Université Polytechnique de Bobo (UPB)
Ecole Supérieure d'Informatique (ESI)
Enseignant : **Dr Malo SADOUANOUAN**

BURKIAN-FASO
Unité-Progrès-Justice

RAPPORT DU PROJET DE TECHNIQUE DE COMPILATION

Par LOUE Boureima Arnaud et OUEDRAOGO Pascal élèves ingénieurs à l'Ecole Supérieure d'Informatique
Année : 2012-2013

PLAN

INTRODUCTION

A la faveur du cours de technique de compilation délivré par **Dr Malo SADOUANOUAN** un projet nous a été donné pour mettre en pratique les connaissances théoriques acquises.
En effet il nous a été demandé de concevoir un interpréteur de commandes SQL avec les grandes fonctionnalités suivantes :

- Fonctionnalité de création de table: génération d'un fichier XML
- Fonctionnalité de modification de la structure de la table
- Fonctionnalité d'insertion de tuples dans une table
- Fonctionnalité de mise à jour de tuples dans une table
- Fonctionnalité de suppression de tuples dans une table
- Fonctionnalité de sélection de tuples dans une table ou dans deux à plusieurs tables : les sélections simples, les jointures et jointures multiples.

Le développement sera fait en utilisant les outils flex et bison, dans leur version attribuée au C.

I. LES FONCTIONALITES DE L'INTERPRETEUR

Les fonctionnalités de l'application ont été développées conformément aux exigences du cahier de charges. Cependant beaucoup d'autres fonctionnalités non mentionnées dans le cahier de charges ont été ajoutées. Ce mini interpréteur SQL que nous avons appelé shellSQL, nous pouvons dire qu'il est à sa version 1.1 fort de sa sémantique et tous les contrôles qui y sont apportés.
Son lexique, sa syntaxe, et sa sémantique sont très proches de SQL.

1. La commande CREATE TABLE

NB : La commande CREATE TABLE créée une table correspond à un fichier xml avec la structure suivante :

```
<?xml version="1.0"  encoding="utf-8"?>
<table name= "nom_table">
<struct>
<attribut1 type="le_type" contrainte="la_contrainte"/>
<attribut2  type="le_type" contrainte="la_contrainte"/>
…
…
< attributN type="le_type" contrainte="la_contrainte"/>
</struct>
</table>
```

a. Syntaxe

Sa syntaxe est :
CREATE TABLE nom_table(attribut1 type1 [contrainte1] [,attribut2 type2 [contrainte2]]..... [,attributN typeN [coontrainte1]]) ;

NB :

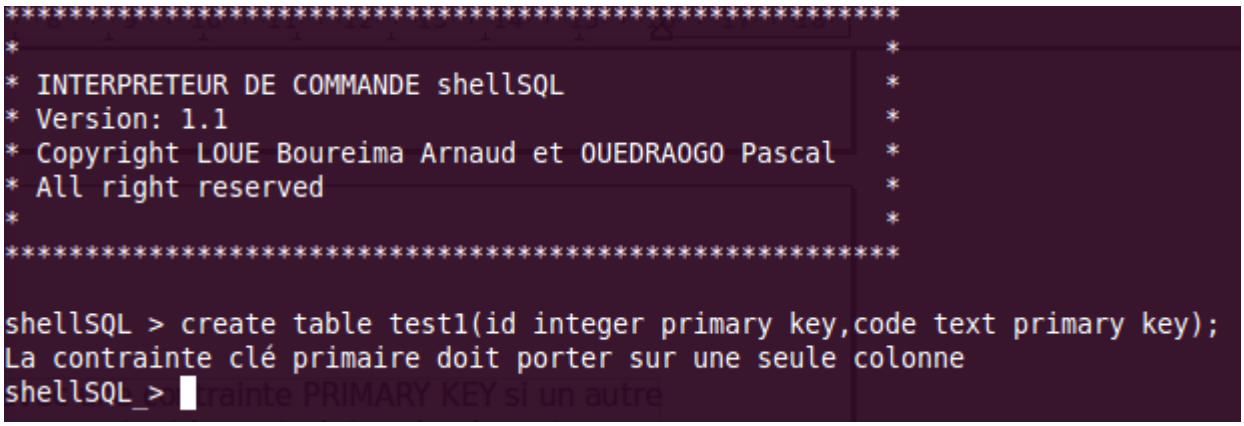
- **Les valeurs entre crochets sont optionnelles**
- **Le nom de la table et des attributs peuvent contenir des lettres et des chiffres et des caractères spéciaux (@_$...) mais commencent par une lettre.**
- **Les types peuvent être : INTEGER, TEXT, BOOLEAN, NUMBER.**
- **Les contraintes peuvent être : PRIMARY KEY, FOREIGN KEY.**

b. Sémantique et Contrôles

La création d'une table doit respecter une sémantique et exige un certain nombre de contrôles:

