TRADUCTORES, COMPILADORES E INTÉRPRETES

Convocatoria extraordinaria de diciembre de 1998

Ingeniería Técnica de informática de Sistemas. 3er curso.

Se pretende realizar el análisis sintáctico y semántico de dos instrucciones fundamentales del lenguaje SQL (*Structured Query Language*). Este lenguaje permite definir las tablas de las bases de datos relacionales, así como manipularlas.

En concreto, queremos gestionar la sentencia CREATE TABLE ..., y la sentencia SELECT FROM ... La sentencia CREATE TABLE ... tiene el siguiente formato:

CREATE TABLE nombreTabla (
nombreCampo1 Tipo [UNIQUE] [NOT NULL],
nombreCampo2 Tipo [UNIQUE] [NOT NULL], etc.
);

Esta sentencia crea una tabla que contiene los campos nombreCampo1, nombreCampo2, etc., y asocia a cada uno de ellos los tipos correspondientes. Asimismo, cada campo puede tener restricciones opcionales:

- UNIQUE, que indica que no puede haber dos valores iguales para ese campo.
- NOT NULL, que indica que ese campo debe tener obligatoriamente algún valor.

La sentencia SELECT FROM ... tiene el siguiente formato:

SELECT campo1, campo2, etc.
FROM tabla1, tabla2, etc.
[WHERE condición];

Esta sentencia efectúa una consulta sobre las tablas tabla1, tabla2, etc., seleccionando sus campos campo1, campo2, etc. Sólo se consultarán aquellas tuplas que cumplan la condición opcional que se indica en la parte WHERE. Esta condición puede usar los operadores lógicos AND, OR y NOT, y los relacionales =, < y LIKE (éste último sólo para elementos de tipo Text).

Opcionalmente, cada nombre de campo puede estar precedido del nombre de la tabla de la que proviene, con objeto de evitar ambigüedades.

Se pide (como parte teórica), dibujar los diagramas de Conway que reconocen la gramática pedida.

Como parte práctica, se piden los programas Lex y Yacc que reconozcan la grámatica con las siguientes características sintáctico/semánticas:

- Toda sentencia acaba obligatoriamente en punto y coma.
- Un programa puede no tener sentencia alguna.
- Los nombres de campos y de tablas tienen el formato usual de identificador.
- Se permiten los operadores + y * para el tipo Integer, y sólo el + para el tipo Text. No se permiten expresiones en las que se mezclen ambos tipos.
- El operador relacional LIKE sólo se permite entre expresiones de tipo texto.
- No puede haber dos tablas con el mismo nombre.
- En una misma tabla no puede haber dos campos con el mismo nombre.
- En un SELECT FROM ...:

- En la parte SELECT no puede haber dos campos con el mismo nombre, a no ser que provengan de tablas distintas.
- Opcionalmente, cada nombre de campo (tanto en la parte SELECT como en la WHERE) puede estar precedido del nombre de la tabla de la que proviene.
- En la parte SELECT sólo puede aparecer nombres de campos que existen en la tablas indicadas en la parte FROM.
- En la condición del WHERE sólo puede aparecer nombres de campos que existen en la tablas indicadas en la parte FROM.
- No puede haber ambigüedad, esto es, si se utiliza un nombre de campo que aparece en dos tablas de la parte FROM, dicho nombre de campo debe estar precedido de la tabla de la que debe provenir.

Para facilitar la construcción de este analizador, se desea usar una estructura auxiliar, con su correspondiente interfaz:

Fichero AUXSIM98.C

```
/* Variables auxiliares */
Tabla codigoTabla;
int longitudListaTabla;
/* Permite almacenar temporalmente la lista de tablas de un SELECT */
Tabla listaTabla[10];
/* Inserta una tabla en la lista temporal */
void insertarListaTabla(char * nombre) {

}
/*Chequea si existe un campo en la lista temporal de tablas.
   Si no existe, saca mensaje de error, y si hay ambigüedad también */
Tipo campoExiste(char * nombreCampo) {
```

```
}
/* Chequea si cierta tabla posee un campo concreto o no, emitiendo errores si los hay */
Tipo tablaYCampoExiste(char * nombreTabla, char * nombreCampo) {

}
```

Además, se suministra una tabla de símbolos con la siguiente interfaz:

Fichero TABSIM98.C

```
/* Tipos definidos */
typedef enum {Integer, Texto, TipoNulo} Tipo;
typedef enum  {Notnull, Unique, NotnullUnique, RestriccionNula} Restriccion;
typedef int Tabla;
typedef int Campo;
/* Inserta una tabla por su nombre */
Tabla insertarTabla(char * nombre);
/* Busca una tabla por su nombre */
Tabla buscarTabla(char * nombre);
/* Devuelve el nombre de una tabla */
char * nombreTabla(Tabla i);
/* Busca un campo por su nombre, en una tabla concreta */
Campo buscarCampo(Tabla t, char * nombre);
/*Inserta un campo nuevo (junto con su información) en una tabla */
Campo insertarCampo(Tabla t, char * nombre, Tipo p, Restriccion r);
/* Devuelve el tipo de un campo */
Tipo buscarTipo(Tabla i, Campo p) ;
```

/* *Devuelve las restricciones de un campo* */
Restriccion buscarRestriccion(Tabla i, Campo p) ;

Y el siguiente esqueleto:
Fichero EXADIC98L.LEX
%%

Fichero EXADIC98Y.YAC
%{

%}

%union {

}

```
%%
programa :      /* epsilon */
        |       programa instruccion ';'
        |       programa error  ';'      {                               }
        ;
instruccion :   create
          |     select
          ;
create :        CREATE TABLE ID '('  {
                                        }
                lista_def_id
                ')'
      ;
lista_def_id  : def_id
              |     lista_def_id def_id
              ;
def_id   : ID tipo opcional       {
                                        }
        ;
opcional :      NOT NULL                {                               }
        |       UNIQUE                  {                               }
        |       UNIQUE NOT NULL         {                               }
        |       NOT NULL UNIQUE         {                               }
        |       /* epsilon */           {                               }
        ;
tipo :   INTEGER                {                       }
    |    TEXTO                  {                       }
    ;
select :        SELECT bloque WHERE condicion
       |        SELECT bloque
       ;
bloque :        ID ',' bloque           {                                       }
```

```
      |            ID '.' ID ',' bloque           {                                        }
      |            ID FROM lista_tabla            {                                        }
      |            ID '.' ID FROM lista_tabla     {                                        }
      ;
lista_tabla :      {                              }        tabla
          |        lista_tabla tabla
          ;
tabla  : ID                  {



                          }
      ;
condicion :        condicion OR condicion
        |          condicion AND condicion
        |          NOT condicion
        |          expr '=' expr       {

                          }
        |          expr '<' expr       {

                          }
        |          expr LIKE expr      {

                          }
        |          '(' condicion ')'
        ;
expr : campo                      {                        }
    | NUMERO                      {                        }
    | CADENA                      {                        }
    | expr '+' expr            {



                          }
    | expr '*' expr            {


```

```
                              }
    | '(' expr ')'                    {                        }
    ;
campo :         ID                        {                              }
     |          ID '.' ID                 {                              }
     ;
%%
```