REPUBLIQUE ALGERIENE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR DE LA RECHERCHE
SCIENTIFIQUE ET DE LA TECHNOLOGIE
UNIVERSITE SAAD DAHLEB BLIDA 1

PROJET DE FIN D'ETUDES

Présenté au Département de l'informatique

Conception d'un traducteur de requêtes écrite en langage naturel vers le SQL

Réaliser Par :
FERROUKHI Soumia & FODIL Selma

Sous la direction de :
Mr. CHERIF Zahar

Soutenu devant la commission de jury :

- Mr HAMMOUDA Président
- Mm FERDI Examinatrice

ANNEES UNIVERSITAIRE : 2020 / 2021

Remerciements

Nous tiens à remercier nos parents, dont le soutien, l'optimisme et la confiance tout au long de nos études, et encore jusque dans la rédaction de ce dernier travail, qui a pour nous une importance toute particulière.

Nous voudrions présenter nos remerciements à notre directeur de mémoire : M.Cherif Zahar.nous voudrions également lui témoigner notre gratitude pour sa patience et son soutien qui nous ont été précieux pour mener à bien notre travail.

Un grand merci par ailleurs à Nos frères et sœurs.

Un grand merci aussi à nos maries pour leur soutien.

Nous voudrions remercier toutes les personnes qui, par leur soutien, leur conseil ou leur participation, ont contribué à la réalisation de ce mémoire.

.

Dédicace :

A ma chère mère :

Mon premier soutien dans la vie, mon héroïne, elle m'a soutenu pendant tout le temps dans mes moments les plus faibles et les moments forts, toutes mes études sont à cause d'elle et pour elle

A mon cher père :

Votre gentillesse vos efforts pour améliorer notre vie, vos soutien m'a fait devenir qui je suis maintenant.

A mon cher mari :

Mon meilleur ami, mon partenaire dans la vie.

A mon fils :

Mon DIAA, mon monde, je souhaite voir mon nom dans votre dédicace de pfe un jour.

A Mon frère : Hamza.

A Mes sueurs : Meriem, Maroua.

La petite Basma : la joie de notre maison.

Fodil Selma.

Sommaire

Liste des figures

Liste des tableaux

Résumé

L'objectif principal de ce projet est la conception et la réalisation d'une application bureau qui repose sur la traduction des requêtes écrites en langage naturel vers le langage SQL par un traitement automatique de la langue française.

La raison qui a poussé à traiter ce thème est la difficulté relative du langage SQL. Beaucoup de personnes éprouvent de grandes difficultés à écrire des requêtes en SQL dès que ces dernières ne sont pas simples.

Aussi a-t-on pensé à proposer –dans la mesure du possible- un traducteur qui permettrait de donner un équivalant SQL à toute requête el langage naturel et faciliter ainsi l'adressage de requêtes aux SGBD.

Mots clés : Traitement du Langage naturel, SQL, base de données, NLP.

Abstract

The main objective of this project is the design and production of a desktop application based on the translation of queries written in natural language into SQL language by automatic processing of the French language.

The reason which pushed to deal with this topic is the relative difficulty of the SQL language. Many people find it very difficult to write SQL queries when they are not simple.

So we thought to offer - as far as possible - a translator that would give an SQL equivalent to any query and natural language and thus facilitate the addressing of requests to DBMS.

Keywords: Natural language processing, SQL, database, NLP.

الملخص

الهدف الرئيسي من هذا المشروع هو تصميم وإنتاج تطبيق سطح المكتب بناءً على ترجمة الاستعلامات المكتوبة بلغة طبيعية إلى لغة SQL عن طريق المعالجة التلقائية للغة الفرنسية.

السبب الذي دفع للتعامل مع هذا الموضوع هو الصعوبة النسبية للغة SQL. يجد العديد من الأشخاص صعوبة بالغة في كتابة استعلامات SQL عندما لا تكون بسيطة.

لذلك اعتقدنا أن نقدم - قدر الإمكان - مترجمًا من شأنه أن يعطي SQL مكافئًا لأي استعلام ولغة طبيعية ، وبالتالي يسهل معالجة الطلبات إلى نظام إدارة قواعد البيانات.

الكلمات المفتاحية: معالجة اللغة الطبيعية ، SQL ، قاعدة البيانات,NLP .

Introduction Générale

1. Introduction :

Rares sont les applications qui se dispensent de manipuler des bases de données. En fait, c'est le meilleur moyen pour stocker des données en mémoire secondaire et pourvoir, ultérieurement les récupérer.

Aussi, les logiciels qui permettent de gérer des bases de données et que l'on désigne sous le nom de SGBD, acronyme pour Système de Gestion de Base de données, offrent, en plus des mécanismes de stockage et de restitution ainsi qu'une foule de fonctionnalités de grand intérêt, des langages de définition des données et de requêtes qui permettent d'interroger les bases de données pour avoir les informations recherchées.

2. Problématique :

Historiquement, plusieurs langages de requêtes ont vu le jour. Ils sont intimement liés à l'architecture de la base donnée. Il y a eu SQL, QBE, QUEL etc. Le langage qui s'est imposé avec le temps est le SQL. C'est un langage déclaratif, c'est-à-dire qu'on décrit ce qu'on veut avoir en retour sans se soucier du comment l'obtenir.

Le langage SQL est très puissant mais pose problème du fait que quelques fois, on ne sait comment décrire ce que l'on recherche. On retrouve dans des forums comme « Codes-sources » des étudiants/informaticiens en quête d'aide pour écrire des requêtes. La difficulté procède également du fait que tout besoin n'est pas forcément exprimable en SQL. Certains besoins nécessiteraient d'écrire des programmes comportant éventuellement plusieurs requêtes.

Sans une excellent maîtrise du SQL, il est difficile de prime abord de savoir si la requête désirée peut être exprimée en SQL et en second lieu, il n'est pas toujours très évident de l'écrire dès qu'elle devient assez complexe.

3. Objectifs :

L'objectif du présent projet et de fournir un traducteur qui permettrait de traduire des requêtes en langage naturel en requêtes SQL. L'intérêt serait :

- Professionnel car il permettrait à un programmeur d'aller vite en écrivant ses requêtes

- Pédagogique car il permettrait à un étudiant de faire des exercices et, quand il est bloqué, de se « faire aider » par l'application résultant du présent projet.

Chapitre 1 : État de l'art

1. Introduction :

L’un des premiers problèmes que soulève l’interrogation d’une base de données par un utilisateur n’ayant aucune connaissance dans ce domaine est qu’il ne connaît ni la structure ni le vocabulaire employé au sein de la base qu’il cherche à interroger. Pour cela, ce chapitre a pour objet de présenter quelques approches existantes et les solutions déjà trouvées pour « la traduction de requêtes écrites en langage naturel vers le langage SQL » d’une manière détaillée, en exposant leurs fonctionnalités.

Ce thème est toujours un sujet de travail et est en pleine croissance.

2. Naturel language processing (NLP) :

Nous ne pourrons pas parler de l'utilisation de la traduction du langage naturel vers le langage des bases de données SQL sans parler du traitement du langage naturel NLP, car ils sont liés les uns aux autres, C'est la base et la source sur laquelle un tel projet a été développé pour cela, il est nécessaire de se tenir sur les stations les plus en vue du développement de ce domaine.

2.1 C'est quoi NLP ?

C'est l'une des branches les plus importantes de l'intelligence artificielle concernée par le traitement de textes, sons, (..) naturels en différents langages machine.

Le NLP est le COLLABORATION de la linguistique et l'informatique également, elle aide les ordinateurs à comprendre, interpréter et utiliser les langages humains. "Le NLP permet aux ordinateurs de communiquer avec les gens, en utilisant un langage humain. Le traitement du langage naturel permet également aux ordinateurs de lire du texte, d'entendre la parole et de l'interpréter. La NLP s'inspire de plusieurs disciplines, dont la linguistique informatique et l'informatique, car elle tente de combler le fossé entre les communications humaines et informatiques"(A Brief History of Natural Language Processing (NLP)) ce qui est la base de notre projet. [1]

3. Natural Language Interfaces to Databases (NLIDB):

NLIDB est la zone la plus intéressante de traitement du langage naturel (NLP) est tout système qui est capable de convertir une requête écrite sur un langage native au langage de base de donnée (le SQL dans notre cas)

le NLIDB est d'après les premiers recherches des NLP et les plus réussis, interroger une base de donné dans la langue naturel est le plus simple et pratique méthode pour accéder une base de donné des projets géants adoptés NLIDB. [2]

3.1 Systèmes existants basé sur le NLIDB :

Ce sont des projets géants qui ont adoptés le NLIDB :

- LUNAR(1973): Lunar était l'un des premiers NLIDB, créé dans les années 60 comme une interface avec une base de données de roches lunaires

- LlFER / LADDER: C'était l'un des premiers bons systèmes de PNL de base de données. C'était conçu comme une interface en langage naturel vers une base de données d'informations sur les navires de l'US Navy. Ce système, tel que décrit dans un article de Hendrix (1978), a utilisé une grammaire sémantique pour analyser les questions et interroger une base de données distribuée. Les Le système LIFERILADDER ne pouvait prendre en charge que des requêtes mono tables simples ou des requêtes multi tables avec une jointure facile conditions

- CHAT-80: Le système CHAT-80 [5] est l'un des systèmes NLP les plus référencés systèmes dans les années quatre-vingt. Le système a été mis en place en Prologue. Le CHAT-80 était un système assez impressionnant, efficace et sophistiqué. La base de données de CHAT-80 se compose de faits (c'est-à-dire les océans, les grandes mers, les grands fleuves et les grandes villes) environ 150 des pays du monde et un petit ensemble d'anglais vocabulaire de la langue suffisant pour interroger base de données.

- GOOGLE NOW : est un assistant personnel intelligent qui prend la forme d'une application Android et iOS basée sur la reconnaissance vocale, le traitement du langage naturel par oral ainsi que sur la synthèse vocale pour apporter des réponses aux requêtes des utilisateurs à l'oral et à l'écrit, faire des recommandations et effectuer des actions en déléguant certaines requêtes à des services en ligne.[3]

4. Les Approches disponibles :

Pour réaliser ce projet, nous avons étudié de nombreux projets similaires, nous parlerons brièvement de ce qui nous a le plus aidé d'entre eux et des points les plus importants qui y sont mentionnés comme suit:

4.1 NLTSQLC:

Une application web qui propose un algorithme qui fait la traduction des requêtes écrite en langage naturel en langage SQL basé sur un rapport de 2016 de l'université Gautam Buddha en Inde.

L'idée de ce travail est de créer une interface utilisateur sur une page web utilisant PHP, HTML, CSS et JavaScript avec une zone pour saisir la requête, après que la requête écrite en anglais est validée, s'en suit un processus qui génère la requête en langage SQL.

Voici les étapes du travail :

- **Lower case Conversion :** convertir les lettres de la requête en minuscule.
- **Tokenisation :** la transformation de texte en des mot séparer et ayant un sens.
- **Escape Word Remover :** élimination des mots qui n'ont pas de sens important dans la requête
- **Noun-Pronoun-Verb Tagger:** Classer les mots selon leur nature
- **Relations-Attributes-Clauses Identifier :** classer les « tokens » en attribut, relation et clause (where,), integer,string , etc...
- **Ambiguity Remover :** si un mot est trouvé comme attribut deux fois, etc.. Cette fonction supprime tous les doublons d'un mot classé
- **Query Generation:**la requête écrit en langage SQL est générer à ce niveau-là. [4]

4.2 Fr2sql :

- le langage naturel source est le français et la méthode présentée est portable (opérationnelle instantanément sur n'importe quelle base de données SQL)
- la méthode présentée possède une grammaire suffisamment permissive pour que l'utilisateur ne ressente aucune restriction lexicale ou syntaxique
- un algorithme très efficace et un intervalle vaste pour les requêtes
- réaliser par Benoît Couderc Jérémy Ferrero en 2015
- l'outil TreeTagger (Schmid, 1994) est proposé comme une méthode pour tagger les mots en verbe, nom, adjective, Plus une lemmatisation en même temps, Helmut Schmid est le

développeur dans le TC projet à l'Institute de 'Computational Linguistics' of the University of Stuttgart. Le logiciel est open source, rapide et facile à utiliser.

- un dictionnaire de synonymes est chargé pour éviter le retour nul de la requêtes si un tel mot n'existe pas exactement le même comme le souligne l'étude de (Mohite & Bhojane, 2014), le dictionnaire utilisé est le thesaurus v.2.3 en date du 20 décembre 2011 de LibreOffice v.3.4. Cette ressource se trouve en accès libre sur internet

4.3 Pasero, 1997 et DISQUE (Pasero & Sabatier, 1998)

- Ce logiciel oblige l'utilisateur d'écrire les requête en langage naturel en respectant certain conditions : les noms de table sont explicitement mentionnée, les attributs devant être écrit de la même maniéré de la base de données

4.4 popescu et al., 2003 :

- La sélection et la condition est seulement traité dans ce travail
- N'utilise pas un dictionnaire de synonymes
- Oblige l'utilisateur à avoir une connaissance parfaite de la structure de la base,

4.5 Chandra, 2006

- rapporte des problèmes de linguistique et d'ambiguïté relatifs à la traduction du langage naturel

4.6 Chaudhari, 2013

- développe un traducteur relativement simple mais déjà plus ambivalent. Il se contente d'identifier le type de demande (select ou delete), de transformer les nombres écrits en lettres en chiffres, d'enlever les apostrophes et la ponctuation, d'extraire les mots clefs et de construire la requête en conséquence. Il utilise un dictionnaire de synonymes à compléter à la main pour élargir le vocabulaire accepté par son système. L'inconvénient de cette méthode est qu'à chaque fois que l'on souhaite porter cet outil sur une nouvelle base, il faut effectuer des entrées manuelles dans le dictionnaire (Benoît Couderc, Jerym.. ,2015)

5. Critique de l'existant

Quelle que soit la méthode utilisée ou la technologie implémentée pour créer un traducteur idéal, ça reste un enjeu difficile :

1. ces travaux ne traitent pas tous les cas possibles des clauses des requêtes SQL
2. la pluparts de ces travaux oblige l'utilisateur a écrit la requête en respectant des règles spécifique
3. une connaissance sur les informations de base de données est demandée avant d'écrire la requête.

6. Conclusion

Dans ce chapitre nous avons essayé d'étudier quelque application ou même des algorithmes de la traduction de requêtes écrites en langage naturel vers le langage SQL dans le but d'avoir une idée sur les fonctionnalités de ces dernières et de ressortir : leurs points forts, les étapes qui sont en communes et les points faibles que nous devrons éviter.

Chapitre 2 :

Les bases de données

1. introduction:

Une base de données est un ensemble d'informations qui est organisé de manière à être facilement accessible, géré et mis à jour. Elle est utilisée par les organisations comme méthode de stockage, de gestion et de récupération de l'information, pur bien géré notre projet il faut bien comprendre les bases de données et comment les utilisées.

2. Historique :

L'histoire des bases de données remonte aux années 1960, avec l'apparition des bases de données réseau et des bases de données hiérarchiques. Dans les années 1980, ce sont les bases de données object-oriented qui ont fait leur apparition. Aujourd'hui, les bases de données prédominantes sont les SQL, NoSQL et bases de données Cloud.

Les bases de données relationnelles ont été inventées en 1970 par E.F. Codd de IBM. Il s'agit de documents tabulaires dans laquelle les données sont définies afin d'être accessibles et de pouvoir être réorganisées de différentes manières.

Un standard SQL officiel a été adopté par l'ANSI (Américain National Standards Institute) en 1986. En 1987, l'ISO (International Organisation for Standardisation) l'a adopté à son tour. Ce standard a été mis à jour plus de six fois depuis lors. La version la plus récente est le SQL:2011

3. Définition d'une BD:

Elles se chargent elles-mêmes de créer, de mettre à jour ou de supprimer des données. Elles effectuent également des recherches parmi les données qu'elles contiennent sur demande de l'utilisateur, et de lancer des applications à partir des données.

Les bases de données sont utilisées par de nombreuses entreprises dans toutes les industries. Elles sont notamment utilisées par les compagnies aériennes pour gérer les réservations. Elles sont utilisées pour la gestion de production. Pour les enregistrements médicaux dans les hôpitaux, ou encore pour les enregistrements légaux dans les compagnies d'assurances. Les bases de données les plus larges sont généralement utilisées par les agences gouvernementales, les grandes entreprises ou les universités.

3.1 Comment fonctionnent les bases de données :

Les bases de données sont stockées sous forme de fichiers ou d'ensemble de fichiers sur un disque magnétique, une cassette, un disque optique ou tout autre type d'appareil de stockage. Les bases de données traditionnelles (hiérarchiques) sont organisées par champs, enregistrements et fichiers. Un champ est une seule pièce d'information. Un enregistrement est un ensemble de champs. Un fichier est une collection d'enregistrements.

Par exemple, un répertoire téléphonique est l'équivalent d'un fichier. Il contient un ensemble d'enregistrements, et chaque enregistrement regroupe trois champs : nom, adresse et numéro de téléphone.

3.2 Système de Gestion de Base de Données (SGBD) : les logiciels qui permettent d'accéder aux bases de données

3.3 Structured Query Language (SQL): est un langage de programmation standardisé utilisé pour gérer les bases de données relationnelles et effectuer différentes opérations sur les données qu'elle contient.

3.4 Requête SQL:

En informatique, une requête est une interrogation d'une base de données. ... Le plus connu est le SQL, il est exploité dans les bases de données relationnelles (dont les informations sont enregistrées dans des tableaux à deux dimensions).

3.5 Type de requête SQL: il existe plusieurs types différents de requêtes

3.5.1 Les requêtes de récupération de données (SELECT) :

SELECT <clause de sélection>

FROM <tables>

WHERE <clause where> ORDER BY <clause trier par > GROUP BY <grouper par clause> HAVING <clause ayant> LIMIT <clause limite>.

3.5.2 les requêtes de manipulation de données (INSERT, UPDATE, DELETE) :

INSERT INTO <insert clause> VALUES <values clause>

UPDATE <clause update> SET <clause set> WHERE <clause where>

DELETE FROM <clause de suppression> WHERE <clause d'endroit>

4. Conclusion :

Le SQL est le moyen pour accéder et manipuler toute base de données, pour passer de langage naturel vers le SQL et continuer le progrès sur ce projet il faut que nous ayons gains et abondance d'informations sur les bases de données et le SQL.

Chapitre 2 :

Les Fonctionnalités de SQL

1. Introduction :

Le SQL est un langage trop vaste dans ce chapitre on vas résumer quelque fonctionnalités comme suit.

2. Fonctionnalité de SQL : [5]

2.1 Création de la base de données :

La création d'une base de données en SQL est possible en ligne de commande. Même si les systèmes de gestion de base de données (SGBD) sont souvent utilisés pour créer une base, il convient de connaître la commande à utiliser, qui est très simple.

- **Syntaxe**

Pour créer une base de données qui sera appelé "dbname" il suffit d'utiliser la requête suivante qui est très simple:

```
CREATE DATABASE dbname
```

2.2 Création de la Table :

La commande CREATE TABLE permet de créer une table en SQL. Un tableau est une entité qui est contenu dans une base de données pour stocker des données ordonnées dans des colonnes. La création d'une table sert à définir les colonnes et le type de données qui seront contenus dans chacun des colonnes (entier, chaîne de caractères, date,…).

- **Syntaxe**

La syntaxe générale pour créer une table est la suivante :

```
CREATE TABLE tablename (Colonne1 type_données,Colonne2 type_données,Colonne 3 type_données,) ;
```

Dans cette requête, 3 colonnes ont été définies. Le mot-clé “type_donnees” sera à remplacer par un mot-clé pour définir le type de données (INT, DATE, VARCHAR,…). Pour chaque colonne, il est également possible de définir des options telles que :

- **NOT NULL :** empêche d’enregistrer une valeur nulle pour une colonne.
- **PRIMARY KEY :** indiquer si cette colonne est considérée comme clé primaire.

2.3 La Clé primaire (PRIMARY KEY)

La clé primaire est un index, chacune des tables ne peut contenir qu’une seule clé primaire, composée d’une ou plusieurs colonnes.

- **Syntaxe**

L’usage courant de PRIMARY KEY peut être effectué lors de la création d’une table à l’aide de la syntaxe suivante :

```
CREATE TABLE tablename (
id INT PRIMARY KEY NOT NULL,
```

OU

```
CREATE TABLE tablename (
id INT  NOT NULL,
```

2.4 SQL SELECT

L’utilisation la plus courante de SQL consiste à lire des données issues de la base de données. Cela s’effectue grâce à la commande SELECT, qui retourne des enregistrements dans un tableau de résultat. Cette commande peut sélectionner une ou plusieurs colonnes d’une table.

- **Syntaxe**

```
SELECT columnname FROM tablename
```

Cette requête SQL va **sélectionner** (SELECT) le champ “columnname” **provenant** (FROM) du tableau appelé “tablename”.

❖ **Ordre des commandes du SELECT**

Il existe plusieurs commandes qui permettent de mieux gérer les données que l'on souhaite lire.

Voici une requête SELECT qui possède presque toutes les commandes possibles:

SELECT *

FROM table

WHERE condition

GROUP BY expression

HAVINH condition

ORDER BY expression

LIMIT count

2.5 SQL DISTINCT

L'utilisation de la commande SELECT en SQL permet de lire toutes les données d'une ou plusieurs colonnes. Cette commande peut potentiellement afficher des lignes en doubles. Pour éviter des redondances dans les résultats il faut simplement ajouter DISTINCT après le mot SELECT.

- **Syntaxe**

SELECT DISTINCT columnname

FROM tablename

Cette requête sélectionne le champ “columnname” de la table “tablename” en évitant de retourner des doublons.

2.6 SQL WHERE

La commande WHERE dans une requête SQL permet d’extraire les lignes d’une base de données qui respectent une condition. Cela permet d’obtenir uniquement les informations désirées.

