Universidad San Carlos de Guatemala
Facultad de ingeniería
Escuela de ciencias y sistemas
Segundo semestre 2023
Organización de lenguajes y compiladores 1

QueryCrypter

Proyecto 2

Manual Técnico

Evelyn Marisol Pumay Soy

Carnet: 202112395

25/10/2023

El proyecto consiste en un intérprete que es capaz de reconocer instrucciones en lenguaje SQL y las sentencias básicas de DDL y DML para el manejo de tablas en memoria. También permite el uso de distintos tipos de datos en las expresiones y el manejo correcto de la precedencia de las operaciones de dichas expresiones..

Herramientas Utilizadas

Jison- Análisis léxico y sintactico
Java Script- interprete
HTML y CSS- Frontend

Traduccion Statpy

Para realizar la lectura de las sentencias se realizo un analizador Léxico y sintactico con la herramienta Jison

GRAMÁTICA

<ini> ::= <instructions> EOF

<instructions> ::= <instructions> <instruction>
| <instruction>
| DECLARE <declarations> PTCOMA

<instruction> ::= RIMPRIMIR PARIZQ <expression_cadena> PARDER PTCOMA
| IDENTIFICADOR IGUAL <expression_cadena> PTCOMA
| RIF PARIZQ <expression_logica> PARDER LLAVIZQ <instructions> LLAVDER
| SET ARROBA IDENTIFICADOR IGUAL <expression_cadena> PTCOMA
| CREATE TABLE IDENTIFICADOR PARIZQ <datos_tabla> PARDER PTCOMA
| ALTER TABLE IDENTIFICADOR RENAME TO IDENTIFICADOR PTCOMA
| ALTER TABLE IDENTIFICADOR ADD IDENTIFICADOR <tipo_dato> PTCOMA
| ALTER TABLE IDENTIFICADOR DROP COLUMN IDENTIFICADOR PTCOMA
| ALTER TABLE IDENTIFICADOR RENAME COLUMN IDENTIFICADOR TO IDENTIFICADOR PTCOMA
| DROP TABLE IDENTIFICADOR PTCOMA
| SELECT <valores_varios> FROM IDENTIFICADOR PTCOMA
| SELECT FROM POR IDENTIFICADOR PTCOMA
| INSERT INTO IDENTIFICADOR PARIZQ <valores_varios> PARDER VALUES PARIZQ <valores_tabla> PARDER PTCOMA

```
        | UPDATE IDENTIFICADOR SET IDENTIFICADOR IGUAL
<expression_cadena> WHERE IDENTIFICADOR IGUAL <expression_cadena>
        | TRUNCATE TABLE IDENTIFICADOR
        | error

<datos_tabla> ::= <datos_tabla> COMA <dato_tabla>
        | <dato_tabla>

<dato_tabla> ::= IDENTIFICADOR <tipo_dato>

<valores_varios> ::= <valores_varios> COMA IDENTIFICADOR
          | IDENTIFICADOR

<valores_tabla> ::= <valores_tabla> COMA <expression_cadena>
          | <expression_cadena>

<declarations> ::= <declarations> COMA <declaration>
          | <declaration>

<declaration> ::= ARROBA IDENTIFICADOR <tipo_dato>
        | ARROBA IDENTIFICADOR <tipo_dato> DEFAULT <expression_cadena>

<tipo_dato> ::= INT
        | VARCHAR
        | DATE
        | BOOLEAN
        | DOUBLE

<expression_numerica> ::= MENOS <expression_numerica> %prec UMENOS
            | <expression_numerica> MAS <expression_numerica>
            | <expression_numerica> MENOS <expression_numerica>
            | <expression_numerica> POR <expression_numerica>
            | <expression_numerica> DIVIDIDO <expression_numerica>
            | PARIZQ <expression_numerica> PARDER
            | ENTERO
            | DECIMAL
            | IDENTIFICADOR

<expression_cadena> ::= <expression_cadena> CONCAT <expression_cadena>
            | CADENA
            | <expression_numerica>

<expression_relacional> ::= <expression_numerica> MAYQUE <expression_numerica>
              | <expression_numerica> MENQUE <expression_numerica>
              | <expression_numerica> MAYIGQUE <expression_numerica>
              | <expression_numerica> MENIGQUE <expression_numerica>
```

| <expression_cadena> DOBLEIG <expression_cadena>
| <expression_cadena> NOIG <expression_cadena>

