Université Aboubekr BELKAID
كلية العلوم – تيجاني هدام
Faculté des Sciences
قسم الإعلام الآلي
Département d'informatique

Module : Compilation **Année universitaire 2021-2022**
Classe : L3- Informatique
Réalisé par : Mr MERZOUG Mohamed
Mr ETCHIALI Abdelhak

TP 4
Analyse Lexicale

1- Objectif :

L'objectif de ce TP est de développer un analyseur lexical d'un sous langage du SQL. Le résultat produit par cet analyseur (**Suite des unités lexicales**) sera utilisé par l'analyseur syntaxique **LL1** dans le TP suivant.

2- Définition :

L'analyse lexicale est la 1ère étape de la compilation. Elle permet de transformer le code source en **une suite d'U.L** qui peuvent être :

- Mots clés ou réservés : (en SQL : select, distinct, from, where, …)
- Identificateurs : (en SQL : nom de tables ou de champs)
- Nombres : (entier, réel, …)

L'analyseur lexical doit assurer les tâches suivantes :

- Lecture caractère/caractère du code source.
- Élimination des informations inutiles (dans notre cas : les espaces, les tabulations, …).
- Identification des U.L. (type de chaque UL, ligne) :
  - La spécification des U.L se fait par des expressions régulières.
  - La reconnaissance des U.L. se fait via l'A.E.F relatif à l'expression régulière.
- Détection des erreurs lexicales (exemple : caractère n'appartenant pas à l'alphabet du langage, nombre illégal…).

3- Etapes à suivre :

3.1- *Définition du sous langage SQL :*

L'analyseur lexical du sous langage souhaité est un AEFD (Automate à Etats Fini Déterministe) permettant de reconnaitre les unités lexicales (**terminaux**) suivantes :

a) **Identifiant :** L'identificateur d'un champ et les noms de tables sont analogues aux noms de variables dans les langages de programmation.

La forme générale d'un identifiant valide est exprimée par l'expression régulière suivante :

r ::=\w[\d\w_]*

Les opérateurs logiques ainsi que les mots clés sont des sous langages du langage des identifiants. Ainsi, si un token est conforme à l'expression régulière d'un identifiant, celui-ci est d'abord comparé avec la liste des mots clés et des opérateurs logiques. Si le token ne correspond à aucun des éléments de ces derniers, celui-ci est reconnu comme un identifiant. Sinon, celui-ci est reconnu comme étant le mot clé (ou l'opérateur logique) correspondant.

b) **Nombres:** Les nombre entiers / réels valides sont représentés par l'expression régulière suivante :

r ::=(([\d]+) | ([\d]*. [\d]+))

c) **Accolade ouvrante / fermante :** les accolades ouvrantes et fermante doivent être reconnues

d) **Opérateurs Arithmétiques :** r ::= [+-*/]

e) **Opérateurs relationnels :** r ::= (= | <= | >= | < | >)

f) **Séparateur Virgule**

g) Séparateur Point-virgule

h) Opérateur Logique : and – or

i) Mot clés: {Select, Count, Distinct, From, Where}

3.2- Reconnaissance des U.L. :

A partir de l'expression régulière qui définit chaque U.L, on donne son AEFD correspondant, puis réunir tous les automates dans un seul automate globale, qui commence par l'état initial 0. A partir de cet état là, on commence à lire la requête SQL caractère par caractère.

Par exemple : La lecture d'une lettre nous ramène à la reconnaissance soit d'un identificateur, d'un mot clé ou d'un opérateur logique, alors que la lecture d'un chiffre nous ramène à la reconnaissance d'un nombre entier ou réel, etc…..

Étendre cet automate pour qu'il ignore les espaces et signale les erreurs lexicales telles que la lecture d'un nombre erroné ou la lecture d'un caractère non autorisé.

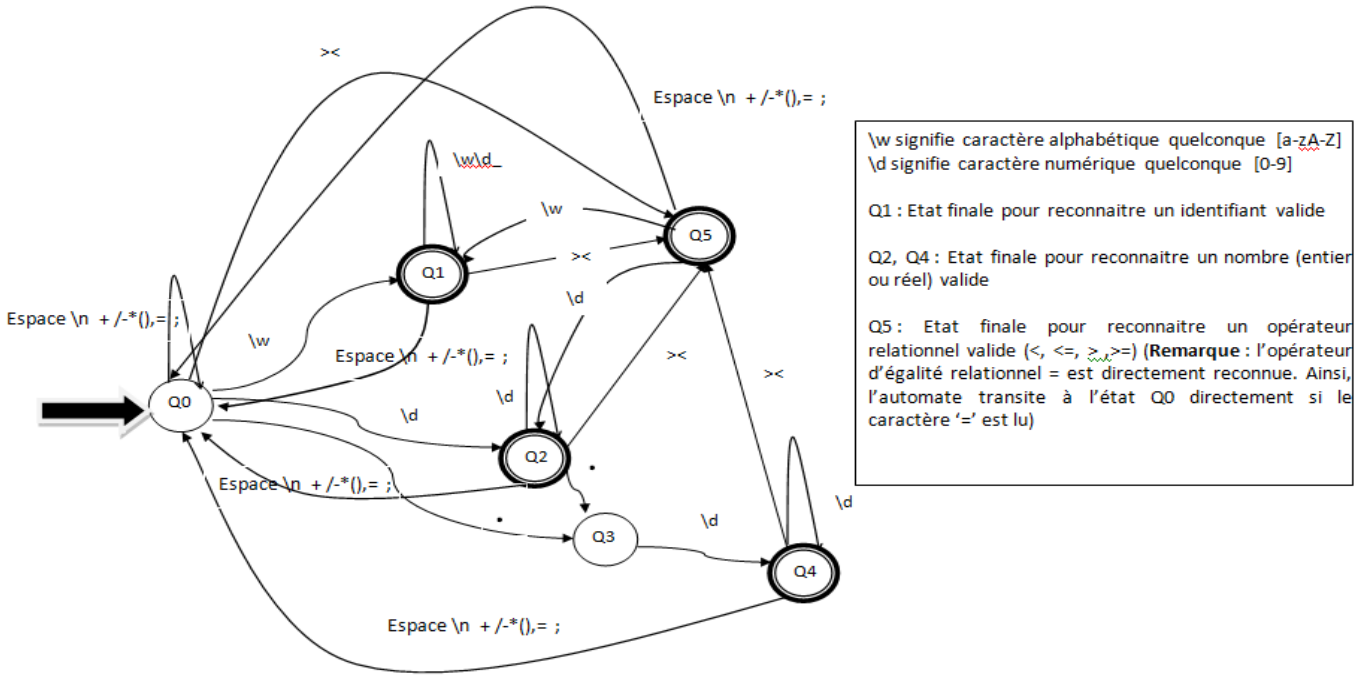


Figure1 Automate global du sous langage SQL

3.3- Attribution des codes aux U.L. :

Chaque **unité lexicale** est représentée par **un code**. Il est préférable d'utiliser un intervalle contigu pour représenter les codes des unités lexicales. Par exemple :

Unité Lexicale	Code (Type_UL)
IDENTIFIANT_UL	260
NOMBRE_UL	261
OPERATEUR_ARETHMETIQUE	262
SEPARATEUR_VIRGULE	263
SEPARATEUR_POINT_VIRGULE	264
ACCOLADE_OUVRANTE	265
ACCOLADE_FERMANTE	266
OPERATEUR_RELATIONNEL	267
OPERATEUR_LOGIQUE	268
SELECT_KW	269
FROM_KW	270
WHERE_KW	271
COUNT_KW	272
DISTINCT_KW	273
EPSILON	111
FIN_SUITE_UL (#)	333
ERREUR	-99

3.4- Structure d'une U.L. :

Une unité lexicale est représentée par la structure suivante :

```
typedef struct Unite_Lexicale {
                        char   Lexeme[20];
                        int    Code;
                        int    Ligne;
                        struct Unite_Lexicale* Suivant;
} UL;
```

Remarque :

A la fin de l'analyse lexicale, il faut retourner la liste chainée des U.L, appelée suite des U.L

3.5- Algorithme :

a- Lire la requête SQL (code source) à partir d'un fichier texte.

b- Analyser le code source caractère par caractère en utilisant l'automate global.

c- Une unité lexicale est reconnue si l'automate est dans l'état final de celle-ci et un caractère séparateur est lu.

d- Chaque U.L reconnue par l'automate sera insérée à la fin de la liste chainée des suite U.L et l'automate transite à l'état initial.

