Mini Projet - THL et Compilation (I513).

Titre : Grammaire formelle de GLSimpleSQL.

Auteure : *FATIMA ZAHRA LEGDOU , WISSAL BEGGAR.*

Date : 20/11/2025 .

---

1. **Objectif du document :**

   Ce document décrit la grammaire formelle (BNF/EBNF) du langage GLSimpleSQL, correspondante aux fichiers fournis (lexer.l, parser.y, main.c). Le but est de préciser les terminaux reconnus, les règles syntaxiques, les priorités, ainsi que les vérifications sémantiques et les tests.

2. **Contexte du projet :**

   GLSimpleSQL est un sous-ensemble simplifié de SQL que nous devons analyser avec Flex (lex) et Bison (yacc). Le compilateur n'exécute pas les requêtes sur une base réelle ; il analyse, construit éventuellement une structure interne et produit des statistiques/erreurs.

3. **Liste des fichiers fournis :**

   - lexer.l (analyseur lexical).
   - parser.y (analyseur syntaxique / grammaire Bison) .
   - main.c (programme principal).
   - tests/ (jeu de tests SQL).

4. **Liste des tokens (terminaux) — spécification lexicale :**

   Mots-clés :
   SELECT, FROM, WHERE, CREATE, TABLE, INSERT, INTO, VALUES, UPDATE, SET, DELETE, DROP, AND, OR, NOT

   Types :
   INT (INT_T), FLOAT (FLOAT_T), VARCHAR (VARCHAR_T), BOOL (BOOL_T)

   Valeurs / littéraux :
   INTVAL -> entiers : [0-9]+
   FLOATVAL -> réels : [0-9]+\.[0-9]+
   STRING -> chaînes entre '...'
   TRUE_T / FALSE_T

   Identificateurs :
   IDENT -> [a-zA-Z_][a-zA-Z0-9_]*

Opérateurs de comparaison :
EQ (=), NEQ (!=), LT (<), GT (>), LEQ (<=), GEQ (>=)

Symboles :
COMMA (','), SEMICOLON (';'), LPAREN ('('), RPAREN (')'), STAR ('*')

Commentaires / espaces :
- Espaces, tabulations, retours-ligne ignorés
- Commentaires ligne : -- ...
- Commentaires multi-lignes : /* ... */

5. **Grammaire formelle (BNF) :**

```
<start> ::= <stmt_list>

<stmt_list> ::= ε
              | <stmt_list> <stmt> ';'

<stmt> ::= <select_stmt>
         | <insert_stmt>
         | <update_stmt>
         | <delete_stmt>
         | <create_table_stmt>

<select_stmt> ::= SELECT <column_list> FROM IDENT <where_clause_opt>

<insert_stmt> ::= INSERT INTO IDENT '(' <column_list> ')' VALUES '(' <value_list> ')'

<update_stmt> ::= UPDATE IDENT SET <assignment_list> <where_clause_opt>

<delete_stmt> ::= DELETE FROM IDENT <where_clause_opt>

<create_table_stmt> ::= CREATE TABLE IDENT '(' <table_def> ')'

<column_list> ::= IDENT
                | <column_list> ',' IDENT

<value_list> ::= INTVAL
               | FLOATVAL
               | STRING
               | TRUE_T
               | FALSE_T
               | <value_list> ',' INTVAL
```

```
            | <value_list> ',' FLOATVAL
            | <value_list> ',' STRING
            | <value_list> ',' TRUE_T
            | <value_list> ',' FALSE_T


<assignment_list> ::= IDENT EQ <expression>
                    | <assignment_list> ',' IDENT EQ <expression>


<expression> ::= INTVAL
            | FLOATVAL
            | STRING
            | TRUE_T
            | FALSE_T
            | IDENT

<table_def> ::= IDENT INT_T
            | IDENT FLOAT_T
            | IDENT VARCHAR_T
            | IDENT BOOL_T
            | <table_def> ',' IDENT INT_T
            | <table_def> ',' IDENT FLOAT_T
            | <table_def> ',' IDENT VARCHAR_T
            | <table_def> ',' IDENT BOOL_T

<where_clause_opt> ::= ε
                    | WHERE <expression>
```

6. **Description des non-terminaux :**

<stmt_list> : liste de statements séparés par des ';'. Permet plusieurs commandes successives.
<select_stmt> : SELECT suivi d'une liste de colonnes (ou '*') depuis une table, optionnellement avec une clause WHERE.
<insert_stmt> : INSERT INTO [table] (liste_colonnes) VALUES (liste_valeurs).
<update_stmt> : UPDATE table SET liste_affectations [WHERE expr]
<delete_stmt> : DELETE FROM table [WHERE expr]
<create_table_stmt> : CREATE TABLE nom (définition des colonnes et types)
<column_list> : liste de noms de colonne séparés par des virgules
<value_list> : liste de valeurs littérales (int, float, string, bool)
<assignment_list> : suite de colonnes affectées: col = expr [, ...]
<expression> : littéral ou identifiant (niveau simple dans cette version)
<table_def> : définition de colonnes et types pour CREATE TABLE

7. Priorités et associativité :

Priorités recommandées :
- Définir l'associativité des opérateurs logiques pour éviter les conflits :
  %left OR
  %left AND
  %right NOT

(Actuellement, la grammaire traite WHERE <expression> simple ; pour expressions logiques complexes,
il faudra étendre <expression> et ajouter les priorités ci-dessus.)

8. Actions sémantiques implémentées :

Actions sémantiques actuelles :
- À la réduction de chaque statement, le parser affiche une ligne informative :
  * SELECT sur la table <nom>
  * INSERT dans la table <nom>
  * UPDATE sur la table <nom>
  * DELETE sur la table <nom>
  * CREATE TABLE <nom>

Remarque : les actions sémantiques sont encore minimales : elles impriment le nom de la table.
Pour compléter le projet, il faut ajouter :
- construction d'un AST pour chaque type de requête,
- table des symboles (stockage des schémas des tables créées),
- vérifications sémantiques (existence table, correspondance champs/valeurs, etc.).

9. Gestion des erreurs :

- Erreurs lexicales (lexer) :
message "Erreur lexicale: <texte>"

- Erreurs syntaxiques (Bison) :
yyerror() affiche "Erreur syntaxique: <message>"

- Erreurs sémantiques (à implémenter) :
messages formatés incluant numéro de ligne (yylineno)

Exemple attendu :
ERREUR SÉMANTIQUE ligne 5 : La table 'Produit' n'existe pas.