<expression_logica> ::= <expression_relacional> AND <expression_relacional>
| <expression_relacional> OR <expression_relacional>
| NOT <expression_relacional>
| <expression_relacional>

Manejo de errores:

Los manejos de errores se realizaron directamente desde cada uno de los analizadores
Analizador lexico

```
<<EOF>>             return 'EOF';
.                   { console.error('error léxico: ' + yytext + ', en la linea: ' + yylloc.first_line + ', en la columna: '
```

Analizador sintactico:

```
| error { console.error('error sintáctico: ' + yytext + ', en la linea: ' + this._$.first_line + ', en la columna:
```

Interprete:

Para el interprete se crearon variables para cada instrucción:

```
const TIPO_INSTRUCCION = {
    IMPRIMIR:        'INSTR_IMPRIMIR',
    DECLARACION:     'INSTR_DECLARACION',
    ASIGNACION:      'INSTR_ASIGANCION',
    DECLARACIONYA:       'INSTR_DECLARACIONYA',
    IF:              'INSTR_IF',
    IF_ELSE:         'INSTR_ELSE',
    CREATE_TABLE:    'INSTR_CREATE_TABLE',
    MNAME_TABLE:     'INSTR_MNAME_TABLE',
    ADDCOLUMN_TABLE:     'INSTR_ADDCOLUMN_TABLE',
    DELETECOLUMN_TABLE: 'INSTR_DELETECOLUMN_TABLE',
    NEWCOLUMNNAME_TABLE:     'INSTR_NEWCOLUMNNAME_TABLE',
    DELETE_TABLE:    'INSTR_DELETE_TABLE',
    INSERT_TABLEDATA:    'INSTR_INSERT_TABLEDATA',
    SELECT_COLUMNAS:     'INSTR_SELECT_COLUMNAS',
    SELECT_ALL:      'INSTR_SELECT_ALL',
    TRUNCATE_TABLE: 'INSTR_TRUNCATE_TABLE'
}
```

Y una serie de funciones que retornaban los valores necesarios para manejar cada sentencia:


```
const instruccionesAPI = {

    nuevoOperacionBinaria: function(operandoIzq, operandoDer, tipo) {
        return nuevaOperacion(operandoIzq, operandoDer, tipo);
    },

    nuevoOperacionUnaria: function(operando, tipo) {
        return nuevaOperacion(operando, undefined, tipo);
    },


    nuevoValor: function(valor, tipo) {
        return {
            tipo: tipo,
            valor: valor
        }
    },


    nuevoImprimir: function(expresionCadena) {
        return {
            tipo: TIPO_INSTRUCCION.IMPRIMIR,
            expresionCadena: expresionCadena
        };
    },
```

Luego en el archivo interprete se realizaba la llamada a dichas funciones y se procesaban sus distintas funciones.

```
exports.procesarBloque=function(instrucciones, tablaDeSimbolos) {
    let results=''
    instrucciones.forEach(instruccion => {

        if (instruccion.tipo === TIPO_INSTRUCCION.IMPRIMIR) {
            // Procesando Instrucción Imprimir

            if(procesarImprimir(instruccion, tablaDeSimbolos)==undefined){
               results+='';
            }else{
               results +=procesarImprimir(instruccion, tablaDeSimbolos) +'\n';
            }
        } else if (instruccion.tipo === TIPO_INSTRUCCION.DECLARACION) {
            // Procesando Instrucción Declaración
            procesarDeclaracion(instruccion, tablaDeSimbolos);
        } else if (instruccion.tipo === TIPO_INSTRUCCION.ASIGNACION) {
            // Procesando Instrucción Asignación
            procesarAsignacion(instruccion, tablaDeSimbolos);
        } else if (instruccion.tipo === TIPO_INSTRUCCION.DECLARACIONYA) {
            // Procesando Instrucción Declaración y Asignación
            procesarDeclaracionYA(instruccion, tablaDeSimbolos);
        } else if(instruccion.tipo===TIPO_INSTRUCCION.CREATE_TABLE){
            // Procesando Instrucción Crear Tabla
            procesarCreateTable(instruccion, tablaDeSimbolos);
```

Manejo de tabla de símbolos:

Manejo de almacenamiento de tablas: