Université Moulay Ismaïl

Faculté des Sciences et Techniques

Département d'Informatique

RAPPORT DE PROJET :

Module : Théorie des Langages et Compilation (I513)
Sujet : Interpréteur de requêtes du langage SQL simplifié (GLSimpleSQL)
Année Universitaire : 2025-2026
Filière : LST GL S5
Réalisé par : Abdelghani Salek
Encadré par : Professeur N. Mouhni

1. Introduction

Ce projet s'inscrit dans le cadre du module de Théorie des Langages et Compilation. L'objectif est de concevoir et développer un interpréteur pour GLSimpleSQL, un sous-ensemble du langage SQL standard. Contrairement à un Système de Gestion de Base de Données (SGBD) complet qui gère le stockage physique, notre interpréteur se concentre sur la chaîne de compilation frontale :

1. **Analyse Lexicale :** Reconnaissance des mots du langage.
2. **Analyse Syntaxique :** Validation de la structure grammaticale.
3. **Analyse Sémantique :** Vérification de la cohérence des données (types, existence des tables).
4. **Reporting :** Affichage de statistiques précises sur les requêtes analysées.

Le projet a été réalisé en langage C, en utilisant les outils standards Flex et Bison.

2. Architecture Logicielle

Le projet est structuré de manière modulaire pour séparer les responsabilités de chaque phase de l'analyse.

2.1 Organisation des Fichiers

- **scanner.l (Flex) :** Contient les expressions régulières pour la tokenisation.
- **parser.y (Bison) :** Contient la grammaire BNF et la logique de coordination.
- **structures.h :** Définit les structures de données C (listes chaînées pour les tables et les champs).

- **semantics.c :** Contient l'implémentation de la Table des Symboles (ajout/recherche) et les fonctions de vérification sémantique.
- **Makefile :** Automatise la compilation et l'édition de liens.

2.2 Flux de Données

Requête SQL (Texte) → Scanner (Flex) → Flux de Tokens → Parser (Bison) ↔ Table des Symboles → Résultat / Erreur

3. Phase 1: Analyse Lexicale (Scanner)

L'analyseur lexical a pour rôle de découper le flux d'entrée en unités atomiques (tokens).

3.1 Spécificités d'Implémentation

- Insensibilité à la casse : Pour respecter le standard SQL, nous avons défini des patterns permettant de reconnaître les mots-clés peu importe leur casse (ex: Select, SELECT, select).
- Gestion des Identificateurs : Les noms de tables et champs sont reconnus par l'expression régulière [a-zA-Z][a-zA-Z0-9_]. Cette règle est positionnée après les mots-clés pour respecter la priorité (éviter que WHERE soit vu comme un ID).
- Types de données : Gestion complète des littéraux :
    - Entiers ([0-9]+) et Réels ([0-9]+\.[0-9]+)
    - Chaînes de caractères ('...' ou "...")
    - Booléens (TRUE, FALSE)

3.2 Gestion des Commentaires

Le scanner filtre automatiquement :

- Les commentaires mono-ligne SQL (-- commentaire).
- Les commentaires multi-lignes C-style (/* commentaire */).

4. Phase 2: Analyse Syntaxique (Parser)

Le fichier parser.y définit la grammaire formelle et orchestre l'analyse.

4.1 Grammaire et Conflits

- La grammaire couvre les instructions DDL (CREATE, DROP) et DML (SELECT, INSERT, UPDATE, DELETE).
- Un point d'attention particulier a été porté à la précédence des opérateurs dans les clauses WHERE. Nous avons structuré la grammaire pour respecter l'ordre naturel :

NOT > AND > OR, évitant ainsi les ambiguïtés sans recourir massivement aux directives (left ou right).

4.2 Calcul des Statistiques

- Conformément au cahier des charges, nous n'avons pas construit d'Arbre Syntaxique Abstrait (AST) complet. Nous avons opté pour une analyse dirigée par la syntaxe : les actions sémantiques (code C entre accolades) sont exécutées lors de la réduction des règles.
- Exemple : Dans une clause WHERE, un compteur est incrémenté à chaque détection d'un opérateur logique (AND, OR), permettant d'afficher le total à la fin de la requête.

5. Phase 3: Analyse Sémantique et Table des Symboles

C'est le "cœur intelligent" de l'interpréteur qui valide le sens des requêtes.

5.1 Structure de Données (Table des Symboles)

1. Puisque le stockage disque n'est pas requis, nous utilisons une structure en mémoire vive (RAM) persistante durant l'exécution du programme.
2. Liste chaînée de Tables : Chaque nœud contient le nom de la table et un pointeur vers ses colonnes.
3. Liste chaînée de Colonnes : Pour chaque table, nous stockons le nom des champs et leurs types (INT, VARCHAR, etc.).
4. Cette structure dynamique permet de gérer un nombre indéfini de tables et de champs sans gaspillage de mémoire.

5.2 Vérifications Sémantiques Implémentées

Le parser interroge la table des symboles pour valider les règles suivantes :

1. Existence des tables : Impossible de faire un SELECT ou un INSERT sur une table non déclarée via CREATE.
2. Unicité : Erreur si on tente de CREATE une table existante.
3. Validité des champs : Vérification que les colonnes mentionnées dans SELECT, WHERE ou SET appartiennent bien à la table.
4. Cohérence INSERT : Vérification stricte que le nombre de valeurs insérées correspond au nombre de colonnes de la table (ou aux colonnes spécifiées).

6. Tests et Validation

Le projet a été validé à l'aide d'un jeu de tests complet (requetes_test.sql) couvrant :

1. **Cas Nominaux :** Création de table, insertion correcte, sélections variées.
2. **Cas d'Erreurs :** Tentative de sélection sur une table inexistante, insertion avec un nombre incorrect de valeurs, erreurs de syntaxe.

L'utilisation d'un Makefile permet une compilation simple et une exécution rapide des tests via la commande make run.

7. Conclusion et Perspectives

Ce projet nous a permis de maîtriser l'interaction complexe entre l'analyse lexicale, syntaxique et sémantique. La principale difficulté technique a résidé dans la gestion correcte de la mémoire C (pointeurs) pour la table des symboles et la synchronisation des erreurs entre Flex et Bison.

Le résultat est un interpréteur fonctionnel, robuste aux erreurs, et conforme aux spécifications pédagogiques. Une évolution naturelle de ce projet serait d'implémenter un moteur de stockage réel (lecture/écriture de fichiers CSV ou binaires) pour transformer cet interpréteur en un mini-SGBD persistant.