République Tunisienne
Ministère de l'Enseignement Supérieur
Et de la Recherche Scientifique
Université de Tunis El Manar
Institut Supérieur d'Informatique

Rapport du mini-projet de techniques de compilation

Réalisé par :

BEN AHMED Yasmine — **CHERNI Arij**

MANSOURI Chaima — **SOUDANI Molka**

Sujet 2

Encadrante académique : Madame MOGAADI Hayet

Réalisé au sein de l'Institut Supérieur d'Informatique

Année Universitaire 2023/2024

Table des matières

Introduction générale **4**

1 Présentation du cadre de projet **5**

Introduction 6

Objectifs du projet 6

Aperçu des outils utilisés 7

.1 Flex (Fast Lexical Analyzer Generator) : 7

.2 Bison (GNU Parser Generator) 8

Conclusion 13

2 Réalisation **14**

Introduction 15

Validation des différentes requêtes 15

.1 Les commandes utilisées 15

.2 Vérification des requêtes 16

.3 Gestion des erreurs 20

Conclusion 21

Bibliographie **23**

Table des figures

1.1 Les patterns 7
1.2 Déclaration dans le code bison 9
1.3 Union et Tokens du code bison 10
1.4 Grammaire 11
1.5 Grammaire 12

2.1 Le fichier Makefile 15
2.2 Requête CREATE 16
2.3 Requête DELETE 16
2.4 Requête UPDATE 17
2.5 Requête ALTER...ADD 17
2.6 Requête ALTER...MODIFY 18
2.7 Requête ALTER...DROP 18
2.8 Requête SELECT 19
2.9 Requête SELECT 19
2.10 Requête SELECT 20
2.11 Requête INSERT 20
2.12 Erreur de selection 20
2.13 Erreur de selection 20
2.14 Erreur de selection 21
2.15 Erreur d'insertion 21

Liste des tableaux

Introduction générale

L'évolution constante des technologies de traitement de données a conduit à une demande croissante pour des outils flexibles et efficaces d'interprétation de requêtes SQL. Dans ce contexte, notre projet se propose de développer un interpréteur de requêtes SQL en utilisant les puissants outils Flex et Bison. L'objectif central de ce projet est de créer un interpréteur capable de gérer les requêtes de manipulation et d'accès à une base de données, notamment les commandes de création, de suppression, de mise à jour et de sélection.

Les requêtes SQL constituent le langage standard pour communiquer avec les bases de données relationnelles. Ainsi, la capacité à interpréter ces requêtes de manière précise et efficace revêt une importance capitale dans de nombreux domaines, allant de la gestion de bases de données à grande échelle à la manipulation de données dans des applications web et mobiles. En utilisant Flex et Bison, nous nous engageons à fournir un interpréteur robuste et performant qui répondra aux besoins variés des utilisateurs dans différents contextes d'application.

Ce rapport présentera en détail notre approche méthodologique pour le développement de cet interpréteur SQL. Nous examinerons les spécifications de l'analyseur lexical et syntaxique, ainsi que les actions sémantiques ajoutées pour améliorer la fonctionnalité et la précision de l'interpréteur. En explorant les différentes étapes de conception et d'implémentation, nous mettrons en lumière les défis rencontrés et les solutions apportées dans le cadre de ce projet ambitieux. En fin de compte, notre objectif est de fournir un outil fiable et convivial qui facilitera la manipulation et l'accès aux données dans diverses applications.

CHAPITRE 1

Présentation du cadre de projet

Plan

Introduction 6

Objectifs du projet 6

Aperçu des outils utilisés 7

.1 Flex (Fast Lexical Analyzer Generator) : 7

.2 Bison (GNU Parser Generator) 8

Conclusion 13

Introduction

Ce chapitre pose les bases du projet en définissant ses objectifs et en présentant les outils utilisés, Flex et Bison, pour développer un interpréteur de requêtes SQL. L'interpréteur vise à analyser et exécuter des requêtes de manipulation et d'accès à une base de données, offrant ainsi une solution robuste pour la gestion des données relationnelles.