- **Syntaxe**

La commande WHERE s’utilise en complément à une requête utilisant SELECT. La façon la plus simple de l’utiliser est la suivante:

SELECT columnname FROM tablename WHERE condition

- **Opérateurs de comparaisons**

Il existe plusieurs opérateurs de comparaisons. La liste ci-jointe présente quelques-uns des opérateurs les plus couramment utilisés.

Opérateur	**Description**
=	Égale
<>	Pas égale
!=	Pas égale
>	Supérieur à
<	Inférieur à
>=	Supérieur ou égale à
<=	Inférieur ou égale à
IN	Liste de plusieurs valeurs possibles
BETWEEN	Valeur comprise dans un intervalle donnée (utile pour les nombres ou dates)
LIKE	Recherche en spécifiant le début, milieu ou fin d'un mot.
IS NULL	Valeur est nulle
IS NOT NULL	Valeur n'est pas nulle

Tableau 1 : listes des opérations dans le langage SQL et leur descriptions

2.7 SQL GROUP BY

La commande GROUP BY est utilisée en SQL pour grouper plusieurs résultats et utiliser une fonction de totaux sur un groupe de résultat.

- **Syntaxe**

SELECT column1, fonction (column2)

FROM tablename

GOUP BY column1

Cette commande doit toujours s'utiliser après la commande WHERE et avant la commande HAVING.

❖ **Utilisation d'autres fonctions de statistiques :**

- **AVG()** pour calculer la moyenne d'un set de valeur.
- **COUNT()** pour compter le nombre de lignes concernées.
- **MAX()** pour récupérer la plus haute valeur.
- **MIN()** pour récupérer la plus petite valeur.
- **SUM()** pour calculer la somme de plusieurs lignes

2.8 SQL HAVING

La condition HAVING en SQL est presque similaire à WHERE à la seule différence que HAVING permet de filtrer en utilisant des fonctions telles que SUM(), COUNT(), AVG(), MIN() ou MAX().

- **Syntaxe**

L'utilisation de HAVING s'utilise de la manière suivante :

SELECT column1,SUM(column2)

FROM tablename

GROUP BY column1

HAVING fonction(column2)condition

2.9 SQL ORDER BY

La commande ORDER BY permet de trier les lignes dans un résultat d'une requête SQL.

- **Syntaxe**

SELECT column1, column2

FROM tablename

ORDER BY column1

2.10 SQL INSERT INTO

L'insertion de données dans une table s'effectue à l'aide de la commande INSERT INTO. Cette commande permet au choix d'inclure une seule ligne à la base existante ou plusieurs lignes d'un coup.

Pour insérer des données dans une base: insérer une ligne en spécifiant toutes les colonnes

- **Syntaxe**

INSERT INTO tablename VALUES ('v1', 'v2'…)

2.11 SQL UPDATE

La commande UPDATE permet d'effectuer des modifications sur des lignes existantes.

- **Syntaxe**

La syntaxe basique d'une requête utilisant UPDATE est la suivante :

UPDATE tablename

SET column1='nouvelle valeur'

WHERE condition

Cette syntaxe permet d'attribuer une nouvelle valeur à la colonne pour les lignes qui respectent la condition stipulé avec WHERE.

2.12 SQL DELETE

La commande DELETE en SQL permet de supprimer des lignes dans une table.

- **Syntaxe**

La syntaxe pour supprimer des lignes est la suivante :

DELETE FROM tablename

WHERE condition

S'il n'y a pas de condition WHERE alors **toutes** les lignes seront supprimées et la table sera alors vide.

2.13 Jointure SQL :

Les jointures en SQL permettent d'associer plusieurs tables dans une même requête. Cela permet d'exploiter la puissance des bases de données relationnelles pour obtenir des résultats qui combinent les données de plusieurs tables de manière efficace.

2.14 Types de jointures :

2.14.1 **INNER JOIN :** jointure interne pour retourner les enregistrements quand la condition est vraie dans les 2 tables. C'est l'une des jointures les plus communes.

2.14.2 **CROSS JOIN :** jointure croisée permettant de faire le produit cartésien de 2 tables. En d'autres mots, permet de joindre chaque ligne d'une table avec chaque ligne d'une seconde table. le nombre de résultats est en général très élevé.

2.14.3 **LEFT JOIN (ou LEFT OUTER JOIN) :** jointure externe pour retourner tous les enregistrements de la table de gauche (LEFT = gauche) même si la condition n'est pas vérifié dans l'autre table.

2.14.4 **RIGHT JOIN (ou RIGHT OUTER JOIN) :** jointure externe pour retourner tous les enregistrements de la table de droite (RIGHT = droite) même si la condition n'est pas vérifié dans l'autre table.

2.14.5 **FULL JOIN (ou FULL OUTER JOIN) :** jointure externe pour retourner les résultats quand la condition est vrai dans au moins une des 2 tables.

2.14.6 **SELF JOIN :** permet d'effectuer une jointure d'une table avec elle-même comme si c'était une autre table.

2.14.7 **NATURAL JOIN :** jointure naturelle entre 2 tables s'il y a au moins une colonne qui porte le même nom entre les 2 tables SQL.

2.14.8 **UNION JOIN :** jointure d'union.

2.15 Exemples de jointures :

2.15.1 INNER JOIN :

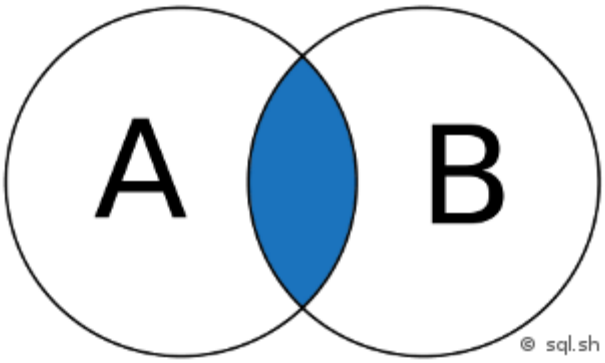


Figure 1: Intersection de 2 ensembles

```
SELECT *

FROM A

INNER JOIN B ON A.key = B.key
```

2.15.2 LEFT JOIN:

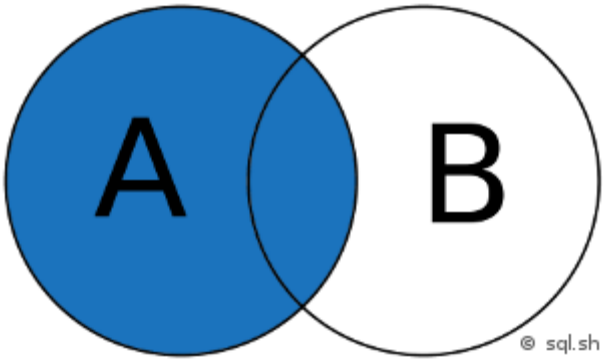


Figure 2: Jointure gauche (LEFT JOINT)

```
SELECT *

FROM A

LEFT JOIN B ON A.key = B.key
```

2.15.3 RIGHT JOIN:

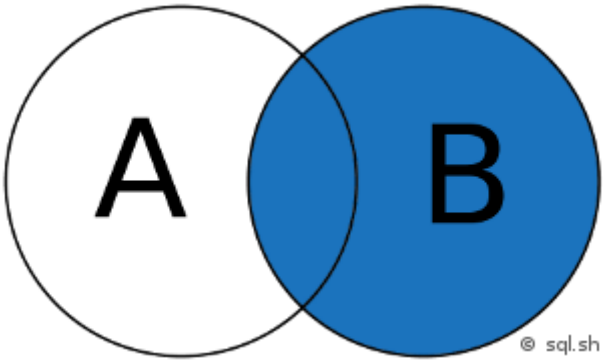


Figure 3: Jointure droite (RIGHT JOINT)

```
SELECT *

FROM A

RIGHT JOIN B ON A.key = B.key
```

2.15.4 FULL JOIN:

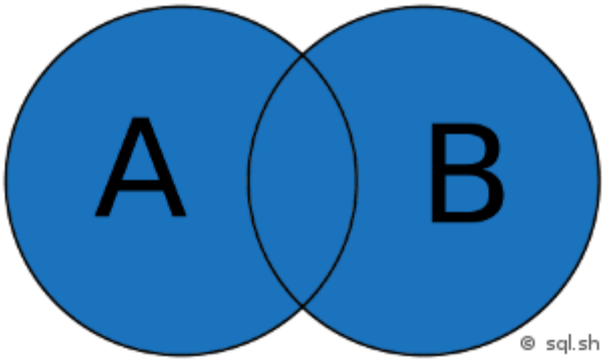


Figure 4: Union de 2 ensembles

```
SELECT *

FROM A

FULL JOIN B ON A.key = B.key
```

3. Conclusion:

Les fonctions SQL permettent d'effectuer des requêtes plus élaborées, par exemple adaptant les résultats pour qu'une chaîne soit affichée en majuscule ou bien pour enregistrer une chaîne avec la date actuelle.

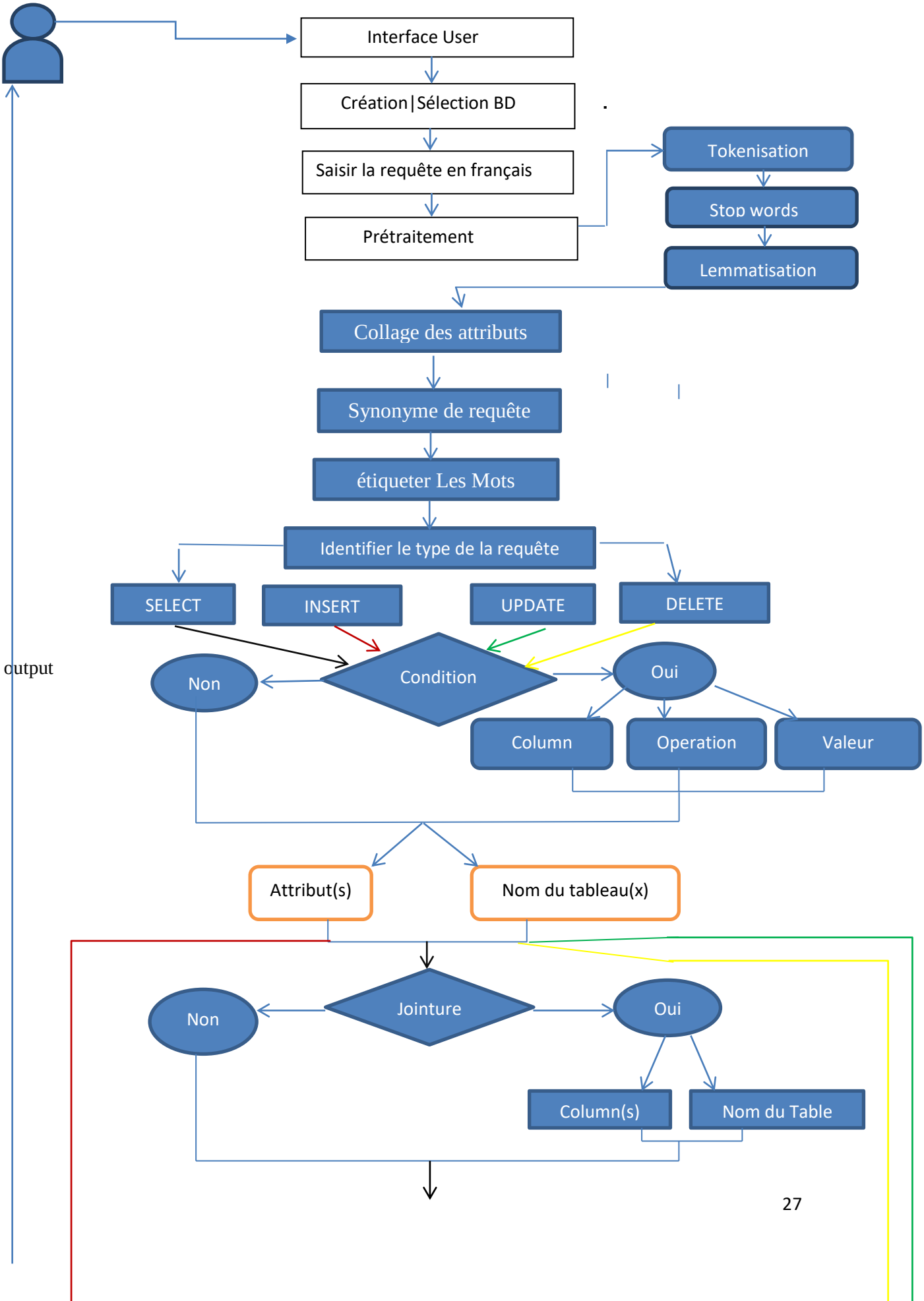
Chapitre 3 :

