_TKN_APAR E_T N_TKN_CPAR;
      E_T::= E_NUM | E_STR | E_VCH | E_CHR
         | E_NUM N_TKN_COMA E_T | E_NUM N_TKN_COMA P_KEY
         | E_STR N_TKN_COMA E_T | E_STR N_TKN_COMA P_KEY
         | E_VCH N_TKN_COMA E_T | E_VCH N_TKN_COMA P_KEY
         | E_CHR N_TKN_COMA E_T | E_CHR N_TKN_COMA P_KEY;

         E_NUM::= N_TKN_ID N_TKN_INTEG | N_TKN_ID N_TKN_INTEG C_INT
              | N_TKN_ID N_TKN_INTEG N_NULL | N_TKN_ID N_TKN_INTEG C_INT N_NULL;
         E_STR::= N_TKN_ID N_TKN_STRING | N_TKN_ID N_TKN_STRING C_INT
              | N_TKN_ID N_TKN_STRING N_NULL | N_TKN_ID N_TKN_STRING C_INT N_NULL;
         E_VCH::= N_TKN_ID N_TKN_VARCHAR C_INT
              | N_TKN_ID N_TKN_VARCHAR C_INT N_NULL;
         E_CHR::= N_TKN_ID N_TKN_CHAR C_INT | N_TKN_ID N_TKN_CHAR
              | N_TKN_ID N_TKN_CHAR C_INT N_NULL | N_TKN_ID N_TKN_CHAR N_NULL;

         C_INT::= N_TKN_APAR N_TKN_INT N_TKN_CPAR;
         N_NULL::= N_TKN_NOT N_TKN_NULL | N_TKN_NOT N_TKN_NULL N_TKN_PRIMARY N_TKN_KEY;
         P_KEY::= N_TKN_PRIMARY N_TKN_KEY N_TKN_APAR C_ID N_TKN_CPAR;
         C_ID::= N_TKN_ID | N_TKN_ID N_TKN_COMA C_ID;

   S::= S_E F ;
      S_E::= N_TKN_SELECT R_Tab | N_TKN_SELECT F_ESTRUCTURA ;
      R_Tab::= N_TKN_ID | N_TKN_TODO | N_TKN_ID N_TKN_PTO R_Tab | N_TKN_ID N_TKN_COMA
R_Tab;
```

```
    F_ESTRUCTURA::= F_SQL N_TKN_APAR N_TKN_ID N_TKN_CPAR;
    F::= N_TKN_FROM R_Col;
    R_Col::= N_TKN_ID | N_TKN_ID N_TKN_COMA R_Col ;
    F_SQL::= N_TKN_COUNT
          |N_TKN_AVG
          |N_TKN_SUM
          |N_TKN_MIN
          |N_TKN_MAX;


  Z::= A | B | C | A B | A C | B C | A B C;
    A::= W Cond;
    B::= O_B R_Col;
    C::= G_B R_Col;
    W::= N_TKN_WHERE | N_TKN_WHERE N_TKN_NOT;
    O_B::= N_TKN_ORDER N_TKN_BY;
    G_B::= N_TKN_GROUP N_TKN_BY;
    R_Camp::= N_TKN_ID | N_TKN_ID N_TKN_PTO R_Camp;

    Cond::= R_Camp N_OP R_Camp | R_Camp N_OP T_ASIG
       | R_Camp N_OP T_INT | T_INT N_OP R_Camp
       |  Cond N_AO N_TKN_APAR Cond N_TKN_CPAR
       |  Cond N_AO Head;

    Head::= R_Camp N_OP R_Camp | R_Camp N_OP T_ASIG
       |R_Camp N_OP T_INT | T_INT N_OP R_Camp;

    T_ASIG::= N_TKN_APOST N_TKN_ID N_TKN_APOST;
    T_INT::= N_TKN_INT;

    N_AO::= N_TKN_AND | N_TKN_OR;
    N_OP::= N_TKN_MEN
         |N_TKN_MAY
         |N_TKN_MAYIGU
         |N_TKN_MENIGU
         |N_TKN_IGUAL
         |N_TKN_DIST;


//-----> Producciones TERMINALES

  //-----> Producciones TERMINALES Palabras Reservadas

  N_TKN_CREATE::= TKN_CREATE {: Interface.Area_de_Resultado.append("CREATE "); :};
  N_TKN_TABLE::= TKN_TABLE {: Interface.Area_de_Resultado.append("TABLE "); :};
  N_TKN_SELECT::= TKN_SELECT {: Interface.Area_de_Resultado.append("SELECT "); :};
  N_TKN_FROM::= TKN_FROM {: Interface.Area_de_Resultado.append("\nFROM "); :};
  N_TKN_WHERE::= TKN_WHERE {: Interface.Area_de_Resultado.append("\nWHERE "); :};
```

