Manual Técnico

XSQL

ORGANIZACION DE LENGUAJES Y COMPILADORES 2

EDV DICIEMBRE 2023

Diego Estrada | William Miranda

XSQL

A continuación se presenta la abstracción de un sistema administrador de base de datos, el cual es capaz de manejar las instrucciones básicas de un DBMS relacional convencional.

Se presenta la abtracción de un IDE con el que el usuario interactúa directamente con el sistema de base de datos.

Dicho IDE cuenta con un conjunto de herramientas básicas que permiten el uso fácil de la herramienta.

Para almacenar y manipular la información de las bases de datos se utilizó un sistema de archivos con formato XML, con la estructura de las bases de datos creadas, especificando las tablas, objetos y procedimientos de las mismas.

El objetivo principal de la realización del mismo es la implementación de un interprete XML para leer la información guardada en los archivos.

Índice

Abstracción

Gramática

Modelos

Estructuras XML

01
Abstracción

Abstracción de Tokens

PALABRAS RESERVADAS ENCONTRADAS:

CREATE, USE, ALTER
DROP, TRUNCATE, SELECT
FROM, WHERE, UPDATE
INSERT, INTO, VALUES
DELETE, TABLE, DATA
BASE, ADD, IF
FUNCTION, PROCEDURE, AS
NOT, NULL, PRIMARY
FOREING, KEY, REFERENCE
int, bit, date
datetime, BEGIN, DECLARE
SET, RETURN, END
CONCATENA, SUBSTRAER, HOY
CONTAR, SUMA, CAS
NCHAR, NVARCHAR

TOKENS

- IDENTIFICADORES
- NUMEROS ENTEROS
- NUMEROS DECIMALES
- OPERADORES ARITMETICOS:
+ - * /
- OPERADORES LÓGICOS
< > <= >= !=

ASPECTOS A TOMAR EN CUENTA:

-. Para nombrar a una variable, se le antepone el @ en su nombre de variable, por ejemplo @variable1, @mi_edad sería como @<identificador>

-. Tipos de datos para las variables:
int
bit
decimal
date
datetime
Nchar
Nvarchar

-. Para las funciones solo se permite retornar valores numericos enteros.

-. Declaracion de variable: DECLARE @<nombre_variable> AS <tipo_de_dato>. Las variables solo se pueden declarar dentro de una funcion y como parametros de una funcion.

-. Asignacion de valor a variable: SET @<nombre_variable> = <expresion>

Abstracción Expresiones Regulares

PALABRAS RESERVADAS ENCONTRADAS:

ID = @ [a-zA-Z][a-zA-Z0-1]*
entero = [0-9]+
decimal = [0-9]+\.[0-9]+
SEMICOLON = ';'
NOMBRE = [a-zA-Z]+
L_PAREN = '('
R_PAREN = ')'
COMMA = ','
ASSIGN = '='
EQUALS = '=='
NOT_EQ = '!='
LESS_THAN = '<'
GREATER_THAN = '>'
LESS_EQ = '<='
GREATER_EQ = '>='
AND = \&\&
OR = \|\|
NOT = \!
PLUS = '+'
MINUS = '-'
TIMES = '*'
DIVIDE = '/'

Abstracción de Funciones

FUNCIONES PROPIAS DEL SISTEMA:

CONCATENA: recibe dos parametros de tipo nchar o nvarchar. El valor a retornar es la concatenación entre ambas cadenas, por lo que el tipo de dato a retornar es nvarchar

SUBSTRAER: Extrae porciones de texto de una cadena, necesita 3 parametros: el texto, la posicion desde donde quiere substraer, la posicion final donde se desee finalizar la substracción. Devuelve el texto subtraído y el tipo de dato es nvarchar

HOY: Devuelve la fecha y hora actual del sistema, tipo de dato a devolver es datatime.

CONTAR: Devuelve la cantidad de filas que contiene una tabla, es decir, la cantidad de registros que contiene una tabla según las condiciones que se le agreguen.
El tipo de dato que devuelve es un int.