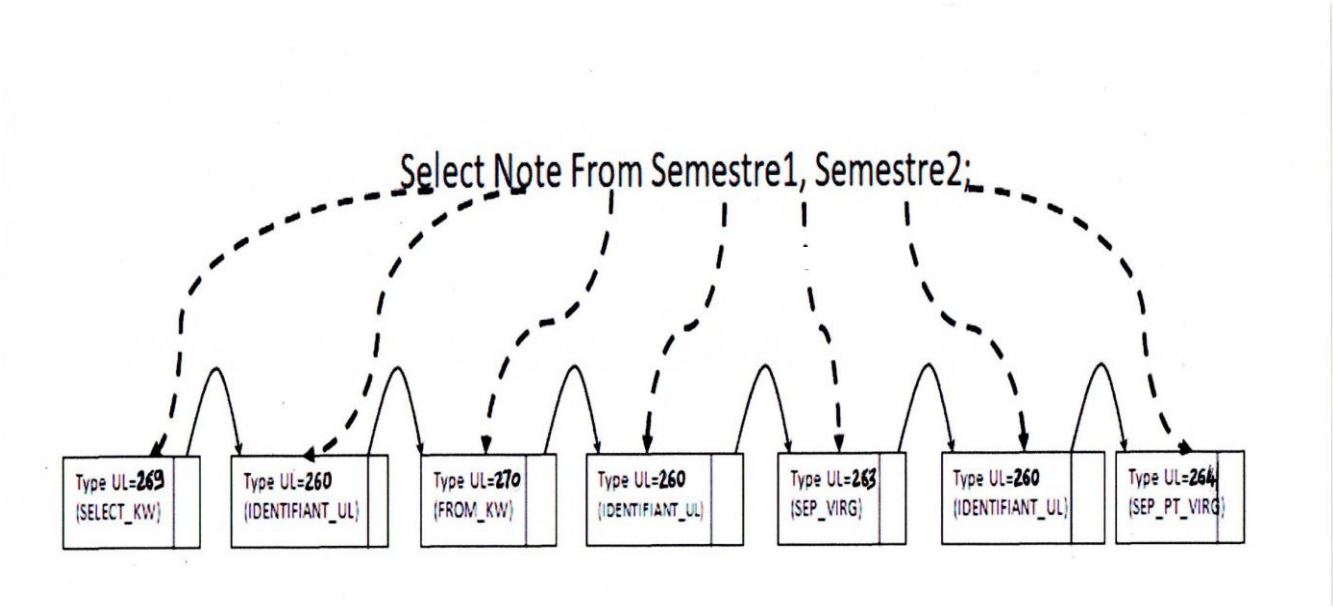
e- En cas où une erreur lexicale est détectée, l'analyseur doit avorter le traitement et retourne la position et la ligne du caractère où l'erreur a été détectée.

f- Si l'analyse lexicale se termine avec succès, insérer le « # » à la fin de la suite U.L.

Exemple : la requête :

Select Note From Semestre1, Semestre2 ;

Est transformée vers la suite d'unité lexicale suivante :



Exemple d'exécution :

1- ***Analyse lexicale avec succès :***

```
                LECTURE ET AFFICHAGE DU CODE SOURCE LIGNE PAR LIGNE

select distinct age, salaire, tel

from emp

where slaire >= 30000

or age >25 and age < 50;

                Afficher la Suite des Unites Lexicales

                LEXEME = [select]       CODE = [269]    LIGNE = [1]
                LEXEME = [distinct]     CODE = [273]    LIGNE = [1]
                LEXEME = [age]  CODE = [260]    LIGNE = [1]
                LEXEME = [,]    CODE = [263]    LIGNE = [1]
                LEXEME = [salaire]      CODE = [260]    LIGNE = [1]
                LEXEME = [,]    CODE = [263]    LIGNE = [1]
                LEXEME = [tel]  CODE = [260]    LIGNE = [1]
                LEXEME = [from] CODE = [270]    LIGNE = [2]
                LEXEME = [emp]  CODE = [260]    LIGNE = [2]
                LEXEME = [where]        CODE = [271]    LIGNE = [3]
                LEXEME = [slaire]       CODE = [260]    LIGNE = [3]
                LEXEME = [>]    CODE = [267]    LIGNE = [3]
                LEXEME = [=]    CODE = [267]    LIGNE = [3]
                LEXEME = [30000]        CODE = [261]    LIGNE = [3]
                LEXEME = [or]   CODE = [268]    LIGNE = [4]
                LEXEME = [age]  CODE = [260]    LIGNE = [4]
                LEXEME = [>]    CODE = [267]    LIGNE = [4]
                LEXEME = [25]   CODE = [261]    LIGNE = [4]
                LEXEME = [and]  CODE = [268]    LIGNE = [4]
                LEXEME = [age]  CODE = [260]    LIGNE = [4]
                LEXEME = [<]    CODE = [267]    LIGNE = [4]
                LEXEME = [50]   CODE = [261]    LIGNE = [4]
                LEXEME = [;]    CODE = [264]    LIGNE = [4]
                LEXEME = [#]    CODE = [999]    LIGNE = [5]
```

2- ***Analyse lexicale échouée (erreur) :***

```
                LECTURE ET AFFICHAGE DU CODE SOURCE LIGNE PAR LIGNE

select distinct age, salaire, tel

from emp?


Erreur Lexicale, Caractere ? Non Reconnu Ligne 2
```

```
                LECTURE ET AFFICHAGE DU CODE SOURCE LIGNE PAR LIGNE

select distinct age, salaire, tel

from emp

where slaire >= 30j000


Erreur Lexicale, Caractere j Non Reconnu Ligne 3
```