Objectifs du projet

L'objectif de ce projet est de concevoir et de développer un interpréteur de requêtes SQL utilisant les outils Flex et Bison. L'interpréteur sera capable d'analyser et d'exécuter des requêtes de manipulation et d'accès à une base de données, en utilisant les fonctionnalités suivantes :

- **Acceptation des Requêtes SQL :**

  L'interpréteur doit être capable d'accepter les requêtes de manipulation et d'accès à une base de données, notamment les requêtes de création, suppression, mise à jour et sélection.

- **Analyse Lexicale :**

  Le projet impliquera la conception et l'implémentation d'un analyseur lexical utilisant l'outil Flex. Cet analyseur sera chargé de décomposer les requêtes SQL en jetons lexicaux significatifs.

- **Analyse Syntaxique :**

  Enrichir l'analyseur syntaxique avec des actions sémantiques permettant d'évaluer les expressions mathématiques données en entrée, assurant ainsi la capacité de calcul de ces expressions.

- **Actions Sémantiques :**

  Une spécification Bison sera utilisée pour définir la grammaire et réaliser l'analyse syntaxique des requêtes SQL, garantissant ainsi leur conformité structurelle. Actions Sémantiques : Des actions sémantiques seront intégrées pour améliorer la fonctionnalité de l'interpréteur, notamment :

  — Calcul du Nombre de Champs Sélectionnés : L'interpréteur devra calculer le nombre de champs à sélectionner dans la requête SELECT afin d'effectuer l'extraction de données de manière précise et efficace.

  — Détection et Signalement des Erreurs : Des mécanismes seront mis en place pour détecter et signaler à l'utilisateur différents types d'erreurs, tels que des erreurs syntaxiques, des références à des champs inexistants ou des opérations non autorisées.

En mettant en œuvre ces fonctionnalités, ce projet vise à fournir un interpréteur robuste et fonctionnel pour les requêtes SQL, offrant une interface conviviale pour manipuler et accéder aux données dans une base de données.

Aperçu des outils utilisés

.1 Flex (Fast Lexical Analyzer Generator) :

❖ **Description générale :**

Flex génère rapidement des analyseurs lexicaux en utilisant des expressions régulières pour identifier des motifs dans un flux de caractères. Il associe ces motifs à des actions spécifiques, offrant ainsi une grande flexibilité pour définir des règles lexicales. Flex produit généralement du code en C ou C++, assurant une portabilité élevée et une intégration facile dans différents environnements de développement.

❖ **Fonctionnalités principales :**

Flex offre des fonctionnalités avancées pour la création d'analyseurs lexicaux performants. Il permet de définir des règles lexicales flexibles avec des expressions régulières, générant automatiquement du code source en C ou C++. Flex facilite ainsi la portabilité et l'intégration dans différents environnements de développement. De plus, il propose des options de personnalisation et d'optimisation, ainsi que des mécanismes de gestion des erreurs, en faisant un outil puissant et polyvalent pour la construction d'analyseurs lexicaux efficaces.

❖ **Utilisation dans le projet :**

Dans notre projet utilisant Flex et Bison pour développer un interpréteur SQL, Flex sera essentiel pour l'analyseur lexical. Il permettra de segmenter les requêtes en tokens lexicaux, assurant une interprétation précise des commandes SQL, tout en facilitant la détection et la gestion des erreurs lexicales.

Spécification de l'analyseur lexical

Chaque entité lexicale est décrite par une expression régulière. Dans cette partie, nous allons expliquer chaque expression régulière utilisée.

Les "Patterns"

`("[^"]+")|('[^']+')` pour les chaines de caractères.
`(_?[a-z]+([a-z_]+)?)` pour les noms des tables et colonnes.
`([0-9]+)` pour les nombres.
`(int|bigint|decimal|float|double|real|date|varchar)` pour les types SQL (la fichier .l contient beaucoup plus).
`("[^"\n]+)|('[^'\n]+)|(_?[a-z]+([a-z_]+)?)("|')` pour les erreurs dans les chaines ou bien les noms.

FIGURE 1.1 – Les patterns

Les "Lateral Matches"

— (SELECT)
— (DELETE)
— (UPDATE)
— (INSERT)
— (CREATE)
— (ALTER)
— (TABLE)
— (COLUMN)
— (ADD)
— (DROP)
— (MODIFY)
— (INTO)
— (VALUES)
— (FROM)
— (WHERE)
— (SET)
— (AND)
— (LIKE)
— (IN)

Pour les mots-clés de la langage SQL. Comme vous le voyez, nous n'avons travaillé qu'avec des expressions régulières en majuscules, car nous avons activé l'option "caseless" de Flex pour appliquer ces expressions régulières pour les minuscules ou bien toute autre combinaison.

