Module : THL et Compilation (I513)
Filière : LST GL S5
Professeur : N. Mouhni
Année universitaire : 2025-2026

Rapport de Mini Projet :

Interpréteur des requêtes du langage SQL simplifié

Réaliser par :

Fatima Zahra Legdou

Wissal Beggar

19 Novembre, 2025

1.Introduction :

Ce projet, réalisé dans le cadre du module de Théorie des Langages et Compilation, a pour objectif la conception et l'implémentation d'un interpréteur de requêtes SQL simplifiées nommé GLSimpleSQL. Développé en langage C à l'aide des outils Flex et Bison, cet interpréteur permet d'analyser syntaxiquement et sémantiquement des commandes SQL basiques telles que **CREATE TABLE, SELECT, INSERT, UPDATE, DELETE** , **DROP** et **TABLE.** Bien qu'il ne gère pas de base de données réelle, il effectue des vérifications structurelles, détecte les erreurs et génère des statistiques détaillées sur les requêtes traitées.

Ce rapport présente les différentes étapes de réalisation du projet : l'analyse lexicale avec Flex, l'analyse syntaxique avec Bison, la gestion des actions sémantiques, la mise en place d'une table des symboles, ainsi que les tests effectués pour valider le bon fonctionnement de l'interpréteur.

2. Architecture du Projet :

Lexer.l (Analyseur Lexical) :

Définit les expressions régulières pour reconnaître les tokens SQL

Il permet de :

- Reconnaissance des mots-clés, identificateurs, constantes.
- Gestion des commentaires et espaces blancs.
- Détection des erreurs lexicales.

Parser.y (Analyseur Syntaxique) :

Définit la grammaire GLSimpleSQL et les actions sémantiques

Il permet :

- Définir les règles BNF pour chaque type de requête SQL .
- Construire l'AST (Abstract Syntax Tree) .
- Appeler les fonctions de vérification sémantique .
- Gérer les erreurs syntaxiques.

Main.c (Programme Principal) :

Point d'entrée et orchestration globale ,

Il permet de :

- Initialiser la table des symboles .
- Lancer une boucle pour lire les requêtes de l'utilisateur .
- Appeler les analyseurs lexical et syntaxique .
- Afficher les résultats ainsi que les statistiques.

Symbol_table.c/h (Gestion des Métadonnées) :

Maintenir les informations sur les tables et leurs schémas

Il permet de :

- Stocker les noms des tables ainsi que leurs champs .
- Vérifier l'existence des tables et des champs .
- Gérer les types de données .
- Détecter les incohérences sémantiques.

Table des symboles (Métadonnées) :

La table des symboles est utilisée pour stocker les informations sur les tables et champs créés par l'interpréteur.

Elle permet :

- De stocker les tables créées avec la commande CREATE TABLE.
- De stocker les champs et leurs types (INT, VARCHAR, FLOAT, etc.).
- De vérifier l'existence d'une table lors des commandes SELECT, INSERT, UPDATE.
- De vérifier que le nombre de champs est correct lors des insertions.
- De détecter les doublons (une table déjà existante).
- De supprimer les tables avec DROP TABLE.

Flux de Compilation :

1. Flex traite lexer.l pour générer lex.yy.c.
2. Bison traite parser.y pour générer parser.tab.c et parser.tab.h.
3. GCC compile tous les fichiers .c en un exécutable unique.

3.ANALYSE LEXICALE - FICHIER lexer.l :

Le lexer transforme le texte SQL en tokens utilisables par l'analyseur syntaxique. Il lit caractère par caractère et identifie les éléments de base du langage.

3.1 Tokens Reconnaissables:

- Mots-clés SQL : SELECT, FROM, WHERE, CREATE, TABLE, INSERT,

```
18   "SELECT"        return SELECT;
19   "FROM"          return FROM;
20   "WHERE"         return WHERE;
21   "INSERT"        return INSERT;
22   "INTO"          return INTO;
23   "VALUES"        return VALUES;
24   "UPDATE"        return UPDATE;
25   "SET"           return SET;
26   "DELETE"        return DELETE;
27   "DROP"          return DROP;
28   "TABLE"         return TABLE;
29   "CREATE"        return CREATE;
30
```

- UPDATE, DELETE, DROP, AND, OR, NOT
- Types de données : INT, FLOAT, VARCHAR, BOOL, TRUE, FALSE
- Opérateurs : =, !=, <, >, <=, >=

3.2 Éléments divers :

- Identifiants (noms) : nom_table, age
- Nombres : 123, 45.67
- Textes : 'exemple'
- Symboles : ,, ;, (,), *

3.3 Gestion des Erreurs

```
104   . {
105       fprintf(stderr,
106           "ERREUR LEXICALE ligne %d : caractère invalide '%s'\n",
107           yylineno, yytext);
108   }
109
110   %%
```

Tout caractère non reconnu génère un message d'erreur immédiat.

Exemple :

Pour :

SELECT nom FROM Clients;,

le lexer produit :

SELECT, IDENT(nom), FROM, IDENT(Clients), SEMICOLON

4. ANALYSE SYNTAXIQUE – FICHIER parser.y

Le parser vérifie si les tokens envoyés par le lexer sont organisés dans un ordre correct et forment des instructions SQL valides.

