Manual Técnico

Introducción:
El siguiente documento presenta la descripción del proyecto realizado como práctica 1 para el curso de Organización de lenguajes y compiladores. La práctica consiste en el manejo léxico, sintáctico y semántico de dos lenguajes. El primero es el lenguaje para la creación de base de datos y el segundo es para la creación del lenguaje llamado MiniSQL. Ambos serán descritos a continuación así como el entorno de desarrollo que en esta ocasión fue Angular con la ayuda de jison para la creación de los analizadores utilizados.

Plataforma de desarrollo:
- Angular 15.2.5
- IntelliJ (o WebStorm de Jetbrains).
- Jison.

Requisitos:

- Angular CLI.
- Dependencias NPM.
- Su navegador de preferencia.

Estructura del análisis léxico para la base de datos.

```
lineTerminator      \r|\n|\r\n
whitespace          {lineTerminator}|[ \t\f]
INT                 "INT"
DECIMAL             "DECIMAL"
STRING              "STRING"
BOOLEAN             "BOOLEAN"
TRUE                "true"|"TRUE"
FALSE               "false"|"FALSE"
INTEGER             [0]|[1-9][0-9]*
DOUBLE              [0-9]+"."[0-9]+
COMMA               ","
SEMICOLON           ";"
COMMENT             \#[^\r\n]*
LPAREN              "("
RPAREN              ")"
PLUS                "+"
MINUS               "-"
TIMES               "*"
DIVIDE              "/"
EQUALS              "=="
LESS_THAN           "<"
GREATER_THAN        ">"
LESS_EQUALS         "<="
GREATER_EQUALS      ">="
NOT_EQUALS          "!="
ASSIGN              "="
```

```
NOT                 "!"
AND                 "&&"
OR                  "||"
STRING_VALUE        (\"[^\"]*\") | (\'[^\']*\')
NAME                ([a-zA-Z])[a-zA-Z0-9_]*
```

```
{whitespace}        {}
{COMMENT}           {}
{INT}               return "INT";
{DECIMAL}           return "DECIMAL";
{STRING}            return "STRING";
{BOOLEAN}           return "BOOLEAN";
{TRUE}              return "TRUE";
{FALSE}             return "FALSE";
{DOUBLE}            return "DOUBLE";
{INTEGER}           return "INTEGER";
{COMMA}             return "COMMA";
{SEMICOLON}         return "SEMICOLON";
{LPAREN}            return "LPAREN";
{RPAREN}            return "RPAREN";
{PLUS}              return "PLUS";
{MINUS}             return "MINUS";
{TIMES}             return "TIMES";
{DIVIDE}            return "DIVIDE";
{EQUALS}            return "EQUALS";
{LESS_THAN}         return "LESS_THAN";
{GREATER_THAN}      return "GREATER_THAN";
{LESS_EQUALS}       return "LESS_EQUALS";
{GREATER_EQUALS}    return "GREATER_EQUALS";
{NOT_EQUALS}        return "NOT_EQUALS";
{ASSIGN}            return "ASSIGN";
{NOT}               return "NOT";
{AND}               return "AND";
{OR}                return "OR";
{STRING_VALUE}      {yytext = yytext.substr(1,yyleng-2);return
"STRING_VALUE";}
{NAME}              {return "NAME";}
<<EOF>>             {return "EOF";}
.                   {retorna un error léxico en el programa}
```