.2 Bison (GNU Parser Generator)

❖ **Description générale :**

Bison, également connu sous le nom de GNU Parser Generator, est un outil de compilation utilisé pour générer des analyseurs syntaxiques. Il analyse la grammaire formelle d'un langage et produit du code source en C ou C++ pour construire des analyseurs syntaxiques. Bison offre une approche robuste et efficace pour l'analyse syntaxique des programmes, ce qui en fait un choix populaire dans le développement de compilateurs et d'interprètes.

❖ **Fonctionnalités principales :**

Bison propose des fonctionnalités avancées pour la création d'analyseurs syntaxiques puissants. Il gère les grammaires ambiguës, génère des arbres syntaxiques abstraits et offre des mécanismes pour faciliter la gestion des erreurs. Ces fonctionnalités permettent de créer des

analyseurs syntaxiques précis et flexibles, adaptés à une variété de langages de programmation.

❖ **Utilisation dans le projet :**

Dans notre projet d'interpréteur SQL, Bison sera utilisé pour définir la grammaire des requêtes SQL et réaliser l'analyse syntaxique. En intégrant les règles syntaxiques spécifiques à SQL dans une spécification Bison, nous pourrons construire un analyseur syntaxique capable de comprendre et de valider la syntaxe des requêtes SQL entrantes, garantissant ainsi la cohérence structurelle des commandes interprétées.

Spécification Bison pour l'analyse syntaxique

Dans cette partie nous allons expliquer la grammaire avec laquelle nous avons travaillé.

— **Explication des déclarations en C**

```
extern int yylineno;
extern int column;

int fieldsCount        = 0;
int allFields          = 0;
int tablesCount        = 0;
int updatedFields      = 0;
int insertToFields     = 0;
int createdFields      = 0;
int insertValuesFields = 0;

char* queryType;
char *operationType;

void afficherStats(){
```

FIGURE 1.2 – Déclaration dans le code bison

Nous avons également utilisé plusieurs variables comme mentionné dans le code afin de pouvoir stocker le nombre de tables et de colonnes pour chaque requête ainsi que le type de la requête sous forme de chaine de caractères, car elles seront utilisées pour afficher les statistiques en utilisant la fonction que nous avons définie "afficherStats()" (nous avons seulement mentionné l'en-tête de la fonction dans le code ci-dessus).

— **Union et Tokens**

```
%union {
    char* string;
    int number;
}

%token END
%token<string> SELECT
%token<string> DELETE
%token<string> UPDATE
%token<string> INSERT
%token<string> CREATE
%token<string> ALTER
%token<string> INTO
%token<string> VALUES
%token<string> FROM
%token<string> WHERE
%token<string> AND
%token<string> LIKE
%token<string> SET
%token<string> IN
%token<string> TABLE
%token<string> COLUMN
%token<string> ADD
%token<string> DROP
%token<string> MODIFY
%token<string> DATATYPE
%token<number> number
%token<string> pattren
%token<string> attribute
%token<string> comma
%token<string> semicolon
%token<string> dot
%token<string> equals_sign
%token<string> open_parenthesis
%token<string> close_parenthesis
%token<string> all_columns
%token<string> incompleted_pattren
%type <string> queries

```

FIGURE 1.3 – Union et Tokens du code bison

La partie ”Union” concerne les types que nous recevrons de Flex dans la variable yytext afin que nous puissions les utiliser ultérieurement pour écrire le code C si nécessaire ou pour effectuer des opérations de base comme l'incrémentation ou la décrémentation des nombres par exemple, et comme nous avons besoin d'entiers et de chaıines de caractères dans notre langage (car nous avons différents types de données en SQL et la requête peut être un mélange de tous ces types), nous devons utiliser à la fois ”char *” et ”int” dans la partie union. Puis vous trouvez les Tokens avec leurs types (ceux que nous avons définis plus tôt dans Union ”string” et ”number”) que nous avons définis pour con- struire notre grammaire pour cette langue et qui, comme vous pouvez le voir, correspond aux expressions régulières de Flex.