- Si une table porte déjà le même nom que la table que vous voulez créer alors un message vous sera affiché: Table déjà existante.
- La contrainte PRIMARY KEY porte sur au plus un attribut.

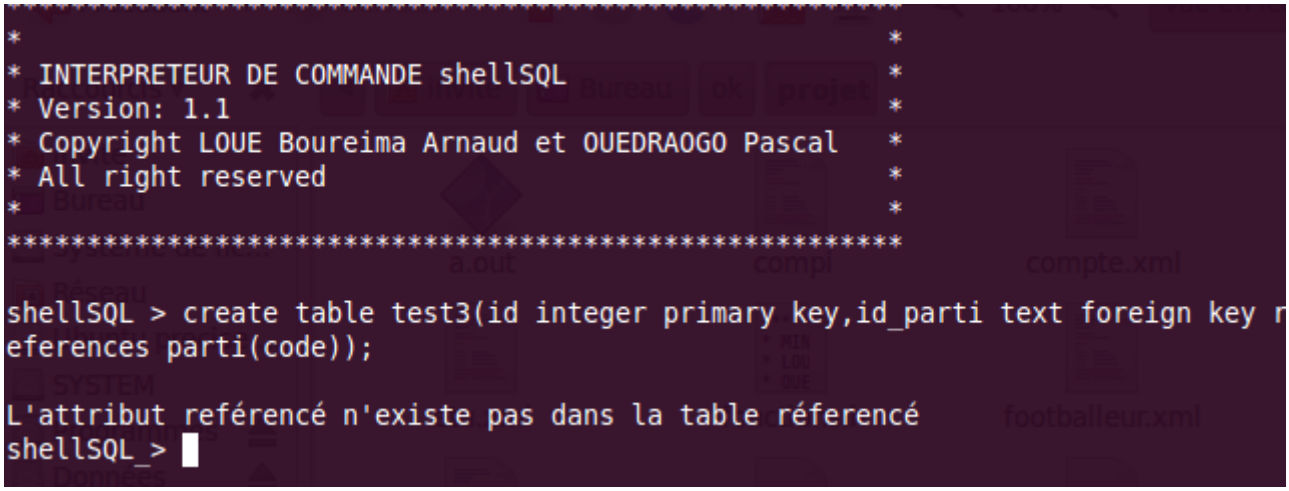
Example: create table test1(id integer primary key,code text primary key) ;
Le message d'erreur suivant sera affiché : La contrainte clé primaire doit porter sur une seule colonne.



- On ne peut pas créer des champs de même nom dans la même table.
- L'utilisation de la contrainte FOREIGN KEY se fait de la façon suivante :

create table test1(id integer primary key,code_foot text foreign key references footballeur(code)) ;

- La table référencée c'est à dire footballeur doit exister, sinon un message d'erreur sera affiché : La table référencée n'existe pas!
- L'attribut de la table référencée c'est à dire code doit aussi exister sinon un message d'erreur sera affiché : L'attribut référencée n'existe pas !



c. **Exemples**

create table compte(id integer text primary key,solde number);
create table etu(marti text primary key,nom text,prenom text,sexe boolean,age integer,id_ecole integer foreign key references ecole(id)) ;

2. La commande RENAME

Elle permet de renommer une table.

a. **Syntaxe**

RENAME ancien_nom TO nouveau_nom ;

b. **Les Sémantique et Contrôles**

Si la table à renommer n'existe pas un message d'erreur est renvoyé : « Table inexistante »

c. **Exemples**

rename etu to etudiant;
rename etudiant to etu;

3. La commande INSERT INTO

a. **La syntaxe**

Sa syntaxe est :
INSERT INTO nom_table VALUES(valeur1 [,valeur2]...[,valeurN]) ;
Les valeurs à insérer peuvent prendre les valeurs suivantes :

- Des réels : entiers relatifs et décimaux.
- Des chaines de caractères notées avec simple quote ('').
- Des booléens : les valeurs 0 ou 1.

b. **Les Sémantique et Contrôles**

- Tous les types ont été contrôlés : on ne peut insérer une valeur ne correspondant pas au type du champ lors de la création de la table.

Par exemple dans un champ de type BOOLEAN on ne peut insérer que 0 ou 1, dans un champ de type INTEGER on ne peut insérer que des entiers.

```
**********************************************************
*                                                        *
* INTERPRETEUR DE COMMANDE shellSQL                      *
* Version: 1.1                                           *
* Copyright LOUE Boureima Arnaud et OUEDRAOGO Pascal     *
* All right reserved                                     *
*                                                        *
**********************************************************

shellSQL > insert into etu values(8,'GOHOU','Michel',3,67,2);
veuillez inserer une valeur Booléen(0 ou 1) dans le champs n° 4

shellSQL_>
```

– On ne peut insérer deux valeurs identiques dans un champ portant une contrainte PRIMARY KEY.

```
**********************************************************
*                                                        *
* INTERPRETEUR DE COMMANDE shellSQL                      *
* Version: 1.1                                           *
* Copyright LOUE Boureima Arnaud et OUEDRAOGO Pascal     *
* All right reserved                                     *
*                                                        *
**********************************************************

shellSQL > insert into compte values('00A',30000.00);
Violation de la contrainte PRIMARY KEY !!
shellSQL_>
```

- On ne pas insérer une valeur dans un champ externe (champ ayant une contrainte foreign key) que si la valeur existe dans le champ qu'il fait référence dans table référencée.

```
**********************************************************
*                                                        *
* INTERPRETEUR DE COMMANDE shellSQL                      *
* Version: 1.1                                           *
* Copyright LOUE Boureima Arnaud et OUEDRAOGO Pascal     *
* All rights reserved                                    *
*                                                        *
**********************************************************

shellSQL > insert into test values(7,'00F');
La valeur '00F' n'existe pas dans l'attribut id de la table référencée parti
shellSQL_>
```

– On ne peut insérer des valeurs NULL dans un champ portant une contrainte PRIMARY KEY.

```
**********************************************************
*                                                        *
* INTERPRETEUR DE COMMANDE shellSQL                      *
* Version: 1.1                                           *
* Copyright LOUE Boureima Arnaud et OUEDRAOGO Pascal     *
* All rights reserved                                    *
*                                                        *
**********************************************************

shellSQL > insert into compte values(NULL,'00B');
Le champ n° 1 est la clé primaire et est de type TEXT : Veuillez inserer une cha
ine de caractère et différent de NULL|null
shellSQL_>
```

c. **Exemples**

insert into compte values('00A',-20000.00);
insert into etu values(1,'LOUE','B Arnaud',1,22,1) ;
insert into etu values(8,'GOHOU','Michel',3,67,2);

4. La commande SELECT

Elle permet de sélectionner des champs avec éventuellement des conditions.

a. **Syntaxe**

NB : | signifie OU

- **Syntaxes sans jointure**

Les syntaxes sont les suivantes :
SELECT * | attribut1 [,attribut2]...[,attributN] from nom_table;
SELECT * | attribut1 [,attribut2]...[,attributN] from nom_table WHERE
attribut op_Rel valeur ;
op_Rel : =|<|> | !=
Valeur : il s'agit d'une valeur insérer
NB : < et > sont utilisés que si le type du champ est INTEGER ou NUMBER.

```
shellSQL_> select * from compte where id<'00A';
le type de id est :text
Les operateurs < ou > ne s'applique que sur une colonne de type integer ou numbe
r
shellSQL_>
```

- **Syntaxe avec simple jointure**

SELECT table1.attribut1 [,table1.attribut2]... [,table1.attributN] [,table2.attribut1]...[table2.attributN] FROM table1 join table2 WHERE table1.attributN=table2.attributN ;

- **Syntaxe avec jointures multiples**

SELECT table1.attribut1,table1.attribut1 [,table1.attribut2]... [,table1.attributN] [,table2.attribut1]...[table2.attributN] [,table3.attribut1]... [,table3.attributN] FROM table1,table2,table3 WHERE table1.attributN=table2.attributN AND table1.attributN=table3.attributN;

b. **Sémantique et Contrôles**

- Si un champ à sélectionner n'existe pas, aucune donnée n'est retournée.
- Si le champ spécifié dans la clause where n'existe pas un message d'erreur est affiché.