4.1 Grammaire Supportée :

- **SELECT** : SELECT colonnes FROM table [WHERE condition]

src > parser.y

```
366   select_stmt:
370       {
386           fprintf(stderr, "\nRequête SELECT analysée :\n");
387
388   fprintf(stderr, "- Table : %s\n", $4);
389
390   if(strcmp($2->columns, "*") == 0)
391
392       fprintf(stderr, "- Nombre de champs : %d (ALL *)\n", t ? count_table_fields(t) : 0);
393
394   else
395       fprintf(stderr, "- Nombre de champs : %d (%s)\n", $2->num_columns, $2->columns);
396
397   fprintf(stderr, "- Clause WHERE : %s\n", $5->where_present ? "OUI" : "NON");
398
399   fprintf(stderr, "- Nombre de conditions : %d\n", $5->cond_count);
400
401   fprintf(stderr, "- Opérateurs logiques : %d\n\n", $5->logic_count);
402
403
404
405           free($4);
406
407           free_reqinfo($2);
408
409           free_reqinfo($5);
410
411       }
412
```

- **INSERT** : INSERT INTO table (colonnes) VALUES (valeurs)
- **UPDATE** : UPDATE table SET champ=valeur [WHERE condition]
- **DELETE** : DELETE FROM table [WHERE condition]
- **CREATE TABLE** : CREATE TABLE table (colonne type, ...)

4.2 Fonctionnement :

- Reçoit les tokens du lexer.
- Vérifie leur agencement selon les règles SQL.
- Affiche un message de confirmation pour chaque requête valide.
- Signale les erreurs de structure.

Exemple

Pour :

SELECT nom FROM Clients;

Sortie :

SELECT sur la table Clients

5. COMPILATION ET EXÉCUTION

Compilation :

flex lexer.l

bison parser.y

gcc lex.yy.c parser.tab.c -lfl -o sql_interpreter

Lancement :

./sql_interpreter # Mode interactier

./sql_interpreter < tests/basic.sql # Fichier d'entrée

Résultats :

Avec basic.sql :

CREATE TABLE Clients

INSERT dans la table Clients

SELECT sur la table Clients

Avec errors.sql :

Erreur syntaxique

Erreur sémantique: Table inexistante

Erreur sémantique: Nombre de valeurs incorrect

6. Tests du projet :

Pour valider le fonctionnement de l'interpréteur, deux fichiers de test ont été utilisés : test1.txt pour les tests réussis et testerreur.txt pour les tests avec erreurs.

6.1 Test 1 (test1.txt) :

Exemples de requêtes :

src > test1.txt

```
-- Test 1 : Création et insertion
CREATE TABLE Etudiant (
    id INT,
    nom VARCHAR(50),
    age INT
);

INSERT INTO Etudiant VALUES (1, 'Diallo', 20);
INSERT INTO Etudiant (id , nom ) VALUES (2 , 'Sow ');

SELECT * FROM Etudiant;
SELECT nom FROM Etudiant;
SELECT nom, age FROM Etudiant WHERE age > 18;
SELECT * FROM Etudiant WHERE id = 1 AND age < 25;

UPDATE Etudiant SET age = 21 WHERE id = 1;
UPDATE Etudiant SET nom = 'Ba', age = 22 WHERE id = 2;

DELETE FROM Etudiant WHERE age < 18;

```

Résultats produits par le programme :

- CREATE TABLE OK
- INSERT OK
- SELECT : affichage des lignes avec statistiques.

6.2 Test Erreur (testerreur.txt) :

Exemples de requêtes avec erreurs :

src > testerreur.txt

```
1   -- Erreur : table inexistante
2    SELECT * FROM Produit ;
3
4    -- Erreur : champ inexistant
5    SELECT prix FROM Etudiant ;
6
7   -- Erreur : nombre de valeurs incorrect
8   INSERT INTO Etudiant VALUES (3 , 'Kane ');
9
10  -- Erreur : table d j existante
11  CREATE TABLE Etudiant ( id INT );
12   CREATE TABLE Etudiant ( num INT );
13
14   -- Erreur : syntaxe invalide
15   SELECT FROM Etudiant ;
16   SELECT * Etudiant ;
17   -- Erreur : champ inexistant
18  SELECT prix FROM Etudiant ;
19
```

Résultats produits par le programme :

- ERREUR SÉMANTIQUE : la table 'Produit' n'existe pas
- ERREUR SÉMANTIQUE : CREATE TABLE Etudiant : existe déjà
- Erreur syntaxique détectée

7. CONCLUSION :

Le projet a permis de comprendre concrètement le fonctionnement d'un interpréteur de langage, en maîtrisant Flex/Bison, les grammaires formelles, la gestion des erreurs et la liaison lexique-syntaxe, tout en identifiant des difficultés de réglage grammatical et de clarté des messages d'erreur, avec des perspectives d'amélioration comme l'ajout de fonctionnalités SQL et une interface plus interactive.