SUMA : Devuelve la suma de los valores de una columna numérica, es decir, la columna debe contener datos numericos, y la funcion suma todos esos valores de la columna
y el tipo de dato que devuelve es DECIMAL.

CAS : Castea el valor a evaluar.

02
Gramática

PLY

¿Qué es PLY?

PLY se refiere a un módulo de Python llamado "PLY" que se utiliza para construir analizadores léxicos y sintácticos (lexers y parsers). Este módulo proporciona herramientas para implementar analizadores basados en las herramientas Lex y Yacc, que son comunes en la construcción de compiladores.

Dicho módulo fue utilizado para la implementación de los analizadores de XSQL.

A continuación se detalla la el archivo grammar.py, encargado de la declaración del analizador léxico y sintáctico.

```
import re

from src.error.xsql_error import xsql_error

global_arr = []
errores_sintacticos = []
matriz_resultante = []
```

SE IMPORTAN LAS LIBRERÍAS CORRES PONDIENTES PARA UTILIZAR PLY Y SE DECLARAN LOS ARREGLOS DE ALMACENIAMIENTO DE DATOS

PLY

```
ireservadas = {
   'create': 'CREATE',   'use': 'USE',   'alter': 'ALTER',
   'drop': 'DROP',   'case': 'CASE',   'column': 'COLUMN',
   'truncate': 'TRUNCATE',   'exec': 'EXEC',   'select': 'SELECT',
   'from': 'FROM',   'where': 'WHERE',   'update': 'UPDATE',
   'insert': 'INSERT',   'into': 'INTO',   'values': 'VALUES',
   'if': 'IF',   'when': 'WHEN',   'then': 'THEN',
   'else': 'ELSE',   'declare': 'DECLARE',   'function': 'FUNCTION',
   'as': 'AS',   'begin': 'BEGIN',   'end': 'END',
   'return': 'RETURN',   'not': 'NOT',   'null': 'NULL',
   'primary': 'PRIMARY',   'key': 'KEY',   'data': 'DATA',
   'base': 'BASE',   'table': 'TABLE',   'procedure': 'PROCEDURE',
   'reference': 'REFERENCE',   'add': 'ADD',   'int': 'INT',
   'decimal': 'DECIMAL',   'date': 'DATE',   'datetime': 'DATETIME',
   'nchar': 'NCHAR',   'nvarchar': 'NVARCHAR',   'concatena': 'CONCATENA',
   'substraer': 'SUBSTRAER',   'hoy': 'HOY',   'contar': 'CONTAR',
   'suma': 'SUMA',   'cas': 'CAS',   'set': 'SET',
   'while': 'WHILE',   'delete': 'DELETE'
}
```

SE AGREGAN LAS PALABRAS RESERVADAS

```
tokens = [
          'ID',
          'INTEGER_VALUE',          'DECIMAL_VALUE',
          'SEMICOLON',          'NAME',
          'STRING',          'L_PAREN',
          'R_PAREN',          'COMMA',
          'ASSIGN',          'EQUALS',
          'NOT_EQ',          'LESS_THAN',
          'GREATER_THAN',          'LESS_EQ',
          'GREATER_EQ',          'AND',
          'OR',          'NOT_SIGN',
          'PLUS',          'MINUS',
          'TIMES',          'DIVIDE',          'DOT'
      ] + list(reservadas.values())
```

SE AGREGAN LOS TOKENS Y VALORES DE CADA UNO

Gramatica XSQL

```
  init   : statements
;

  statements : statements statement
 |  statement
 ;

  statement : create_database_statement SEMICOLON
            | use_statement SEMICOLON
            | declare_statement SEMICOLON
            | set_statement SEMICOLON
            | create_table_statement SEMICOLON
            | select_statement SEMICOLON
            | insert_statement SEMICOLON
            | create_function_statement SEMICOLON
            | create_procedure_statement SEMICOLON
            | call_function_statement SEMICOLON
            | alter_table_statement SEMICOLON
            | if_statement SEMICOLON
            | exec_statement SEMICOLON
            | drop_table_statement SEMICOLON
            | update_statement SEMICOLON
            | while_statement SEMICOLON
            | truncate_statement SEMICOLON
            | return_statement SEMICOLON
            | delete_statement SEMICOLON
;
```