Architecture du système proposé

1. Introduction :

Dans ce chapitre, nous allons d’abords donnez un organigramme général qui donne les principales étapes et leur agencement afin de passer d'une requête en LN vers une requête SQL.

2. Schéma de la Conception :



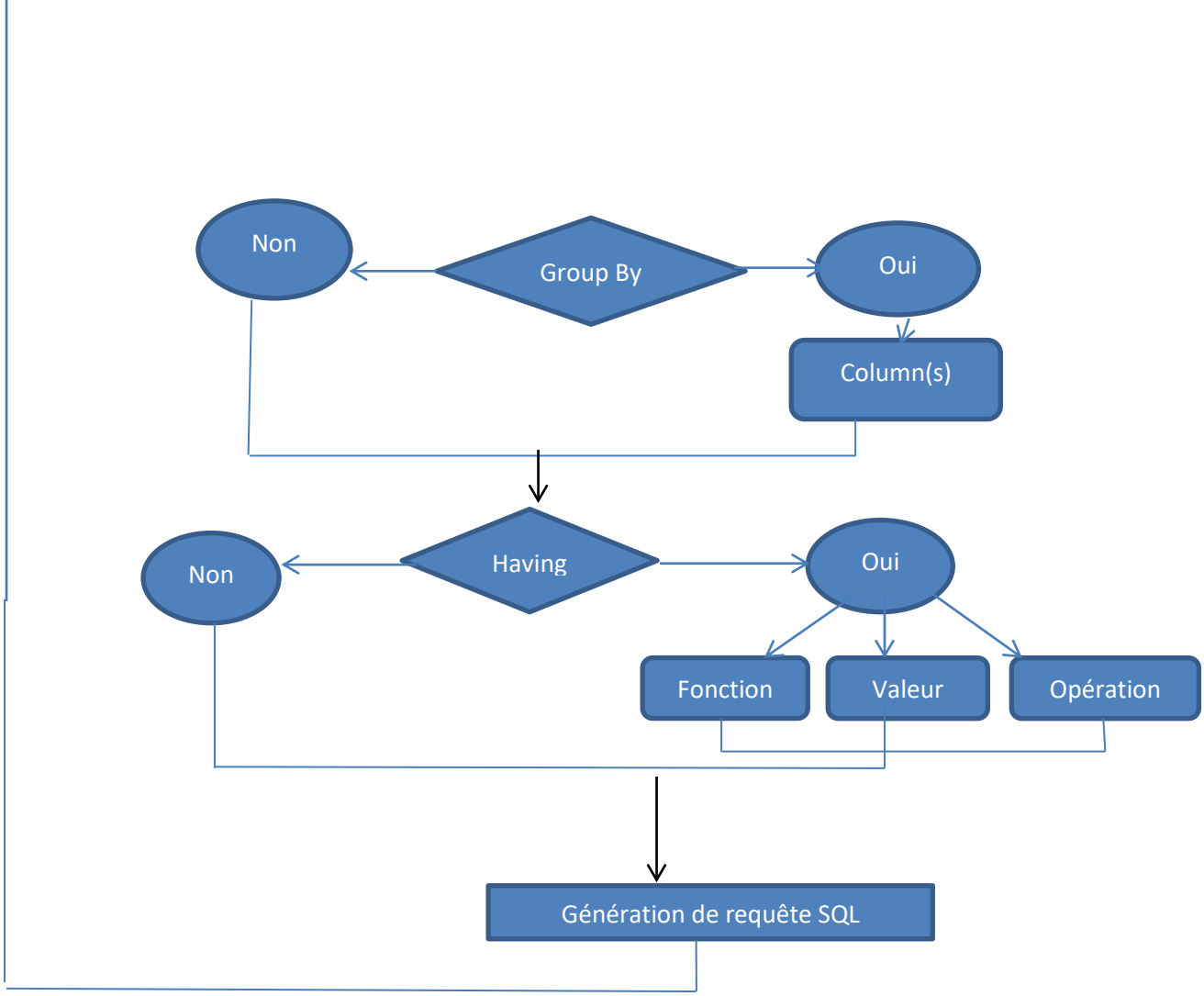


FIG 5 : Architecture du système

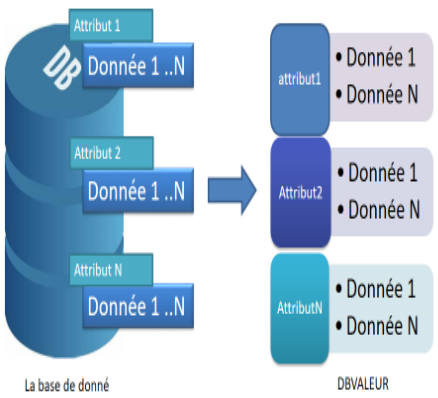
3. Récupérer les informations de la base de données :

D'abord nous devons récupérer les informations de la base de données comme les noms de tables, les noms des colonnes et les données (valeurs), pour les manipuler quand on en a besoin par la suite en les affichant dans la requête finale générée par le programme à sa juste place

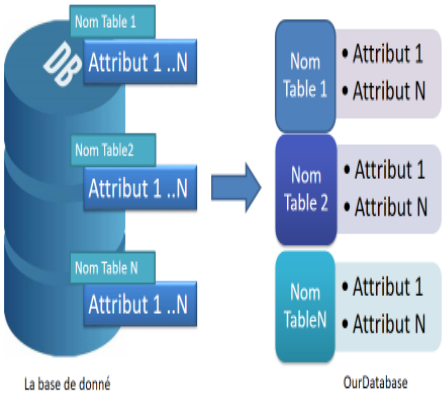
Le stockage des informations se fera en deux phases :

1. Les noms des tables et les attributs :
   Les « attributs » sont stockés en dépendance des « nom de table ».
   - Pour chaque « nom de table » stocké nous avons ses « attributs » qui sont stocké en relation avec lui.
   - Pour chaque attribut stocké nous pouvons s'avoir quelle table lui est associée
2. Les attributs et les valeurs (données) :
   Les « données » sont stockées en dépendance des « attributs ».
   - Pour chaque « attribut » stocké nous avons ses « données » qui sont stocké en relation avec lui.
   - Pour chaque « donnée » stocké nous pouvons savoir le nom d'attribut qui

appartient.



La transformation des informations du base de données 1



La transformation des informations du base de données 2

FIG 6 : Transformation des informations de la BD **FIG 7 :** Transformation des informations de la BD

4.La Phase 1: acquisition et prétraitement de requête

La première phase de travail à faire après qu'on a reçu la requête en langue naturel (français) est:

1. **Tokenization :** la requête est devisée en différents séquences

FIG 8 : la liste des Tokens

exemple1 :

Entrée :" lister les noms et les prénoms ayants un âge supérieur à 20 ans".

On a comme résultat la liste des Tokens suivants :

Résultat : [lister ,les ,nom ,et ,le , prenom,ayants,un,age,superieur,à,20,ans]

Après cette étape on devra éliminer les mots vides

2. **Eliminer les Mots vides du français :**

Ce sont les mots qui n'ont pas de valeur dans la compréhension d'un texte. Exemple : " Le, la, les, du, des, etc. "

Remarque: on a retiré les mots "par" et "est" parce que on en a besoin dans la partie condition et GROUP BY par la suite.

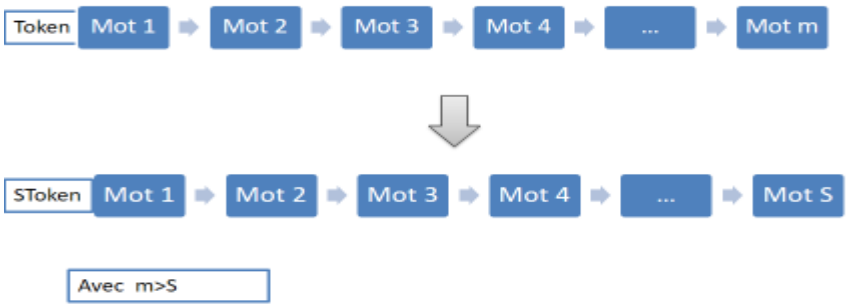


FIG 9 : la liste des STokens

Exemple :
Entrée : la liste des Tokens.
Résultat : [lister, nom, prénom, âge, supérieur, 20, ans].

3. **Lemmatisation :**
On l'utilise pour l'élimination de la notion de pluriel

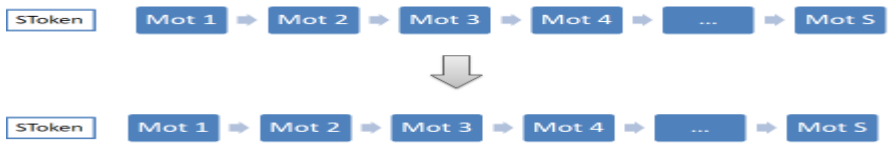


FIG 10 : la lemmatisation

5.La phase 2 : identification des attributs.

1) étiqueter les mots : [6]

Pour la reconnaissance des données numériques plus précisément nous avons choisi une méthode qui nous permettra de détecter la nature des mots comme suit :

Faire une étiquette pour chaque mot dans la liste.

-'NOUN' est réservé pour la valeur de type nom.

-'VERB' pour les verbes.

-'CD' est réservé pour les valeurs numériques, et ainsi de suite pour tout mot dans la liste

Nous testons la valeur de l'étiquette du mot en cours dans la liste et l'étiquette 'CD' qui signifie qu'une valeur numérique est détectée. Si une correspondance est trouvée, on le considère comme une valeur numérique.

Ce test est utilisé surtout au niveau de la recherche des valeurs de conditions si nous avons une valeur numérique qui n'appartient pas aux données de la base de donnés par exemple :

Cette méthode nous aidera surtout au niveau de la détection des attributs et les valeurs de la clause de condition « WHERE » voir la page n°40.

2) Collage des attributs :

Nous créons trois listes : liste 1, liste 2 et liste 3. Dans chaque liste nous mettons le résultat de la combinaison de chaque mot de la liste des STokens avec son suivant.

Liste 1 : une combinaison des mots « sans espace »

Liste 2 : une combinaison « avec un espace »

Liste 3 : une combinaison « avec un tiré de huit ‘Under score’»

Mot1Mot2	Mot2Mot3	…	Mot(n-1)MotN

List 1

Mot1 Mot2	Mot2 Mot3	…	Mot(n-1) MotN

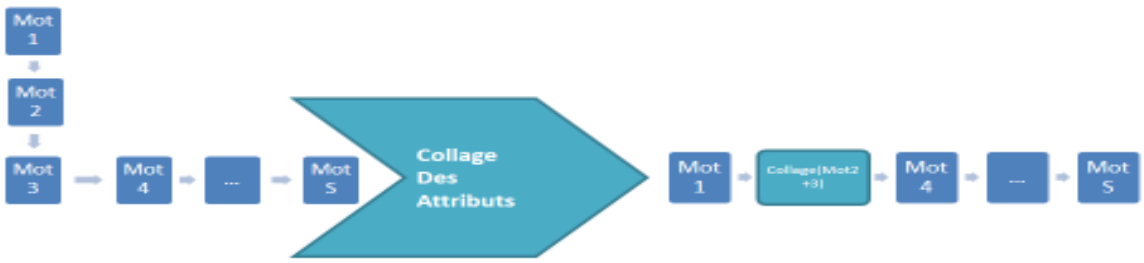
List 2

Mot1_Mot2	Mot2_Mot3	…	Mot(n-1)_MotN

List 3

Table 2 : Collage des attributs

Nous testons si un de ces mots est un « nom de table », « attribut » ou une « valeur » dans la base de donné, si oui nous remplacent ce mot avec les paire de mots qui le représente.



Collage des attributs

FIG 11 : Collage des attributs

Exemple :

« afficher les nom de fournisseurs ayant un salaire supérieur à 20000 DZD »

Le dernier état de la requête est une liste des STokens

afficher	nom	fournisseur	salaire	supérieur	20000	DZD

Liste des STokens

Table 3 : liste STokens de l'exemple 1

affichernom	nomfournisseur	fournisseursalaire	salairesupérieur	supérieur2000	20000DZD

List 1

Afficher nom	Nom fournisseur	Fournisseur salaire	Salaire supérieur	Supérieur 2000	20000 DZD

List 2

Affiche_rnom	Nom_fournisseur	Fournisseur_salaire	Salaire_supérieur	Supérieur_2000	20000_DZD

List 3

Table 4 : les listes de collage des attributs de l'exemple 1

Résultat après le test de collage :

Le dernier état de la requête est une liste des STokens

afficher	Nom_fournisseur	salaire	supérieur	20000	DZD

Liste des STokens

Table 5 : liste finale de STokens de l'exemple 1

Par la suite on test l'existence de chaque "nouveau mot" (chaine de caractère) dans le base de donné si on trouve que une colonne ou une donnée ou bien un nom de tableau est écrit de cette manière on le remplace par cette chaine de caractère.

3) Synonyme des mots :

L'idée est de traduire le mot dont nous voulons avoir le synonyme en anglais puis de générer tous ses synonymes possibles et traduire la liste des synonymes mots par mot en français, pour avoir la liste des synonymes souhaitée voici un schéma qui illustre le :

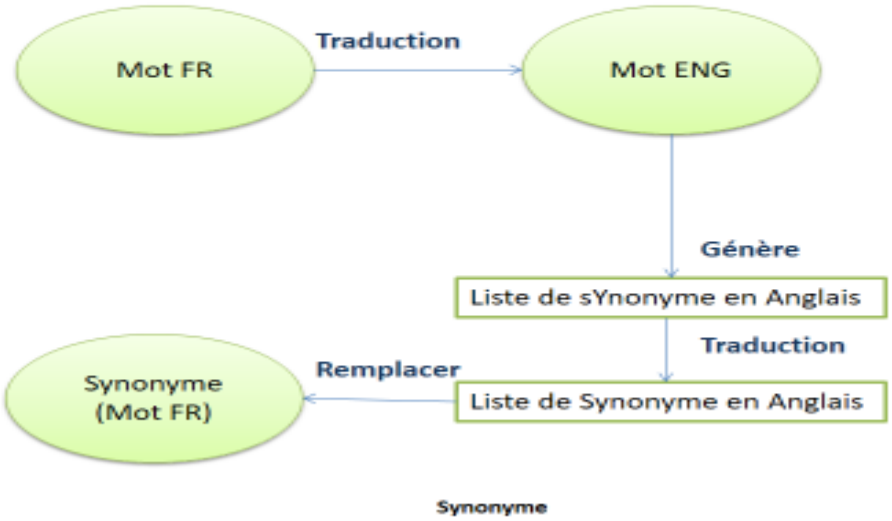


FIG 12 : les synonymes

Exemple : soit une base de données film :

« Afficher les longueurs des films réalisé par ‘ Alfred Hitchcock’ »

Le dernier état de la requête est une liste des STokens

afficher	longueurs	film	réaliser	Alfred Hitchcock

Table 6 : Liste des STokens

afficher	longueurs	film	réalisé	Alfred Hitchcock
synonyme (afficher)	Synonyme =durée			

Table 7 : détection de synonymes

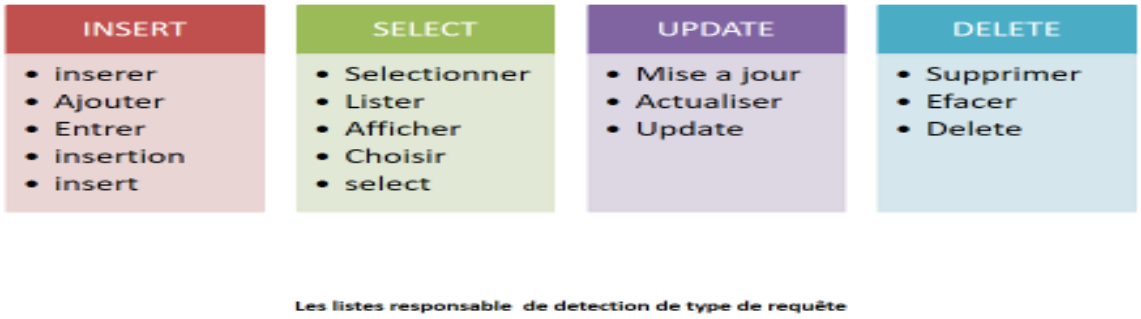
6. La Phase 3 : Déterminer le type de la requête

L’étape suivante est de tester sur quel type de requête SQL on va travailler.

La décision de savoir si l'instruction en langue naturel représente une requête d'extraction de donnée (SELECT) ou une requête de mise à jour (INSERT, UPDATE, DELETE) est prise à ce stade à l'aide du dictionnaire de donnée
Pour désigner le type de la requête, par exemple lorsque des mots comme « insérer » ou un de ses synonymes apparaissent en entrée, le type de la requête est 'INSERT' et ainsi de suite.

Pour cela on a créé quatre listes : une pour la sélection, l'insertion, la mise à jour et la suppression, chaque liste est chargée par tous les synonymes du mot.

Les listes sont données comme suit :



Les listes responsable de detection de type de requête

FIG 13 : les types d'une requête SQL

L'algorithme va tester l'existence de tous les "Token" de la requête si une correspondance est trouvée, on fait le traitement nécessaire correspondant à ce type de requête pour continuer le travail.

Dans le cas où aucun mot n'est trouvé dans les listes, Alors le type de la requête est 'SELECT' par défaut.

1. Formation de requête 'SELECT' :

On cas ou une requête de sélection est trouvée, nous continuons le travail par la détection des éléments pricipaux de cette requête, nous les définissons comme suit :

SELECT COUNT **(**attributes|***)** **or** AVG **(**attributes|***)** **or...**MIN **(**attributes|***)**| * | attributes

FROM nom_table

[JOIN type_join attribut table_a_join]

[WHERE attribut_condition opération valeur]

[GROUP BY attribut_groupe_by]

[HAVING fonction_condition operation valeur]

[ORDER BY attribut_order_by type_order_by **]**

1. La condition : "WHERE CLAUSE " :

Exemple :

« Trouver tous les fournisseurs ayant un salaire supérieur à 20000 DZD »

Le dernier état de la requête est une liste des STokens

trouver	tous	fournisseur	salaire	supérieur	20000	DZD

Table 8 : Liste des STokens 2

Le traitement de cette partie passe par trois étapes essentielles :

- Tester l'existence d'une condition dans la requête écrite en langage naturel
- Récupérer les éléments essentiels :
    1. Attribut(s)_condition.
    2. Opération_condition.
    3. la(les) valeur(s)_condition.
- relie les éléments en prenant en compte tous les cas possibles.

A. l'existence d'une "condition" :

Nous chargeons un tableau par les mots

égale	entre	précise	valeur	supérieur	inferieur	moins	plus	seulement	supérieur ou égale	…

Table 9 : Tableau global 1. Les mots signifiant la condition.

On va parcourir la liste STokens mot par mot, si une correspondance est trouvée entre les mots dans cette liste et le tableau 1 on dit qu'une condition est détectée.

B. Récupérer les éléments essentiels :

C. Solution proposé :

Pour récupérer les éléments de condition on lance une recherche dans la liste des STokens on arrête la recherche dès que nous trouvons un mot de dictionnaire global. Exemple : Le mot « supérieur » dans la requête du paragraphe précédent.

Nous démarrons une deuxième recherche définie par :

Un point de départ : le mot trouvé dans le dictionnaire (la position courante de la première recherche) par exemple le mot « supérieur» dans l'exemple 1

Un intervalle : deux mots en avant et deux en arrière, qui veut dire nous travaillons sur une nouvelle liste définie ci-dessous :

fournisseurs	salaire	supérieur	20000
k-2	k-1	k	k+1

Table 10 : Liste des STokens avec intervalle

L'attribut de condition :

- les mots [k+1], [k-1], [k-2], [k+2] sont testés pour voir si 'ils appartiennent aux noms de colonnes DBVALEUR.
- Le test est fait selon cet ordre et le premier mot validé est prêt comme attribut de condition.
- Le mot [k-1] : le premier mot testé est « salaire » et comme le test est valide (parce que « salaire » est le nom de colonne de tableau DBVALEUR) l'attribut_condition = salaire est ajouté à la liste des attributs de « where clause ».
- Nous reprenons les étapes précédentes jusqu'à la fin de la liste .