— **Les règles de production**

```
input:
    line input
;

line:
    queries semicolon END { afficherStats(); return 0; }
|   END
;

queries:
    SELECT columns FROM tables { queryType="SELECT"; }
|   SELECT columns FROM tables WHERE condition { queryType="SELECT"; }

|   DELETE FROM attribute { queryType="DELETE"; }
|   DELETE FROM attribute WHERE condition { queryType="DELETE"; }

|   UPDATE attribute SET column_value_list { queryType="UPDATE"; allFields = 1; }
|   UPDATE attribute SET column_value_list WHERE condition { queryType="UPDATE"; }

|   INSERT INTO attribute VALUES insert_values { queryType = "INSERT"; }
|   INSERT INTO attribute insert_columns VALUES insert_values { queryType = "INSERT"; }

|   CREATE TABLE attribute open_parenthesis create_list close_parenthesis { queryType =
"CREATE"; }

|   ALTER TABLE attribute alter_option { queryType = "ALTER"; }
;

columns:
    all_columns { allFields = 1; }
|   attribute { fieldsCount++;}
|   attribute comma columns { fieldsCount++;}
;

tables:
    attribute { tablesCount++; }
|   attribute comma tables { tablesCount++; }
;

condition:
    condition AND condition
|   attribute IN open_parenthesis queries close_parenthesis
|   attribute IN open_parenthesis pattren_list close_parenthesis
|   equals_list equals_sign equals_list
|   attribute LIKE pattren
;
```

FIGURE 1.4 – Grammaire

```
equals_list:
    attribute
|   number
|   pattren
|   attribute dot attribute
;

pattren_list:
    pattren
|   pattren comma pattren_list
;

column_value_list:
    attribute equals_sign pattren { updatedFields++; }
|   attribute equals_sign pattren comma column_value_list { updatedFields++; }
|   attribute equals_sign number { updatedFields++; }
|   attribute equals_sign number comma column_value_list { updatedFields++; }
;

insert_columns:
    open_parenthesis insert_columns_list close_parenthesis
;

insert_values:
    open_parenthesis insert_values_list close_parenthesis
;

insert_columns_list:
    attribute { insertToFields++; }
|   attribute comma insert_columns_list { insertToFields++; }
;

insert_values_list:
    pattren { insertValuesFields++; }
|   number { insertValuesFields++; }
|   pattren comma insert_values_list { insertValuesFields++; }
|   number comma insert_values_list { insertValuesFields++; }
;

create_list:
    attribute DATATYPE { createdFields++; }
|   attribute DATATYPE comma create_list { createdFields++; }
;

alter_option:
    ADD attribute DATATYPE { operationType = "Ajout"; }
|   DROP COLUMN attribute { operationType = "Suppression"; }
|   MODIFY COLUMN attribute DATATYPE { operationType = "Modification"; }
;
```

FIGURE 1.5 – Grammaire

— **Gestion d'erreur**

Dans Bison, lorsqu'une erreur de syntaxe se produit, et par erreur de syntaxe nous voulons dire que le Token courant ne correspond à aucune règle de notre grammaire, une fonction "yyerror" est invoquée. Cette fonction vous permet de redéfinir son contenu pour personnaliser le message d'erreur, et elle prend comme paramètre le token courant. Cette connaissance est tout ce dont nous avions besoin pour traiter les erreurs de manière appropriée et c'est exactement ce que nous avons fait, en utilisant une macro "HANDLE COLUMN", deux variables "yylineno" et "column" que nous avons définies et dont nous avons parlé plus tôt, et cette fonction "yyerror", nous avons pu tariter tous les erreurs (avec l'utilisation d'une ex- pression

régulière de repli pour agir comme un ”Garbage Collector”, de sorte que si rien ne correspond, un Token d'erreur est renvoyé au Bison) et afficher un message d'erreur utile avec l'emplacement exact de l'erreur et une ligne dessinée pour montrer où regarder exactement si pour une raison l'utilisateur ne pouvait pas comprendre l'erreur (Vous pouvez vérifier comment la gestion des erreurs se présente dans la section ”Réalisation”).

Conclusion

En résumé, ce chapitre a établi le contexte et les objectifs du projet d'interpréteur SQL, mettant en lumière l'importance des outils Flex et Bison dans sa réalisation. L'interpréteur promet d'offrir une solution efficace et flexible pour la manipulation des données, répondant ainsi aux besoins variés des utilisateurs dans différents domaines d'application.