Gramatica XSQL

```
create_database_statement    : CREATE DATA BASE NAME
  ;

  use_statement   : USE NAME
;

  declare_statement   : DECLARE ID AS type
 |  DECLARE ID  type
 ;

  set_statement   : SET assignments
;

  assignments    : assignments COMMA ID ASSIGN a
    | ID ASSIGN a
;

  create_table_statement : CREATE TABLE NAME L_PAREN properties R_PAREN
 ;

  properties : properties COMMA property
|  property
;

  property   : NAME type null_prod PRIMARY KEY
|  NAME type null_prod
| NAME type null_prod REFERENCE NAME L_PAREN NAME R_PAREN
;

  null_prod  : NOT NULL
| NULL
| e
;
```

Gramatica XSQL

```
select_statement   : SELECT columns FROM NAME
| SELECT columns FROM NAME WHERE a
;

 insert_statement   : INSERT INTO NAME L_PAREN column_names R_PAREN
VALUES L_PAREN vals R_PAREN
;

 column_names   : column_names COMMA NAME
| NAME
;

 columns    : columns COMMA column
  |column
;

 column   : TIMES
| NAME
| NAME DOT NAME
  | case_statement NAME
          | a NAME
         | if_statement NAME
;

 vals   : vals COMMA a
|  a

create_function_statement  : CREATE FUNCTION NAME L_PAREN parameters
R_PAREN RETURN type AS BEGIN statements END
| CREATE FUNCTION NAME L_PAREN R_PAREN RETURN type AS BEGIN statements
END

 create_procedure_statement : CREATE PROCEDURE NAME L_PAREN parameters
R_PAREN AS BEGIN statements END
| CREATE PROCEDURE NAME L_PAREN R_PAREN AS BEGIN statements END
```

Gramatica XSQL

```
parameters : parameters COMMA ID AS type
    | parameters COMMA ID type
| parameters : ID type
    | ID AS type

  alter_table_statement  : ALTER TABLE NAME ADD COLUMN NAME type
    | ALTER TABLE NAME DROP COLUMN NAME
 ;

  if_statement   : IF a THEN statements END IF
    | IF a THEN statements ELSE statements END IF
| IF L_PAREN a COMMA a COMMA a R_PAREN
;

  exec_statement : EXEC NAME vals
    | EXEC NAME args
   | EXEC NAME
;

  args   : args COMMA ID ASSIGN a
  | ID ASSIGN a
  ;

 drop_table_statement   : DROP TABLE NAME
;
  update_statement   : UPDATE NAME SET column_assignments WHERE a
;

  column_assignments  : column_assignments COMMA NAME ASSIGN a
| NAME ASSIGN a
;

  while_statement    : WHILE a BEGIN statements END
;
```

Gramatica XSQL

```
truncate_statement : TRUNCATE TABLE NAME
 ;

  delete_statement : DELETE FROM NAME WHERE a
 ;

  case_statement : CASE when_statements END NAME
;

  when_statements : WHEN a THEN a when_statements
 | ELSE THEN a
 ;

  type : INT
       | DECIMAL
       | DATE
       | DATETIME
| NCHAR L_PAREN a R_PAREN
       | NVARCHAR L_PAREN a R_PAREN
;

  a   : a OR b
  | b
;

 b  : b AND c
| c
;

  c  : NOT_SIGN d
  | d
;
```