- Si le champ spécifié dans la clause where ne contient pas la valeur recherchée alors aucune ligne n'est retournée.

c. Exemples

- **Sans jointure :**

Select * from compte where solde!=0.0;
select id,solde from compte where solde<0;
select * from etu where nom='LOUE' ;

- **Simple jointure :** select depute.nom,depute.id_parti,parti.id,parti.nom from depute join parti where depute.id_parti=parti.id ;

```
shellSQL >  select depute.nom,depute.id_parti,parti.id,parti.nom from depute joi
n parti where depute.id_parti=parti.id ;
+-----------------+-----------------+-----------+-----------+
| depute.nom      | depute.id_parti | parti.id  | parti.nom |
+-----------------+-----------------+-----------+-----------+
| 'LOUE'          | '00A'           | '00A'     | 'UPC'     |
+-----------------+-----------------+-----------+-----------+
| 'OUEDRAOGO'     | '00B'           | '00B'     | 'ADF-RDA' |
+-----------------+-----------------+-----------+-----------+
| 'KIMA'          | '00C'           | '00C'     | 'CDP'     |
+-----------------+-----------------+-----------+-----------+

shellSQL_>
```

- **Jointure multiple :** select depute.nom,parti.nom,region.nom from depute,parti,region where depute.id_parti=parti.id and depute.id_region=region.id ;

```
shellSQL_>  select depute.nom,parti.nom,region.nom from depute,parti,region where depute.id_parti=parti.id and depute.id_region=region.id ;
+-----------------+-----------+--------------+
| depute.nom      | parti.nom | region.nom   |
+-----------------+-----------+--------------+
| 'LOUE'          | 'UPC'     | 'Hauts-Bassins' |
+-----------------+-----------+--------------+
| 'OUEDRAOGO'     | 'ADF-RDA' | 'Centre'     |
+-----------------+-----------+--------------+
| 'KIMA'          | 'CDP'     | 'Centre-Sud' |
+-----------------+-----------+--------------+

shellSQL_>
```

5. La commande DELETE

Elle permet de supprimer un ou plusieurs enregistrements.

a. Syntaxe

DELETE FROM nom_table WHERE nom_attribut op_Rel valeur ;

b. Sémantique et Contrôles

- Si la valeur spécifiée dans la clause WHERE n'existe pas aucune ligne n'est retournée, le programme ne plante pas.
- Si l'attribut nom_attribut n'existe pas dans la table nom_table le programme ne plante, un message d'erreur est envoyé.

c. Exemples

delete from footballeur where nom='Ronaldo' ;

delete from compte where solde>0;

6. La commande ALTER TABLE

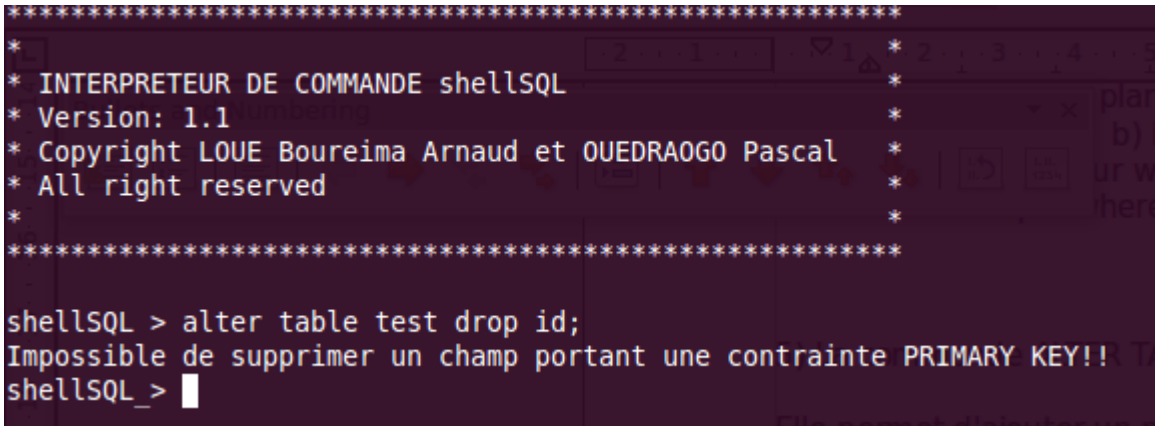
Elle permet de modifier la structure d'une table.