Les valeurs de condition :

- Le mot [k+1] est prêt sans test d'appartenance sachant que généralement la valeur de condition est mentionnée après un des mots du « tableau global »

L'opération de condition :

Le tableau global est divisé en six listes :

supérieurs	plus	dépasse

Liste 1 : les mots exprimant la supériorité

inferieurs	Moins

Liste 2 : les mots exprimant l'infériorité

supérieur ou égale

Liste 3 : les mots exprimant la notion « supérieure ou égale »

inferieur ou égale

Liste 4 : les mots exprimant la notion « inferieur ou égale »

égale	précise	valeur	seulement

Liste 5 : les mots exprimant l'égalité

entre

Liste 6 : les mots exprimant la notion de « comparaison »

Table 11 : listes de 1 à 6 des mots signifiant les opérations

- Liste1 : déclare une opération de supériorité" >" en cas d'appartenance
- Liste2 : déclare une opération infériorité "<" d'appartenance
- Liste3 : déclare une opération de supériorité ou égale ">=" d'appartenance
- Liste4 : déclare une opération de infériorité ou égale "<=" d'appartenance
- Liste5 : déclare une opération de égalité" =" d'appartenance
- Liste6 : déclare une opération de comparaison "BETWEEN" d'appartenance
- Si aucun mot de ses listes est détecter l'opération d'égalité '=' est déclarer comme opération de condition par défaut.

Problématique :

Voici les exemples suivants :

Exemple 1 : « afficher tous les fournisseurs habitant à Tipaza »

afficher	tous	fournisseur	habitant	Tipaza

Table 12 : Liste des STokens de l'exemple 1

Exemple 2 : « le salaire et les noms des fournisseurs qui ont plus que 30 ans »

salaire	nom	fournisseur	plus	30	ans

Table 13 : Liste des STokens de l'exemple 2

L'attribut de condition	opération de condition	La valeur de condition
NONE	« = »	NONE

Table 14 : Résultats de traitement de condition. Exemple 1

L'attribut de condition	opération de condition	La valeur de condition
nom	« > »	30

Table 15 : Résultats de traitement de condition. Exemple 2

D. Solution optimisée :

La détection d'une condition dans une requête écrite en langage naturel se fait par la

détection d'une valeur (une valeur de DBVALEUR) : on parcourt la liste des STokens si un mot est détecté comme une valeur des DBVALEUR on dit qu'une condition est trouvé dans cette requête.

afficher	tous	fournisseurs	habitant	Tipaza

Table 16 : Liste des STokens de l'exemple 1

- L'attribut de condition est désormais détecter automatiquement à partir de la table DBVALEUR, dès que nous trouvons la valeur « Tipaza » nous cherchons quel attribut est-en relation avec cette valeur.
- Dans l'exemple 1 :

L'attribut de condition	opération de condition	La valeur de condition
Adresse	« = »	Tipaza

Table 17 : Résultats de traitement de condition

salaire	nom	fournisseur	plus	30	ans

Table 18 : Liste des STokens de l'exemple 2

- Dans le deuxième exemple :

L'attribut de condition	opération de condition	La valeur de condition
Age	« > »	30

Table 19 : Résultats de traitement de condition

E. la concaténation des éléments :

L'étape finale est d'analyser et traiter tous les cas possibles et valides qu'on peut rencontrer dans une condition « where clause », en ajoutant les mots clé nécessaires :

2. le nombre des attributs =le nombre des valeurs =le nombre d'opérations=1 : » une seul condition est le cas le plus simple qu'on peut trouver dans une « **where** clause » la concaténation sur ce niveau est faite par une combinaison simple des éléments de condition sans rien changer ou ajouter.
   Attribut 1 opération 1 valeur 1
3. le nombre des attributs < le nombre des valeurs : le cas où nous avons pour chaque attribut plusieurs valeurs
   2.1 Le nombre des attributs =1, le nombre des valeurs = 2 et le nombre d'opération =1 :c'est un cas correcte. Nous avons deux valeurs avec le même attribut et la même opération. Cela veut dire que nous sommes sur deux possibilités : soit la valeur d'opération est le mot 'entre' qui va exprimer un intervalle sur les résultats demandés, soit elle appartient à une des opérations arithmétiques {= , > , < , <= , >=} dans ce cas, l'interrogation de la BD est

exprimée avec l'opération OR .
Attribut 1 opération 1 valeur 1 OR attribut 1 opération 1 valeur 2.

2.2 Le nombre des attributs =1, le nombre des valeurs = 2 et le nombre d'opérations =2 : un cas non valide, nous ne pouvons pas avoir deux valeur associé à un même attribut par deux opération différent
Attribut 1 opération 1 valeur 2 OR Attribut 1 opération 2 valeur 2
Attribut 1 opération 1 valeur 2 OR Attribut 1 opération 2 valeur 2
Exemple 1: Age <12 OR Age >10
Exemple 2: Age <12 AND Age >10
Exemple 3 : nom = Selma OR nom<Fodil
Tous les cas ci-dessus sont mal former, illogique et incorrect.

4. le nombre des attributs =le nombre des valeurs = le nombre des opérations =2 :
   Un cas validé, nous avons deux attributs de condition, deux opérations et deux valeurs pour les associer, la combinaison entre ces éléments est faite :
   Attribut 1 opération 1 valeur 1 AND attribut 2 opération 2 valeur2.

5. le nombre des attributs =1 le nombre des valeurs >2.
   Le cas où un attribut a plusieurs possibilités des valeurs :
   Attribut 1 IN (valeur 1, valeur 2, .., valeur N).

6. le nombre des attributs >=2 avec le nombre des attributs <le nombre des valeurs :
   nous reprenant les étapes de 1 jusqu'à 4 en les traitent cas par cas en séparent avec le mot clé AND.
   Attribut 1 opération 1 valeur 1 AND Attribut 2 opération 2 valeur2 OR Attribut 2 opération 2 valeur 3.

le nombre des attributs	le nombre des valeurs	le nombre d'opérations	Valeur d'opération	Négation détecté	Mot clé ajouté	Résultats
1	1	1	V : [= ou > ou < ou <= ou >=] / |V|=1	faux	rien	Adresse=Tipaza
1	1	1	V : [= ou > ou < ou <= ou >=] / |V|=1	vrai	rien	Adresse=Tipaza
1	2	1	V : [= ou > ou < ou <= ou >=] :|V|=1	faux	OR	code=p2001 OR code=p3
1	2	1	V : [entre] /V=1	faux	BETWEEN ….AND….	Date BETWEEN '2014' '2015
1	>2	>2	V : [= ou > ou < ou <= ou >=] / |V|>1	faux	IN	Adresse IN ('Tipaza','Alger','Blida'

1	>2	>2	V : [= ou > ou < ou <= ou >=] / |V|>1	vrai	NOT IN	Adresse NOT IN (‘Tipaza’,’Alger’,’Blida’
2	2	2	V : [= ou > ou < ou <= ou >=] / |V|=2	faux	AND	Adresse NOT IN (‘Tipaza’,’Alger’,’Blida’ salaire >25000

Table 20 : Les combinaisons possible pour génère le « where clause »

Remarque : les mots qu’ils sont classé comme des attributs de condition sont retiré de la liste STokens pour éviter de les voir au niveau des attributs de « select clause »

trouver	tous	fournisseur	salaire	supérieur	20000	DZD

Liste des STokens

trouver	tous	fournisseur	supérieur	20000	DZD

Liste des STokens

Table 21 : la liste des STokens après la suppression des attributs de condition

1.2 Les attributs « select clause » :

Pour trouver les attributs (noms de colonnes) en lance une recherche dans la liste des STokens Token par Token, en testant l’existence de chaque Token dans DBVALEUR comme un nom de colonne si une correspondance est trouvée nous ajoutons ce mot à la liste des attributs de « **select** clause ».

Si la liste des attributs détecter est encore vide on dit que la requête demande une sélection total (l'étoile (*)).

Nous avons les deux exemples suivant :

Exemple1 :

« Afficher les noms et prénoms des fournisseurs habitant soit à Tipaza»

Le dernier état de la requête est une liste des STokens

afficher	nom	prénom	fournisseur	habitant	Tipaza

Table 22 : Liste des STokens

Résultats :

nom prénom

Table 23 : La liste des attributs.

Exemple 2 :

« Trouver tous les fournisseurs ayant un salaire supérieur à 20000 DZD »

Le dernier état de la requête est une liste des STokens

trouver	tous	fournisseur	supérieur	20000	DZD

Table 24 : Liste des STokens

Résultats :

La liste des attributs est vide alors l'étoile est générer comme une valeur d'attributs sur « select clause ».

1.3 Le nom du tableau :

- **Les requêtes mono table :**
  Pour trouver les attributs (noms de colonnes) en lance une recherche dans la liste des STokens Token par Token, en testant l'existence de chaque Token dans OurDATABASE comme un nom de colonne si une correspondance est trouvée nous prenant ce mo comme un nom de tableau de « **from** clause ».
  Si aucun nom de table n'est explicitement mentionné nous parcourons la liste des attributs de « **select** clause » et nous faisons retirer le nom de colonne d'OurDATABASE qu'elle est en relation

« Afficher les noms et prénoms des fournisseurs habitant soit à Tipaza»

Le dernier état de la requête est une liste des STokens

afficher	nom	prénom	fournisseur	habitant	Tipaza

Table 25 : Liste des STokens

Résultats :

fournisseur

La liste des noms de tableaux

donner	tous	salaire

Liste des STokens

Résultats :

fournisseur

Table 26 : La liste des noms de tableaux

- Les requêtes multi table :
  Si la liste des noms de la table contient plus qu'un élément alors nous disons qu'une jointure est vérifiée nous prenant que le premier élément comme un nom de table de « **from** clause ».

Remarque : dans le cas où les attributs appartiennent au plusieurs tableaux, les requêtes multi-tables n'ont pas traité dans ce cas ils vont être traités au niveau de la clause jointure. Voir la page

1.4 Les fonctions d'agrégats :

Les fonctions d'agrégats sont les fonctions responsables du calcule dans le SQL, ils sont situés dans la clause SELECT après l'emplacement des attributs, si ils existent dans la requête.

1. tester l'existence d'une fonction d'agrégat dans la requête.

Les fonctions d'agrégats sont appliqués sur les colonnes du tableau sinon, l'étoile (*) est le cas le plus dominant dans les requêtes SQL.

COUNT (attribut|*): le comptage est utiliser pour le calcul de nombre de linge par colonne donné

AVG (attribut|*) : calculer la moyenne d'une colonne donnée ou de tout le tableau, le cas de (*)

SUM (attribut|*) : calculer la somme d'une colonne donnée ou de tout le tableau, le cas de (*)

MAX (attribut|*) : extraire la valeur maximal d'une colonne ou de tout le tableau, le cas de (*)

MIN (attribut|*) : extraire la valeur minimal d'une colonne ou de tout le tableau, le cas de (*)

Pour connaitre est ce qu'il est demandé de faire une fonction d'agrégat nous fessons un test d'appartenance entre la liste des STokens et les liste montrer ci-dessous.

Liste_COUNT= ["nombre", "combien"]

Liste_AVG= ["moyennes", "moyenne"]

Liste_MAX= ["maximum", "plus haut", haut"]

Liste_MIN= ["minimum"," bas", "plus bas"]

Liste_SUM= ["somme", "addition"]

La détection de la forme final d'une fonction d'agrégat consiste à :

1. choisir le type de fonction demandé.