CHAPITRE 2

Réalisation

Plan

Introduction 15

Validation des différentes requêtes 15

.1 Les commandes utilisées 15

.2 Vérification des requêtes 16

.3 Gestion des erreurs 20

Conclusion 21

Introduction

Ce chapitre présente la validation des différentes requêtes SQL à l'aide de l'interpréteur développé, mettant en avant la procédure utilisée pour combiner les analyseurs Flex et Bison afin de détecter et de signaler les erreurs.

Validation des différentes requêtes

Dans cette partie nous allons présenter le résultat obtenu de la combinaison des deux analyseurs, l'interpréteur qui va accepter les requêtes SQL, détecter et signaler différents erreurs.

.1 Les commandes utilisées

Pour combiner les deux analyses, nous devons générer un fichier header qui sera inclus dans le programme Flex afin qu'il puisse renvoyer chaque mot correspondant aux expressions régulières définies vers son Token correct, de sorte que Bison puisse effectuer l'analyse syntaxique.

1. **Commande 1 :** Génère le code source pour l'analyseur lexical à partir du fichier Flex (`.l`).
2. **Commande 2 :** Génère l'analyseur syntaxique à partir du fichier Bison (`.y`) et crée des fichiers de débogage.
3. **Commande 3 :** Compile les fichiers générés par Flex et Bison pour créer un interpréteur.