Estructura del análisis sintáctico para la base de datos:
Abajo se muestran las producciones usadas en jison para el lenguaje de base de datos. Hay qu eaclarar que lo que está escrito en minúsculas son los no terminales usados en las producciones y las palabras en mayúsculas son las que retorna la parte léxica de jison. Por simplicidad no se muestran las operaciones realizadas en cada producción pero eso puede verse directamente en el documento dbParser.jison en el proyecto.

```
tableStatement
 : NAME LPAREN properties RPAREN
 ;

properties
 : properties COMMA property
 | property
 ;

property
 : NAME type
 ;

type
 : INT
 | DECIMAL
 | STRING
 | BOOLEAN
 ;

rowStatement
 : rowStatement COMMA attribute
 | attribute
 ;

attribute
 : NAME ASSIGN number
 | NAME ASSIGN STRING_VALUE
 | NAME ASSIGN booleanResult
 ;

booleanResult
 : booleanResult OR e
 | e
 ;

e
 : e AND f
 | f
 ;

f
 : n g
```

```
 | g
 ;

n
 : n NOT
 | NOT
 ;

g
 : number EQUALS number
 | number NOT_EQUALS number
 | number LESS_THAN number
 | number LESS_EQUALS number
 | number GREATER_THAN number
 | number GREATER_EQUALS number
 | h
 ;

h
 : TRUE
 | FALSE
 | LPAREN booleanResult RPAREN
 ;

number
 : number PLUS b
 | number MINUS b
 | b
 ;

b
 : b TIMES c
 | b DIVIDE c
 | c
 ;

c
 : MINUS d
 | d
 ;

d
: INTEGER
| DOUBLE
|LPAREN number RPAREN
;
```

Análisis léxico del lenguaje MiniSQL:

```
lineTerminator      \r|\n|\r\n
whitespace          {lineTerminator}|[ \t\f]
DECLARE             "DECLARE"
INT                 "INT"
DECIMAL             "DECIMAL"
BOOLEAN             "BOOLEAN"
TEXT                "TEXT"
INTEGER             [0-9]+
DOUBLE              [0-9]+"."[0-9]+
TRUE                "TRUE"
FALSE               "FALSE"
AS                  "AS"
SET                 "SET"
AND                 "AND"
INPUT               "INPUT"
PRINT               "PRINT"
IF                  "IF"
ELSE                "ELSE"
ELSEIF              "ELSEIF"
END                 "END"
THEN                "THEN"
NOT                 "NOT"
AND                 "AND"
OR                  "OR"
SELECT              "SELECT"
FROM                "FROM"
WHERE               "WHERE"
LIMIT               "LIMIT"
OFFSET              "OFFSET"
ID                  [@][a-zA-Z][a-zA-Z0-9_]*
COMMENT             ("--")[^\r\n]*
COMMA               ","
SEMICOLON           ";"
LPAREN              "("
RPAREN              ")"
PLUS                "+"
MINUS               "-"
TIMES               "*"
DIVIDE              "/"
EQUALS              "="
LESS_THAN           "<"
GREATER_THAN        ">"
LESS_EQUALS         "<="
GREATER_EQUALS      ">="
NOT_EQUALS          "<>"
TEXT_VALUE          (\"[^\"]*\") | (\'[^\']*\')
NAME                ([a-zA-Z])[a-zA-Z0-9_]*
```

```
{whitespace}        {}
{COMMENT}           {}
{DECLARE}           {return "DECLARE";}
{INT}               {return "INT";}
{DECIMAL}           return "DECIMAL";
{TEXT}              return "TEXT";
{BOOLEAN}           return "BOOLEAN";
{TRUE}              return "TRUE";
{FALSE}             return "FALSE";
{DOUBLE}            {return "DOUBLE";}
{INTEGER}           {return "INTEGER";}
{COMMA}             return "COMMA";
{SEMICOLON}         return "SEMICOLON";
{AS}                return "AS";
{SET}               return "SET";
{AND}               return "AND";
{INPUT}             return "INPUT";
{PRINT}             return "PRINT";
{IF}                return "IF";
{ELSEIF}            return "ELSEIF";
{ELSE}              return "ELSE";
{END}               return "END";
{THEN}              return "THEN";
{SELECT}            return "SELECT";
{FROM}              return "FROM";
{WHERE}             return "WHERE";
{LIMIT}             return "LIMIT";
{OFFSET}            return "OFFSET";
{LPAREN}            return "LPAREN";
{RPAREN}            return "RPAREN";
{PLUS}              return "PLUS";
{MINUS}             return "MINUS";
{TIMES}             return "TIMES";
{DIVIDE}            return "DIVIDE";
{NOT_EQUALS}        return "NOT_EQUALS";
{GREATER_EQUALS}    return "GREATER_EQUALS";
{LESS_EQUALS}       return "LESS_EQUALS";
{LESS_THAN}         return "LESS_THAN";
{GREATER_THAN}      return "GREATER_THAN";
{EQUALS}            return "EQUALS";
{NOT}               return "NOT";
{AND}               return "AND";
{OR}                return "OR";
{ID}                return "ID";
{TEXT_VALUE}        {yytext = yytext.substr(1,yyleng-2);return
"TEXT_VALUE";}
{NAME}              {return "NAME";}
<<EOF>>             return "EOF";
.                   {Muestra un error léxico en el análisis}
```