a. Syntaxes

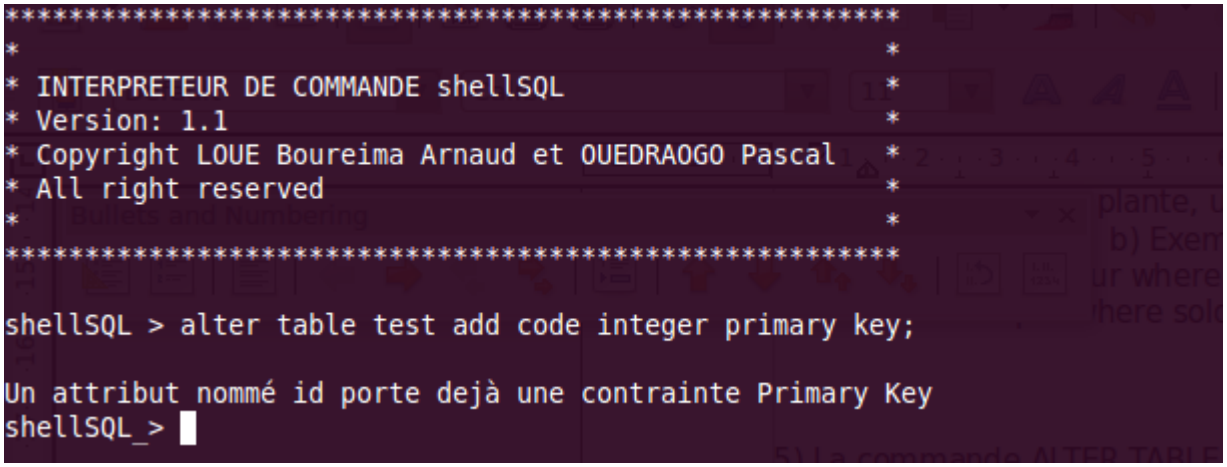
Elle permet d'ajouter un nouveau champ ou de supprimer un champ dans une table.
Pour l'ajout la syntaxe est la suivante :
ALTER TABLE nom_table ADD nom_nouveau_champ type [contrainte] ;
Pour la suppression :
ALTER TABLE nom_table DROP nom_champ_a_supprimer;
NB : Les types et les contraintes sont les mêmes q
ue dans la commande CREATE TABLE.

b. Sémantique et Contrôles

- On ne peut supprimer un champ portant une contrainte PRIMARY KEY



- On ne peut insérer un champ avec une contrainte PRIMARY KEY si un autre champ l'a déjà : l'unicité du champ portant la contrainte primaire est vérifiée.



- Le contrôle de la contrainte FOREIGN KEY est fait comme dans la commande CREATE TABLE.
- On ne peut ajouter un champ déjà existant.

c. Exemples

- alter table depute add ethnie text ;

Si la table depute contient des données alors les valeurs du nouveau seront par défaut à NULL.

- alter table depute drop ethnie ;