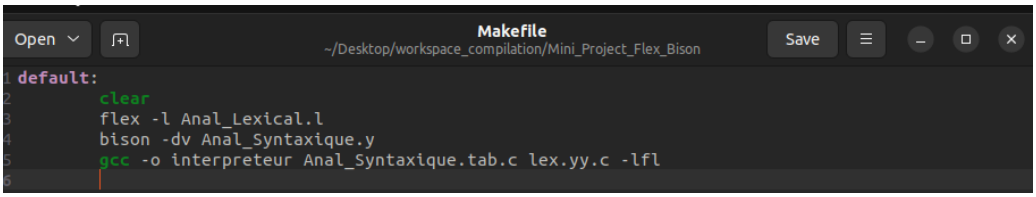

FIGURE 2.1 – Le fichier Makefile

4. **Commande 4 :** Exécuter le programme avec ./interpreteur < programme.txt

.2 Vérification des requêtes

Requête CREATE

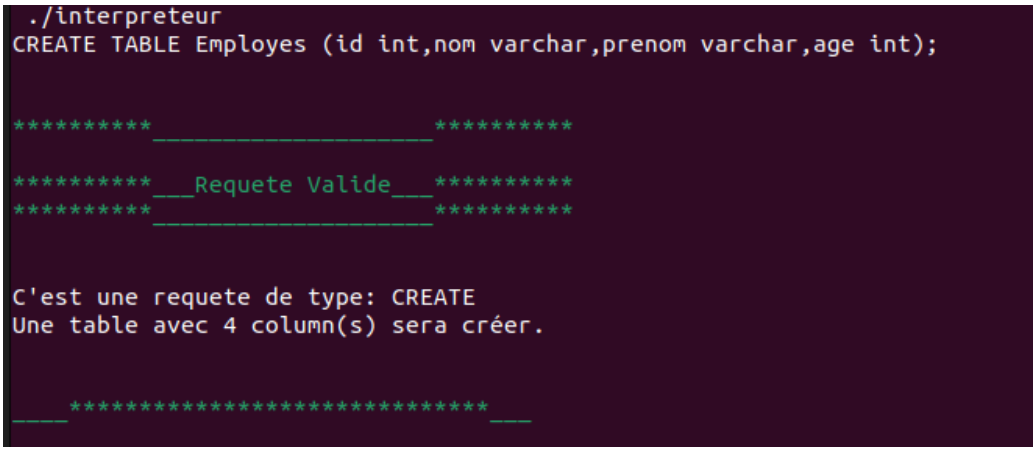

FIGURE 2.2 – Requête CREATE

Requête DELETE

FIGURE 2.3 – Requête DELETE

Requête UPDATE

FIGURE 2.4 – Requête UPDATE

Requête ALTER Ajout

FIGURE 2.5 – Requête ALTER...ADD

Requête ALTER Modification

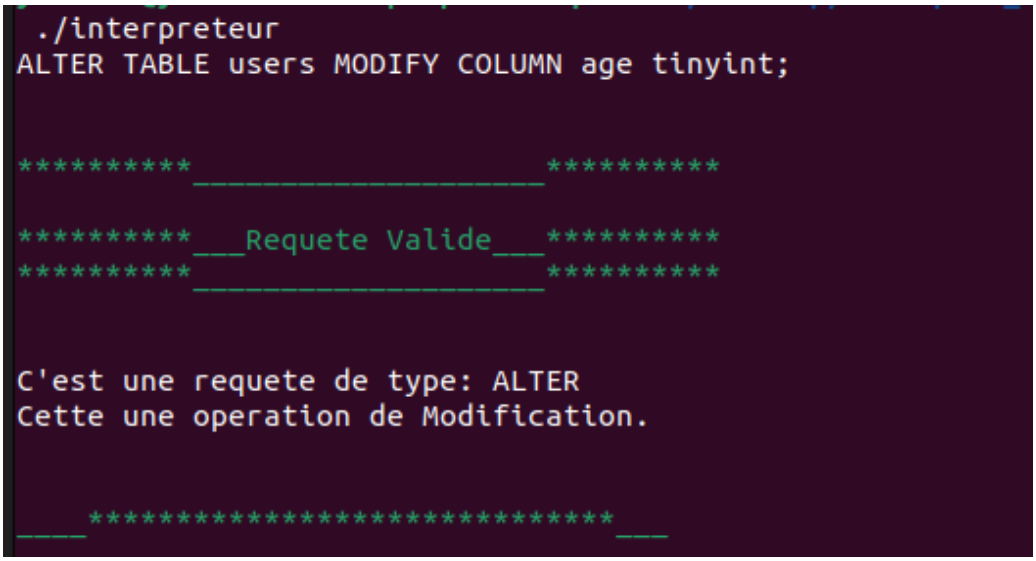

FIGURE 2.6 – Requête ALTER...MODIFY

Requête ALTER Suppression

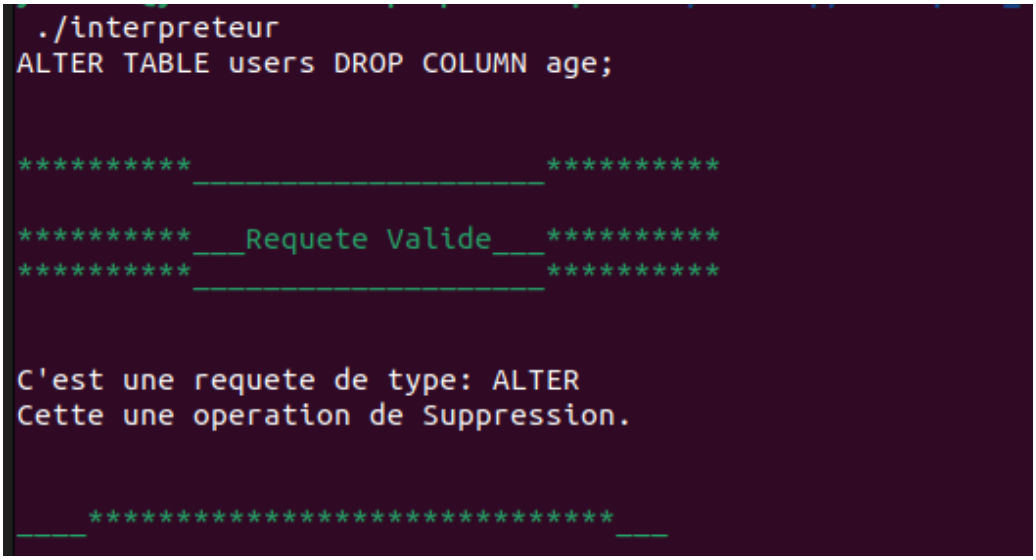

FIGURE 2.7 – Requête ALTER...DROP

Requête SELECT

FIGURE 2.8 – Requête SELECT

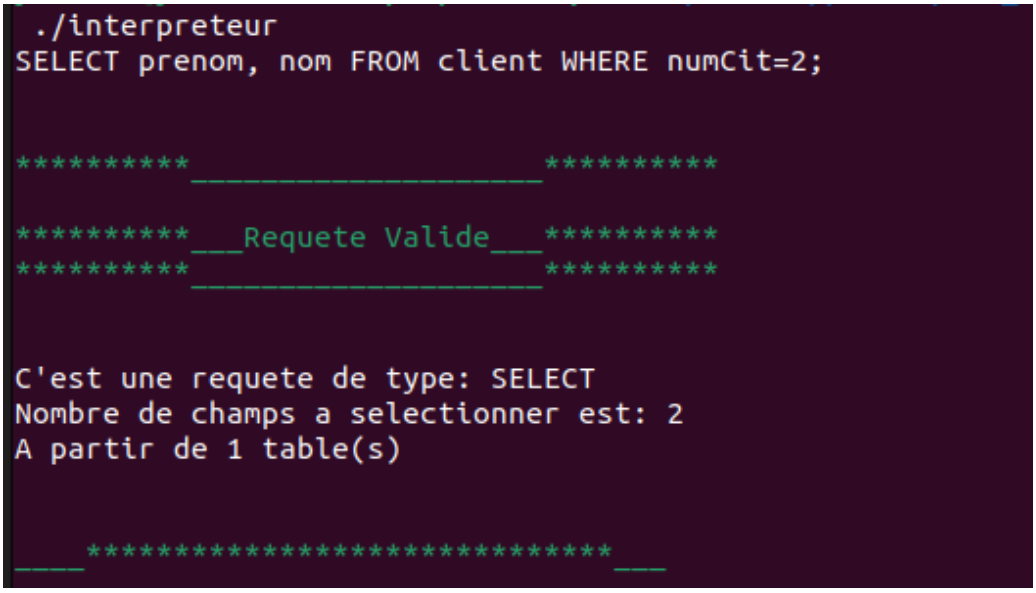

FIGURE 2.9 – Requête SELECT

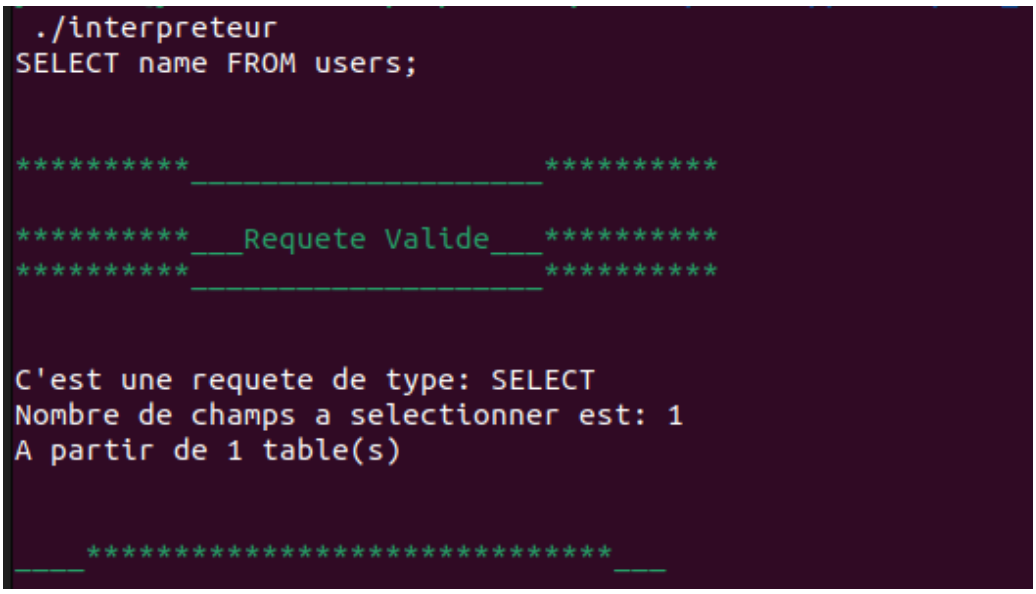

FIGURE 2.10 – Requête SELECT

Requête INSERT

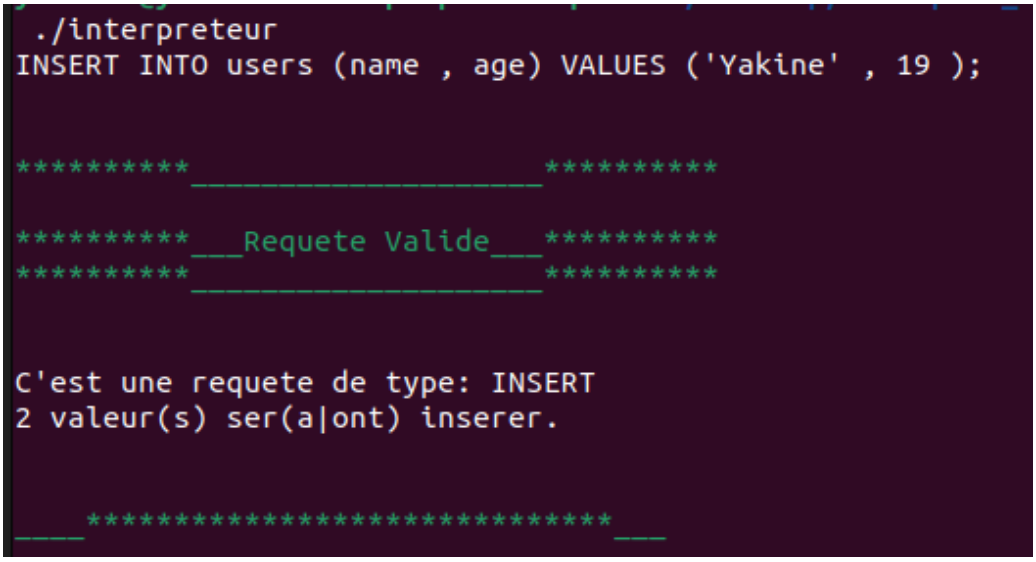

FIGURE 2.11 – Requête INSERT

.3 Gestion des erreurs

FIGURE 2.12 – Erreur de selection

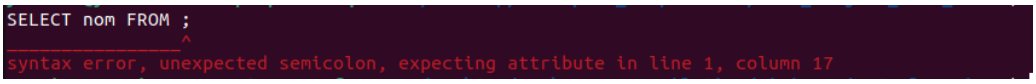

