

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



**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 línea: ' + yylloc.first_line + ', en la columna: '

```

Analizador sintactico:

```

| error { console.error('error sintáctico: ' + yytext + ', en la línea: ' + 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',
  DELETETABLE:   'INSTR_DELETETABLE',
  NEWCOLUMNNAME_TABLE: 'INSTR_NEWCOLUMNNAME_TABLE',
  DELETE_TABLE:  'INSTR_DELETE_TABLE',
  INSERT_TABLEDATA: 'INSTR_INSERT_TABLEDATA',
  SELECT_COLUMNS: 'INSTR_SELECT_COLUMNS',
  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:

```
class TS {  
    /**  
     * El constructor recibe como parámetro los símbolos de la tabla padre.  
     * @param {*} simbolos  
     */  
    constructor (simbolos) {  
        this._simbolos = simbolos;  
    }  
  
    agregar(id, tipo) {  
        const nuevoSimbolo = crearSimbolo(id, tipo);  
        this._simbolos.push(nuevoSimbolo);  
    }  
  
    agregarValor(id, tipo, valor) {  
        const nuevoSimbolo = crearSimbolo(id, tipo, valor);  
        this._simbolos.push(nuevoSimbolo);  
    }  
  
    actualizar(id, valor) {  
        const simbolo = this._simbolos.filter(simbolo => simbolo.id === id)[0];  
        if (simbolo) {  
            simbolo.valor = valor.valor;  
        }  
        else {  
            throw 'ERROR: variable con id: ' + id + ' no ha sido definida';  
        }  
    }  
}
```

Manejo de almacenamiento de tablas:

```
class Tabla {  
    constructor(nombreTabla) {  
        this.nombre = nombreTabla;  
        this.columnas = [];  
    }  
  
    // Método para agregar una columna a la tabla  
    agregarColumna(nombreColumna, tipoDato) {  
        this.columnas.push({ nombre: nombreColumna, tipo: tipoDato, valores: [] });  
    }  
  
    // Método para imprimir la estructura de la tabla  
    imprimirEstructura() {  
        console.log(`Datos de la tabla "${this.nombre}":`);  
  
        // Imprimir encabezados de las columnas  
        const encabezados = this.columnas.map((columna) => columna.nombre);  
        console.log(encabezados.join('\t'));  
  
        // Obtener el número de registros (la cantidad de valores en una columna)  
        const numeroRegistros = this.columnas.length > 0 ? this.columnas[0].valores.length : 0;  
  
        // Imprimir los valores de cada columna  
        for (let i = 0; i < numeroRegistros; i++) {  
            const fila = this.columnas.map((columna) => columna.valores[i]);  
            console.log(fila.join('\t'));  
        }  
    }  
}
```