7. La commande UPDATE

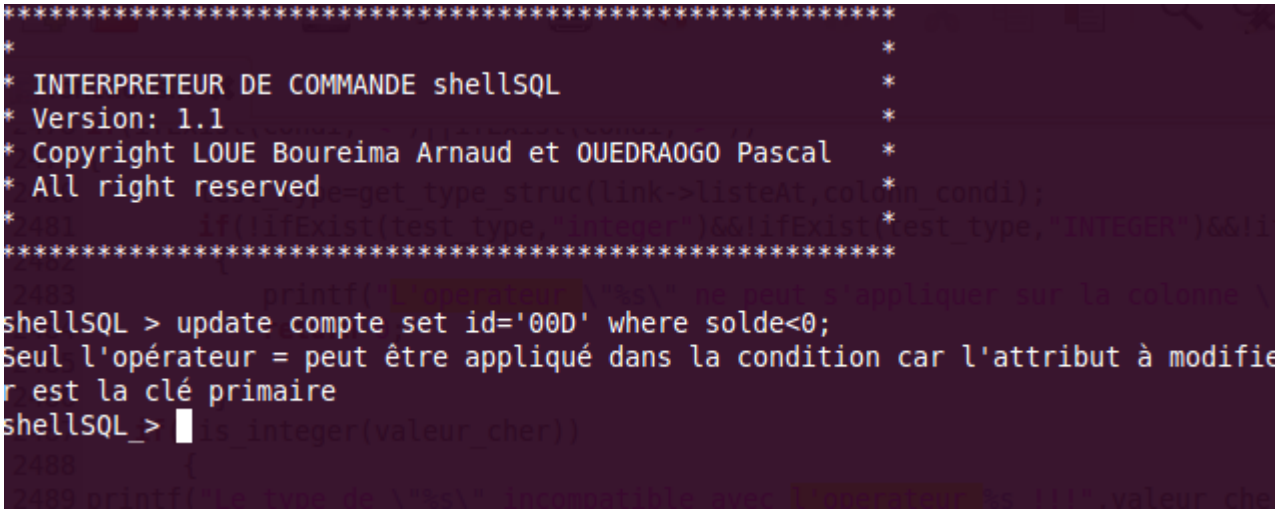
Elle permet de mettre à jour un ou plusieurs enregistrements dans une table.

a. Syntaxe

UPDATE nom_table SET nom_attributN=valeurN WHERE nom_attributN op_Rel valeurN ;

b. Sémantique et Contrôles

- Si l'attribut ou la valeur spécifiée dans la clause WHERE n'existe pas alors aucune ligne n'est retournée, l'application ne plante pas.
- Étant donné qu'UPDATE fait des mises à jour c'est à dire des insertions, les Sémantique et Contrôles au niveau de l'insertion sont également faits ici. (l'unicité des valeurs du champ de contrainte PRIMARY KEY...)
- Si le champ à mettre à jour est la clé primaire alors la condition du WHERE est forcément une condition d'égalité pour que l'unicité des valeurs de la clé primaire soit toujours respectée.



c. Exemples

- update footballeur set nom='PEDRO RODRIGEZ' where code='00G';
- update compte set solde=0 where solde<0;

8. La commande DROP TABLE

Elle permet de supprimer une table.

a. Syntaxe

DROP TABLE nom_table ;

b. Sémantique et Contrôles

Si la table à supprimer, un message d'erreur est renvoyé : « Table inexistante »

c. Exemples

drop table etudiant ;

9. La commande DESC

Elle permet de décrire la structure d'une table.

a. **Syntaxe**

DESC nom_table ;

b. **Sémantique et Contrôles**

Si la table dont on veut donner la structure n'existe pas un message d'erreur est renvoyé : « Table inexistante »

c. **Exemples**

desc etu ;

10. Les autres commandes

a. **La commande EXIT**

Elle permet de quitter le programme c'est à dire l'interpréteur de commande.
Sa syntaxe est : exit ;

b. **La commande CLEAR**

Elle permet d'effacer le termimal.
Sa syntaxe est : clear ;

c. **La commande HELP**

Elle permet d'avoir de l'aide sur l'utilisation d'une commande ; sa syntaxe est :
HELP nom_commande ;
Exemples :
help delete ;
help create;
Le resultat est comme suit pour help delete ;

```
* INTERPRETEUR DE COMMANDE shellSQL                    *
* Version: 1.1                                         *
* Copyright LOUE Boureima Arnaud et OUEDRAOGO Pascal   *
* All right reserved                                   *
*                                                      *
********************************************************

shellSQL > help delete;
*********************  manuel de delete ****************************

pour supprimer un enregistrement on tape la commande suivante:

delete from nom_table where condition;

pour la condition on a l'egalite (=) , different (!=) ,superieur (>) et inf
r (<)
exemple :
delete from personne where age=23;
delete from personne where age<30;
delete from personne where nom!='LOUE';
*******************************************************************************
```