- Extraire l'attribut de la fonction ou génère l'étoile si aucun attribut n'est trouvé.
- Tester si une élimination de redondance est demandée.

Exemple 1 :

« Trouver la moyenne de salaire de fournisseurs »

Le dernier état de la requête est une liste des STokens

trouver	moyenne	salaire	fournisseur

Table 27 : Liste des STokens

Lorsqu'un mot de STokens est trouvé dans la liste « liste_AVG » alors une fonction d'agrégat de calcul de la moyenne est détectée il suffit de trouver l'attribut associé à elle.

Pour faire cela en lance une sous recherche d'intervalle 2k de la position courant (le mot « moyenne »), nous cherchant un mot dans cette intervalle et qui appartient au nome de colonne de DBVALEUR.

trouver	moyenne	salaire	fournisseur
k-1	k	k+1	k+2

Table 28 : Liste des STokens

Le mot « salaire » est dans la liste de attribut de « **select** clause » alors en le retire et remplace avec :

Le nom de fonction (attribut)

AVG (salaire).

Exemple 2 :

« Afficher le nombre total des produits ».

Le dernier état de la requête est une liste des STokens

afficher	nombre	total	produits
k-1	k	k+1	k+2

Table 29 : Liste des STokens

Le mot « nombre » appartient au liste « liste_count » alors la fonction est une fonction de comptage.

Aucun attribut n'est trouvé dans l'intervalle de recherche alors l'étoile est passé comme paramétré.

L'expression COUNT (*) est ajouter au niveau des attributs de « la clause **select** ».

1.5 Les jointures :

Est la manière de récupérer des informations (data) à partir de deux ou plusieurs tableaux, ont au moins un attribut (colonne) commun.

Nous disons qu'une jointure est détectée si la liste « des noms des tables » a plus qu'un élément²&²²

1.5.1 Définir les types de jointure :

Les plus célèbre sont le (INNER) JOIN, LEFT (OUTER) JOIN, RIGHT (OUTER) JOIN, FULL (OUTER) JOIN, la commande de la jointure est préciser par les mots clés citer en dessus, les mots clé entre parenthèse sont optionnelle de les écrire.

La jointure LEFT, RIGHT, and FULL:

L'idée est de voir si le mot « tout » est mentionné dans la requête faire une sous recherche dans un intervalle de 2k, et voir si un nom de tableau et mentionnée dans cette intervalle.

Si le nom de tableau est celui exprimé dans « la clause **from** » alors un **left join** est détecter**,** si le nom de table mentionner est le deuxième « **nom de tableau** » dans la liste des nomes des tableaux alors un **right join** est détecter, si le deux table sont dans cette intervalle nous disons qu'un **full join** est détecter.

La plus simple et dominante entre les quatre jointures, est le **inner join** elle est utilisée quand il est demandé d'afficher les attributs de différentes tables, on cas ou aucun condition des autre type est vérifier.

Exemple :

« Afficher tous les produits ayant un fournisseur de Alger ».

Le dernier état de la requête est une liste des STokens donnée ci-dessous

afficher	tous	produits	fournisseur	Alger

afficher	tous	produits	fournisseur	Alger
k-1	k	k+1	k+2	

afficher	tous	produits	fournisseur	Alger
k-1	k	k+1	k+2	

Table 30 : détection de la jointure

- Deux tables sont détectées.
- Détection de mot « tous »
- La table « produit » est mentionné
- « produit » est le nom de table de la clause « **from** »
- Un left join est détectée.

1.6 Le GROUPE BY :

Nous chargeons la liste « liste_groupe_by » par les mots indiquant qu'un groupe by est détecté dans une requête :

total	chaque	par	agréger	agréger	rassembler	linge

Table 31 : Liste_groupe_by

Si un mot dans la liste des STokens est détecter aussi dans la liste liste_groupe_by alors on positionne la recherche sur ce mot et en lance une recherche, partir de ce mot jusqu'à deux pas en avant si un mot appartient aux un des valeurs de dictionnaire OurDATABASE (un attribut) alors on le garde comme un attribut de groupe by

1.7 Le HAVING () :

Si une condition est détectée dans la requête en langage naturel mais la valeur de l'attribut est toujours nul, donc il n'existe pas un attribut constant pour la condition, nous sommes donc sur le HAVING () clause, parce que le having clause est généré pour faire une condition après le lancement de la requête SQL, c'est à dire nous voulons appliquer la condition sur un résultat obtenu, l'attribut de condition de « **having** » est une fonction d'agrégat réaliser en retournant sur la partie de fonction d'agrégat expliqué ci-dessus .

1.8 L'ORDER BY :

Le traitement de détection de la clause d'ODRER BY est similaire au GROUPE BY seulement la liste des mots est changer, le parcours de la sous recherche est fait avec 2k, pour la récupération d'attribut d'ordre by plus précisément.

ordre	ascendant	descendant	triés	décroissant	croissant	classée	classement

Table 32 : Liste global. Les mots signifiant l'ordre by

1.8.1 **Un ordre ascendant ou descendant** ?

Pour s'avoir quel est le type d'ordre utilisé nous divisant la liste global en deux listes :

ascendant	alphabétique	croissant

Table 33 : Liste d'ordre by ascendant

descendent	décroissant	

Table 34 : Liste d'ordre by descendant

Le mot clé ASC est ajouté à l'attribut d'ordre by si un de ces mots est trouvé dans la liste des STokens.

Le mot clé DESC est ajouté à l'attribut d'ordre by si un de ces mots est trouvé dans la liste des STokens

1.9 Génération de requête (sortie) :

La concaténation entre tous les résultats obtenu nous donne la requête final de la sélectionne.

SELECT liste_attributs FROM nom_table [TYPE JOIN nom_table2]

[WHERE where_clause] [GROUPE BY attribut_groupe_by] [HAVING having_clause] ORDER BY attribut_ordre type_ordre]

- ❖ Les clauses entre [ ] sont optionnel, si son test d'existence n'est pas validé nous les retirent de la requête finale

2. La suppression :

Les éléments nécessaire pour la clause « **Delete** »

1. Le nom de table : nous le récupère de la liste des « **nom** de **table**» expliqué ci-dessus.
2. La clause « **where** » : les éléments de condition.
   - ❖ Si aucune condition n'est demandée tous la table vas être supprimé.

3. La Mis à jour :

Les éléments nécessaire pour la clause « **Update** »

1. Le nom de table : nous le récupère de la liste des « **nom** de **table**» expliqué ci-dessus.
2. Les attributs : nous le récupère de la liste des « **Liste attributs**» expliqué ci-dessus.
3. Les nouvelles valeurs : comment peut ont détecte qu'un mot dans la liste des STokens est une nouvelle valeur associée à un attribut ? pour cela nous ferons trios test :
   3.1 Des valeurs existantes : des équivalentes des nouvelles valeurs peuvent être déjà dans la base de données pour d'autres instances.
   3.2 Des valeurs numériques : les nouvelles valeurs peuvent être des valeurs numériques nous pouvant les détecter facilement par son étiquette 'CD' par la méthode illustrée dans la phase des « **étiquette** les **mots** ».
   Exemple :

   « Fait une mise à jour pour le salaire de fournisseur 'Ahmed Imad' égale à 25000 DZD »

   Le dernier état de la requête est une liste des STokens :

Mise à jour	salaire	fournisseur	Ahmed	Imad'	25000	DZD

Table 35 : Liste des STokens

Mise à jour	salaire	fournisseur	Ahmed	Imad'	25000 | CD	DZD

Table 36 : Liste des STokens Après le 'Tage' des mots

   3.3 Autres : nous peuvent détecter les nouvelle valeurs par la méthode de NER
   - ❖ Cette méthode est utiliser pour les listes des « Token » elle prend chaque mot et le fait une étiquette selon son sens « Person, city, country, .. » et 'none' pour les mots n'avons pas un sens :
     Exemple :

« Fait une mise à jour de fournisseur Fodil Selma sur son adresse pour quel soit ‘ Tipaza’ »
Le dernier état de la requête est une liste des STokens :

Mise à jour	fournisseur	Fodil	Selma	adresse	égale	Tipaza

Table 37 : Liste des STokens

Mise à jour	fournisseur	Fodil	Selma	adresse	égale	Tipaza | CITY

Table 38 : liste des STokens avec une NER classification

4. La clause « **where** » : les éléments de condition
   - ❖ Si aucune condition n’est demandée tous les linges serrant modifié.

4. L’insertion :

Les éléments nécessaire pour la clause « **Insert** »

1. Le nom de table : nous le récupère de la liste des « **nom** de **table**» expliqué ci-dessus.
2. Les attributs : nous le récupère de la liste des « **Liste attributs**» expliqué ci-dessus.
   - ❖ Si la liste des attributs n’a aucuns éléments l’insertion serra faite sur le tableau
3. Les valeurs : les mêmes méthodes utilisées pour **la mise à jour** ci-dessus

5. Conclusion :

Le processus de génération d’une requête SQL à partir d’une requête écrite en langage naturel traverse une phrase finale ‘génération’ l’étape qui combine tous les résultats que nous avons obtenues pour obtenir la requête souhaité . la combinaison des résultats se fait en respectant les règles de combinaison et d’écriture et l’exécution d’une requête SQL .De nombreux facteurs, dont la position des mots ,l’explicitement attributs ,l’ambiguïté entre eux et même leur existence dans la base de données ou pas sans oublier les signes de ponctuations, les lettres en majuscules ont un grand rôle sur la requête généré(sélection, insertion ,suppression, et la Mis à jour) .

Chapitre 4: Implémentation et résultat

1. Définitions

1.1 MySQL :

La meilleure base de données relationnelle Open Source. MySQL est un système de gestion de base de données relationnelle (RDBMS) basé sur le SQL (Structured Query Language). Ce SGBDR est compatible avec toutes les plateformes : **Linux, UNIX, et Windows**. Il peut être utilisé avec de nombreuses applications, mais on l'associe le plus souvent aux applications web.

1.2 Python 3 .7 : Python 2.1 est une version dérivée de Python 1.6.1 et de Python 2.0. Sa licence est renommée Python Software Foundation License. Tout code, documentation et spécification ajouté, depuis la sortie de Python 2.1 alpha, est détenu par la Python Software Foundation (PSF), une association sans but lucratif fondée en 2001, modelée d'après l'*Apache Software Foundation*.

1.3 Jupyter : Créé à partir de Python en 2014, **Jupyter est un notebook de calcul** (computational notebook) open source, gratuit et interactif. C'est une application web basée client permettant de créer et de partager du code, des équations, des visualisations ou du texte.

2. Les bibliothèques utilisées :

Ce sont des bibliothèques près défini sur python 3.7.

```
Entrée [1]: from tkinter import *
            import nltk
            from nltk.tokenize import word_tokenize
            import networkx as nx
            import tkinter as tk
            from nltk.corpus import stopwords
            from nltk.stem.porter import PorterStemmer
            from nltk.stem import WordNetLemmatizer
            from nltk.corpus import wordnet
```

FIG 14 les bibliothéques utilisées

❖ Création de l'interface et input query

On prend en entrée une requête en langue naturel on a choisis la langue française

Mais tout d'abord on a créé une interface pour recevoir la requête en utilisant la bibliothèque ''Tkinter''.