```
N_TKN_GROUP::= TKN_GROUP {: Interface.Area_de_Resultado.append("\nGROUP "); :};
N_TKN_ORDER::= TKN_ORDER {: Interface.Area_de_Resultado.append("\nORDER "); :};
N_TKN_BY::= TKN_BY {: Interface.Area_de_Resultado.append("BY "); :};
N_TKN_NOT::= TKN_NOT {: Interface.Area_de_Resultado.append("NOT "); :};
N_TKN_TODO::= TKN_TODO {: Interface.Area_de_Resultado.append("TODO* "); :};
N_TKN_PRIMARY::= TKN_PRIMARY {: Interface.Area_de_Resultado.append("PRIMARY "); :};
N_TKN_KEY::= TKN_KEY {: Interface.Area_de_Resultado.append("KEY "); :};
N_TKN_NULL::= TKN_NULL {: Interface.Area_de_Resultado.append("NULL "); :};

//-----> Producciones TERMINALES Funciones SQL

N_TKN_COUNT::= TKN_COUNT {: Interface.Area_de_Resultado.append("COUNT "); :};
N_TKN_AVG::= TKN_AVG {: Interface.Area_de_Resultado.append("AVG "); :};
N_TKN_SUM::= TKN_SUM {: Interface.Area_de_Resultado.append("SUM "); :};
N_TKN_MIN::= TKN_MIN {: Interface.Area_de_Resultado.append("MIN "); :};
N_TKN_MAX::= TKN_MAX {: Interface.Area_de_Resultado.append("MAX "); :};

//-----> Producciones TERMINALES Tipos de Datos

N_TKN_VARCHAR::= TKN_VARCHAR {: Interface.Area_de_Resultado.append("VARCHAR "); :};
N_TKN_CHAR::= TKN_CHAR {: Interface.Area_de_Resultado.append("CHAR "); :};
N_TKN_STRING::= TKN_STRING {: Interface.Area_de_Resultado.append("STRING "); :};
N_TKN_INTEG::= TKN_INTEG {: Interface.Area_de_Resultado.append("INTEGER "); :};

//-----> Producciones TERMINALES Simbolos

N_TKN_APAR::= TKN_APAR {: Interface.Area_de_Resultado.append("A_PAR \n"); :};
N_TKN_CPAR::= TKN_CPAR {: Interface.Area_de_Resultado.append("\n" +"C_PAR "); :};
N_TKN_PTO::= TKN_PTO {: Interface.Area_de_Resultado.append("PTO "); :};
N_TKN_COMA::= TKN_COMA {: Interface.Area_de_Resultado.append("COMA "); :};
N_TKN_MEN::= TKN_MEN {: Interface.Area_de_Resultado.append("MEN "); :};
N_TKN_MAY::= TKN_MAY {: Interface.Area_de_Resultado.append("MAY "); :};
N_TKN_MAYIGU::= TKN_MAYIGU {: Interface.Area_de_Resultado.append("MAY_IGUAL "); :};
N_TKN_MENIGU::= TKN_MENIGU {: Interface.Area_de_Resultado.append("MEN_IGUAL "); :};
N_TKN_IGUAL::= TKN_IGUAL {: Interface.Area_de_Resultado.append("IGUAL "); :};
N_TKN_DIST::= TKN_DIST {: Interface.Area_de_Resultado.append("DISTINTO "); :};
N_TKN_AND::= TKN_AND {: Interface.Area_de_Resultado.append("AND "); :};
N_TKN_OR::= TKN_OR {: Interface.Area_de_Resultado.append("OR "); :};

N_TKN_APOST::= TKN_APOST {: Interface.Area_de_Resultado.append("APOST "); :};

P_c::= TKN_PTOCOMA {: Interface.Area_de_Resultado.append("PTO_COMA" + "\n \n ** La Sentencia
de entrada ha sido aceptada ** \n \n" ); :};

//-----> Producciones TERMINALE Expresiones Regulares
N_TKN_ID::= TKN_ID {: Interface.Area_de_Resultado.append("'ID' "); :};
N_TKN_INT::= TKN_INT {: Interface.Area_de_Resultado.append("'INT' "); :};
```