II. AUTRES PRISES D'ECRAN

```
*************************************************************
*                                                           *
* INTERPRETEUR DE COMMANDE shellSQL                         *
* Version: 1.1                                              *
* Copyright LOUE Boureima Arnaud et OUEDRAOGO Pascal        *
* All right reserved                                        *
*                                                           *
*************************************************************

shellSQL > select * from ecole;
| id                    | nom_ecole
_____________________________________
| 1                     | 'ESI'
_____________________________________
| 2                     | 'ISNV'
_____________________________________
| 3                     | 'ISEA'
_____________________________________
| 4                     | 'IDR'
_____________________________________

4 ligne(s) trouvée(s) !!

shellSQL_>
```

Ecran 1 : Sélection de l'ensemble des écoles

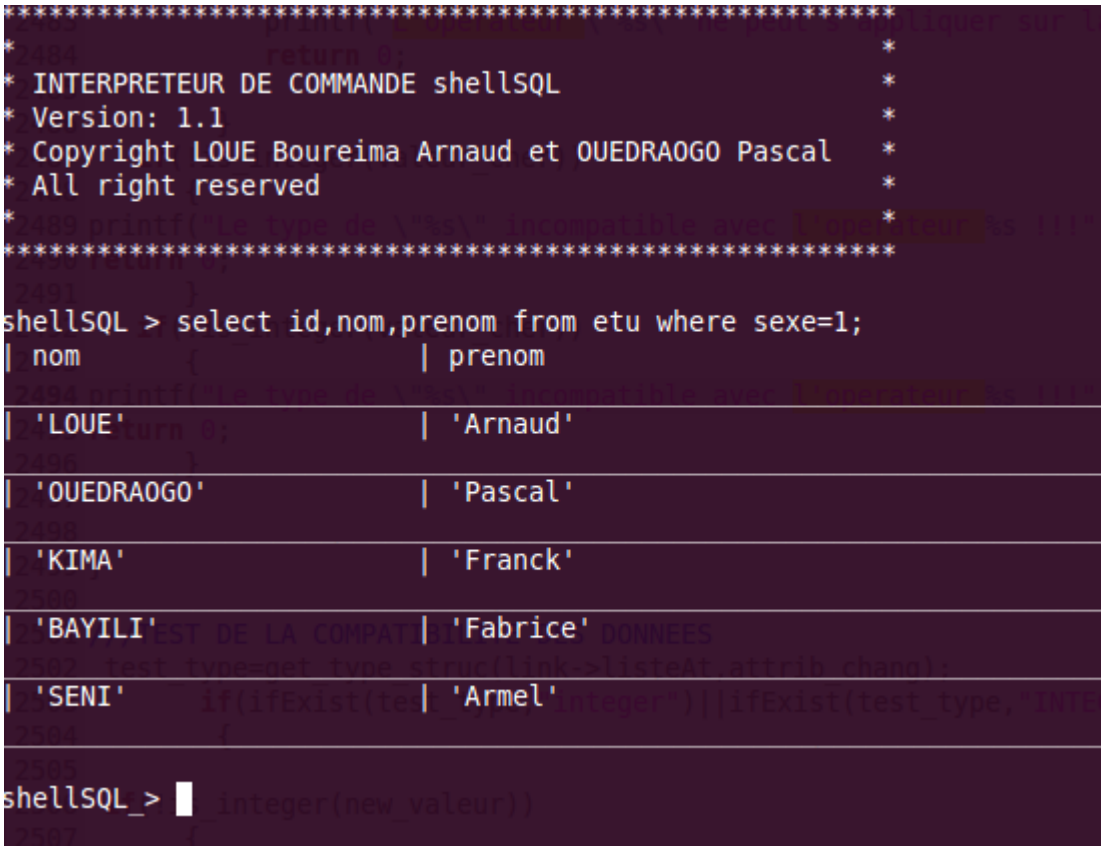
```
*******************************************************
*                                                     *
* INTERPRETEUR DE COMMANDE shellSQL                   *
* Version: 1.1                                        *
* Copyright LOUE Boureima Arnaud et OUEDRAOGO Pascal  *
* All right reserved                                  *
*                                                     *
*******************************************************

shellSQL > select * from etu;
 matri                | nom                  | prenom               | sexe                 | age                  | id_ecole
 1                    | 'LOUE'               | 'Arnaud'             | 1                    | 22                   | 1
 2                    | 'OUEDRAOGO'          | 'Pascal'             | 1                    | 24                   | 1
 3                    | 'KIMA'               | 'Franck'             | 1                    | 22                   | 1
 4                    | 'BAYILI'             | 'Fabrice'            | 1                    | 23                   | 3
 5                    | 'SAVADOGO'           | 'MADINA'             | 0                    | 24                   | 4
 6                    | 'SENI'               | 'Armel'              | 1                    | 23                   | 2
 7                    | 'NANA'               | 'Sonia'              | 0                    | 23                   | 2

 ligne(s) trouvée(s) !!

shellSQL_>
```