❖ Analyse syntaxique

La requête en entrée doit être nettoyé passons par plusieurs étapes :

3.1. Tokenization : se fait à l'aide de tokenizer, WordTokenize du package 'NLTK'.

3.2 Remouve Stopwords : éliminer les mots vides français qui sont près défini sur Python.

3.3 Lemmatization : se fais à l'aide du LEMMATIEZ du NLTK.

- Tous les processus après ces étapes utilisent ces séquences pour le traitement.

❖ Identification des attributs:

4.1 Etiqueter les mots :

-les mots sont étiquetés selon le Tagger utilisant le tagger StanfordTagger.
-Les éléments de la requête tokeniser sont soit des mots soit des verbes.
-Les mots sont utilisés pour trouver les attributs et le nom de la table.

4.2 Synonyme de requête :

La fonction la plus puissante et célèbre de nltk qui génère les synonymes d'un mot précis, support que la langue anglaise, qui nous obligent de chercher d'autre méthodes

Alors on a choisir de travailler avec "TRANSLATE"

Pour quoi "TRANSLATE"?

Translate est une bibliothèque de NLP extrêmement puissante implémenter sur Python, construite sur NLTK fais la possibilité de la traduction d'un mot d'une langue à une autre.

Translate nous aidera à trouver une solution au problème de “synonyme”,

❖ Analyse de requête :

L’ensemble de tables est préparer, qui contiendra les tables qui sont nécessaire dans la requête à former

- **Une requête SELECT**

  La syntaxe d’une commande SELECT est toujours construite de la même façon. Après le mot clé SELECT on énumère:

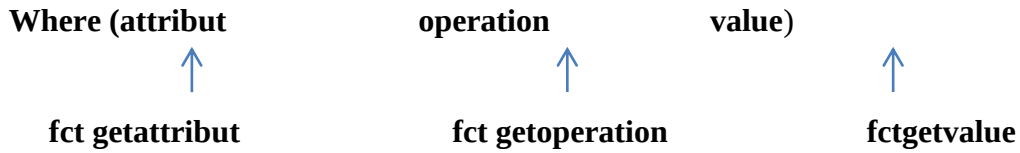
  Select (….) From (…..) Where (…….)

  Select (*): elle est générer à la suite de l’algorithme
  Select (attribut) :c’est la fonction ‘**getfocus’** qui est responsable de récupérer l’attribut qui se situe après le Select
  Select (plusieurs attributs) : la fonction **‘getNewfocus’** dans le cas où il existe plusieurs attributs après le select
  Select (distinct) : ’existDISTINCT’
  Select (Agrégat) : sont les fonctions **Count, Avg, Max, Min, Sum** par exemple (if focus =moyenne donc la fonction **avg et ainsi de suite)**

  From :
  La fonction **‘gettablename’** qui va récupérer le nom de la table

  Where : maintenant on passe à la condition et on va tester s’il existe la condition where ou pas **(fct existcondition)**

- Dans le cas où la condition where existe donc elle va être de la forme



❖ Généralisation :

-Lors de la conception de cette partie nous avons plusieurs test serons réalisables Cela va dépendre de la conception détaillée des fonctions utilisées Par contre le fait de décrire les tests fait réfléchir au meilleur moyen d'implanter les fonctions concernées.

Apres qu'on a pu former une simple requête SQL, à chaque fois quand on a fait un teste nous trouvons un problème qu'il va nous guidé a une solution pour une application améliorée et plus développé

Parmi les problèmes trouvés lors de la conception :

1. après qu'on a entré la requête en langue naturel et être découper en séquence on a trouvé un problème à cause de lui notre algorithme nous a retourner une erreur car si on veut afficher un titre complet composé comme 'Honkytonk Man' il fallait prendre ce titre complet comme un seul attribut par contre la phase de tokenization retourne 'Honkytonk' et 'Man' c'est ce que nous a obligé de faire un collage des attributs pour prendre les mots composées comme un seul attribut.

Exemple :

Afficher Les films avec le titre « Honkytonk Man ».

Résultat avant collage des attributs :

Erreur

Résultat attendus :

SELECT * FROM film WHERE titre = Honkytonk Man

Résultat après collage des attributs :

SELECT * FROM film WHERE titre = Honkytonk Man

2. Dans le cas où on veut afficher tous les éléments d'un tableau c'est obligatoire d 'ajouter 'Tous' et ces synonymes dans un dictionnaire pour avoir un SELECT * (étoile) a la place de l'attribut après le SELECT.
   Exemple :

La liste des étudiants de la section 'A'.

Résultat :

SELECT attribut FROM étudiant WHERE section =A

Résultat attendu :

SELECT * FROM étudiant WHERE section =A

3. On veut traduire une requête en langue naturel en entrée contient un attribut et dans la base de donnée on n'a pas le même attribut on a son synonyme c'est pour ça on a ajouté la Phase 'Synonyme de requête' pour que ce problème soit évité.

Exemple :

Base de donnée: {

OurDATABASE= {

"film":("numf","titre", "genre", "annee", "duree", "directeur","pays"),

"personne": ("nump", "prenom", "nom", "datenaissance"),

"ACTEUR": ("numa","role")

}

Retrouver la liste des films dont la longueur dépasse 180 minutes.

Résultat:

SELECT * FROM film WHERE attribut > 180

Résultat attendu:

SELECT *FROM film WHERE duree > 180

Résultat après ‘synonyme de requête’ :

SELECT *FROM film WHERE duree > 180

4. Toujours avec les attributs, au début on a traité le cas où on trouve un seul attribut après le SELECT après on a amélioré notre algorithme jusqu’on a pu extraire et afficher tous les attributs demandé dans la requête à traduire.

Exemple :

La liste des films dirigés par « Hitchcock », en n’affichant que le titre, l’année et en les triant par année.

Résultat:

SELECT titre FROM film film GROUPE BY année ORDER BY année

Résultat attendu :

SELECT titre , année , année FROM film GROUPE BY année ORDER BY année

- **Table des statistiques :**

Nombre totale de requête	Nombre de requête correcte	Nombre d’erreur	Nombre de requête correcte après résolution de problème trouvée
100	42	58	80

Table 39 :Table de statistique

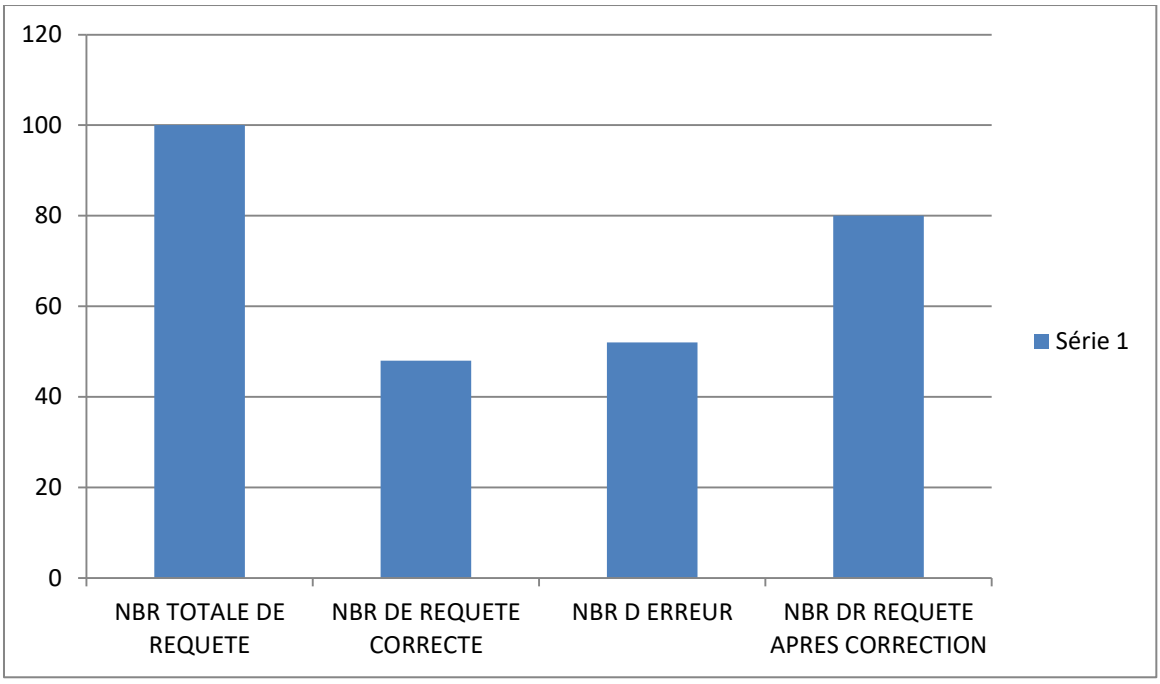


FIG 14 -Diagramme de statistique de requête-

Conclusion

Tout au long du projet qui nous a été confié, nous avons eu la possibilité de constater les problèmes que pose la traduction des requêtes du langage naturel au langage SQL.

Ces problèmes sont les suivants :

- Identification du type de la requête: select, update...
- Identification des tables concernées
- identification des attributs concernés par les requêtes
- identification des éventuelles clauses

S'agit-il de requêtes simples, de jointures naturelles, de fonctions d'agrégation...

Pour cela, nous avons essayé de trouver les formes générales qui permettent de dire qu'il s'agit de telle ou telle requête. Mais cela reste un problème épineux.

Nous avons fait de notre mieux pour essayer de cerner toutes les catégories de requêtes mais le problème demeure dans l'immensité du langage naturel.

Une des idées aurait été de fournir un mini langage naturel très restreint permettant de générer des requêtes standardisées. Mais l'inconvénient d’une pareille approche demeure dans le fait qu'on diminue de la liberté d'expression de l'utilisateur.

Le travail n'est qu'une tentative et certainement qu'elle pourrait déboucher sur des travaux qui viendront l'enrichir .De notre côté, nous pouvons affirmer que nous avons fait ce qu'on a pu.

Bibliographies

[1] A Brief History of Natural Language Processing

[2] NLIDB (Natural Language Interface to DataBases)

[3] ACCESSING DATABASE USING NLP - IJRET

[4] Natural Language text to SQL query (Amey Baviskar ,Akshay Borse,Eric White,Umang Shah)Mastére pfe 2017

[5] https://sql.sh/cours/create-database

[6] LinguisticsWeb.org Stanford PoS Tagger: tagging from Python

Webographie

https://www.tutorialspoint.com/sql/sql-insert-query.htm

https://nlp.stanford.edu/software/tagger.shtml

http://new.galalaly.me/index.php/2011/05/tagging-text-with-stanford-pos-tagger-in-java-applications/

https://cis.uni-muenchen.de/~schmid/tools/TreeTagger/#:~:text=The%20TreeTagger%20is%20a%20tool%20for%20annotating%20text,TreeTagger%20has%20been%20successfully%20used%20to%20tag%20German%2C

https://cis.uni-muenchen.de/~schmid/tools/TreeTagger/data/tree-tagger2.pdf