Gramatica XSQL

```
d   : d EQUALS e
        | d NOT_EQ e
        | d LESS_THAN e
        | d GREATER_THAN e
        | d LESS_EQ e
        | d GREATER_EQ e
   | d  : e
;

  e    : e PLUS f
        | e MINUS f
   | f
;

  f    : f TIMES g
        | f DIVIDE g
  | g
;

  g  : MINUS h
  | h
;

h    : INTEGER_VALUE
   | DECIMAL_VALUE
| STRING
  | ID
  | NAME
| L_PAREN a R_PAREN
| exec_statement
        | call_function_statement
;
```

Gramatica XSQL

```
call_function_statement   : function_name_prod L_PAREN vals R_PAREN
    | function_name_prod L_PAREN R_PAREN
| CAS L_PAREN a AS type R_PAREN
;

function_name_prod   : HOY
                     | CONCATENA
                     | SUBSTRAER
                     | CONTAR
                     | SUMA
;

return_statement : RETURN a
;
```

03
Modelos

Modelos XSQL

Se crearon modelos XSQL para la manipulación de datos al momento de ejecución del interprete, esto para dinamizar el análisis semántico en el manejador de bases de datos.

Se realizarón modelos de las siguientes acciones:

AlterTable
Assignment
BinaryOperation
CallFunctionStatement
CasStatement
CreateDBStatement
CreateTableStatement
DeclareStatement
DeleteStatement
DropTableStatemen
EOF, Node
ElseStatement
ExecStatement
FunctionModel
IfStatement
InsertStatement
Instruction
OperationType
OrdenEjecucion
ParameterStatement
ProcedureModel, ProdecureStatement
ReturnStatement
SelectStatement
SetStatement
SymbolType
TableColumn
TableProperty
TruncateTableStatement
UnaryOperation, UpdateStatement, UseStatement, Value, ValueType, Variable, VariableType, WhenStatement, WhileStatement

04
Estructura XML

Estructura DDL en XML

Se creó una estructura para el DDL, es decir, para la estructura de las bases de datos a manipular en el IDE, agregando todos los atributos escenciales, como: nombre de la base de datos, tablas, nombre de cada tabla, campos de cada una, atributos de cada campo, etc.

Ejemplo:

```
<base>
   <nombre>usuarios</nombre>
   <tabla>
      <nombre>tbtipo_usuario</nombre>
   </tabla>
   <tabla>
      <nombre>tbtipo_usuario2</nombre>
      <campo>
         <nombre>id_tipo</nombre>
         <tipo_dato>int</tipo_dato>
         <llave_primaria>1</llave_primaria>
         <nulo>0</nulo>
            <tabla_referencia>tipo_usuario</tabla_referencia>
            <campo_referencia>id_tipo_usuario</campo_referencia>
      </campo>
      <campo>
         <nombre>tipo_usuario</nombre>
         <tipo_dato>varchar</tipo_dato>
         <llave_primaria>0</llave_primaria>
         <nulo>0</nulo>
            <tabla_referencia>tipo_usuario</tabla_referencia>
            <campo_referencia>id_tipo_usuario</campo_referencia>
      </campo>
   </tabla>
</base>
```

Estructura DML en XML

Asímismo, se creó una estructura para el DML, es decir, para los registros correspondientes de cada tabla, correspondiente a una base de datos a manipular en el IDE, agregando todos los atributos escenciales, como: nombre de la base de datos, nombre de la tabla, cada registro, con sus diferentes campos y valores correspondientes.

Ejemplo:

```
<registros>
   <base_datos>usuarios</base_datos>
   <tabla>usuario</tabla>
   <registro>
      <campo>id_usuario</campo>
      <campo>nombre</campo>
      <campo>apellido</campo>
      <valor>1</valor>
      <valor>Diego</valor>
      <valor>Estrada</valor>
   </registro>
   <registro>
      <campo>id_usuario</campo>
      <campo>nombre</campo>
      <campo>apellido</campo>
      <valor>2</valor>
      <valor>William</valor>
      <valor>Miranda</valor>
   </registro>
</registros>
```