Ecran 2 : Sélection de l'ensemble des étudiants



Ecran 3 : Sélection des étudiants de sexe masculin

III. MISE EN OEUVRE TECHNIQUE

Les outils utilisés dans le cadre de ce projet sont : le couple Flex/Bison dans leur version attribué au C et Valgring qui est un outil permettant de détecter des fuites de mémoire et erreurs de segmentation.
Flex est un analyseur lexical c'est à dire il est chargé de vérifier si un mot fait partie du langage ou non.
Bison quant à lui est un analyseur syntaxique c'est à dire il est chargé de vérifier que la syntaxe (l'enchainement des mots) est bon.
Bison fait de l'analyse ascendante c'est à dire du dernier mot jusqu'à l'axiome.
Nous avons travaillé essentiellement sur 4 fichiers que nous avons nommé lexique.l, syntaxe.y, defines.h, fonctions.h.

1. Le fichier lexique.l

Il contient le lexique c'est à dire l'ensemble des mots qui forment le langage. Si un mot n'y figure pas alors il ne sera pas reconnu par l'analyseur lexical.
Le fichier se présente comme suit :

```
%{

/*inclusion ou déclaration de variables C*/

%}

/*descriptions des expressions régulières constituant les mots du langage*/
```

%%

/*partie au niveau de laquelle on renvoie les mots du langage vers le fichier
syntaxe.y ie vers l'analyseur syntaxique*/

%%
/*FIN*/

Pour notre interpréteur le langage défini contient dans ce fichier plus de 80 mots et expressions régulières.

2. Le fichier syntaxe.y

Il contient la définition de la grammaire du langage. Ce fichier sera parcouru par l'analyseur syntaxique c'est-à-dire Bison.
Il contient la fonction main qui est la fonction principale dans laquelle l'analyseur syntaxique est appelé par : yyparse().
Le fichier se présente comme suit :

%{

/*déclaration et inclusion C*/

%}

/*déclaration des terminaux par %token et des non-terminaux et type par %type*
Ex : %token INTEGER*/

%%

/*partie de la définition de la grammaire à travers des dérivations*/
/*C'est ici que les fonctions définies seront appelées*/
/*Exemple : */
/*

```
cmdSQL: cmd_instr_CREATE
      | cmd_instr_ALTER
      | cmd_instr_INSERT
      | cmd_instr_UPDATE
      | cmd_instr_SELECT
      | cmd_instr_DELETE
      | cmd_instr_DROP
      | cmd_instr_RENAME
      | cmd_instr_DESC
      | QUIT FIN {   exit(1); YYACCEPT;   }

      | OP_CLEAR FIN { system("clear");printf("\nshellSQL_> ");yyparse(); }
      | CMD_HELP IDENT FIN  {  help($2);printf("\nshellSQL_> ");yyparse();}
      | error {yyerrok;yyclearin;}
      ;
```

*/
%%

/*redéfinitions et définitions de fonctions C notamment la fonction main ()*/

/*FIN*/

3. Le fichier defines.h

C'est un fichier classique C qui contient les définitions de structures de données dynamiques (listes chainées…) et les prototypes de fonctions.
Par exemple la structure de données représentant un champ est :

```
typedef struct Element Element;
struct Element {
char att[TAILLE_DONNEE];///nom du champ
char typ[TAILLE_DONNEE];///type du champ
char contrainte[TAILLE_DONNEE];//contrainte
Element *suivant;//element suivant
};
```

4. Le fichier fonctions.h

Ce fichier comme son l'indique contient la définition des fonctions, l'implémentation de la sémantique du langage se fait donc ici.

IV. PERSPECTIVES D'EVOLUTION VERS LA VERSION 1.2

To be continued...

- Extension du langage à d'autres requêtes : CREATE INDEX…
- Ajout du type VARCHAR, DATE
- Ajout de la jointure naturelle (NATURAL JOIN)
- …