FIGURE 2.13 – Erreur de selection

FIGURE 2.14 – Erreur de selection

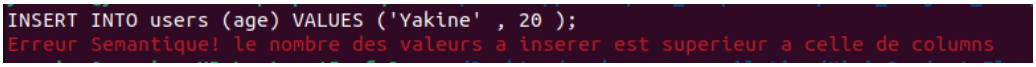

FIGURE 2.15 – Erreur d'insertion

Conclusion

En conclusion, ce chapitre expose les résultats obtenus lors de la validation des requêtes, ainsi que la gestion des erreurs détectées, offrant ainsi un aperçu clair de la fiabilité et de l'efficacité de l'interpréteur dans le traitement des commandes SQL.

Conclusion générale

Le développement de l'interpréteur de requêtes SQL avec les outils Flex et Bison représente une étape significative dans la création d'outils puissants pour la manipulation et l'accès aux données dans les bases de données relationnelles. Notre projet a démontré la valeur de ces outils dans la conception et l'implémentation d'un interpréteur robuste et performant, capable de gérer efficacement les requêtes de manipulation et d'accès à une base de données.

L'utilisation de Flex pour l'analyseur lexical a permis une segmentation précise des requêtes en tokens lexicaux significatifs, tandis que Bison a facilité la validation de la structure syntaxique des requêtes. De plus, l'intégration d'actions sémantiques a ajouté une couche supplémentaire de fonctionnalités pour améliorer la précision et la convivialité de l'interpréteur.

En explorant les différentes étapes de conception et d'implémentation, nous avons surmonté divers défis pour aboutir à un interpréteur fonctionnel et fiable. Ce projet a également souligné l'importance de la collaboration entre l'analyseur lexical et l'analyseur syntaxique pour garantir la cohérence et la précision des résultats.

En fin de compte, notre objectif est de fournir aux utilisateurs un outil flexible et efficace qui simplifie la manipulation et l'accès aux données dans diverses applications. Nous espérons que cet interpréteur SQL apportera une valeur ajoutée à la communauté des développeurs et des professionnels de la gestion de données, en répondant à leurs besoins croissants en matière de traitement de requêtes SQL.