Análisis sintáctico del lenguaje MiniSQL:
Al igual que con el lenguaje anterior, se usan las palabras en minúsculas para las producciones no terminales y las mayúsculas para los terminales. La creación del árbol AST no se muestra en este documento, sin embargo, en el documento miniSQLParser.jison puede ver claramente cómo es la creación de este.

```
main
 : assignments statements EOF
 | error
 ;

assignments
 : assignments assignment
 |
 ;

statements
 : statements statement
 |
 ;

assignment
 : DECLARE ids AS type SEMICOLON
 | DECLARE ids AS type EQUALS value SEMICOLON
;

ids
 : ids COMMA ID
 | ID
 ;

type
 : INT
 | DECIMAL
 | TEXT
 | BOOLEAN
 ;

statement
 : setStatement SEMICOLON
 | ifStatement SEMICOLON
 | printStatement SEMICOLON
 | selectStatement SEMICOLON
 ;

setStatement
 : SET idsAssignment
 ;

idsAssignment
```

```
 : idsAssignment COMMA idAssignment
 | idAssignment
 ;

idAssignment
 : ID EQUALS value
 | ID EQUALS inputProd
 ;

inputProd
 :INPUT LPAREN TEXT_VALUE RPAREN
;

ifStatement
 : IF value THEN statements END IF
 | IF value THEN statements elseIfStatements END IF
 ;

elseIfStatements
 : ELSEIF value THEN statements elseIfStatements
 | ELSEIF value THEN statements
 | ELSE statements

 ;

printStatement
 : PRINT LPAREN content RPAREN
 ;

content
 : content COMMA value
 | value
 ;

value
 : value OR e
 | e
 ;

e
 : e AND f
 | f
 ;

f
 : NOT g
 | g
 ;

g
 : g EQUALS h
```

```
 | g NOT_EQUALS h
 | g LESS_THAN h
 | g GREATER_THAN h
 | g LESS_EQUALS h
 | g GREATER_EQUALS h
 | h
 ;

h
 : number
 | TEXT_VALUE
 | TRUE
 | FALSE
 ;


selectStatement
 : SELECT properties FROM NAME
 | SELECT properties FROM NAME whereProd
 | SELECT properties FROM NAME limitProd
 | SELECT properties FROM NAME offSetProd

 ;

whereProd
 : WHERE whereValue
 | WHERE whereValue limitProd
 | WHERE whereValue offSetProd
 ;

limitProd
 : LIMIT number
 | LIMIT number offSetProd
 ;

offSetProd
 : OFFSET number
 ;


properties
 : TIMES
 | propertyNames
 ;

propertyNames
 : propertyNames COMMA NAME
 | NAME
 ;

number
```

```
 : number PLUS b
 | number MINUS b
 | b
 ;

b
 : b TIMES c
 | b DIVIDE c
 | c
 ;

c
 : MINUS d
 | d
 ;

d
 : INTEGER
 | DOUBLE
 | ID

 | LPAREN value RPAREN
 ;

whereValue
 : whereValue OR i
 | i
 ;

i
 : i AND j
 | j
 ;

j
 : NOT k
 | k
 ;

k
 : k EQUALS m
 | k NOT_EQUALS m
 | k LESS_THAN m
 | k GREATER_THAN m
 | k LESS_EQUALS m
 | k GREATER_EQUALS m
 | m
 ;

m
 : whereNumber
```

```
 | TEXT_VALUE
 | TRUE
 | FALSE
 | NAME
 ;

whereNumber
 : whereNumber PLUS x
 | whereNumber MINUS x
 | x
 ;

x
 : x TIMES y
 | x DIVIDE y
 | y
 ;

y
 : MINUS z
 | z
 ;

z
 : INTEGER
 | DOUBLE
 | ID
 | LPAREN whereValue RPAREN
 ;
```