Rapport Projet Big Data & DataMining

Bachir Kassoum Ahmadou

Tarek Seba

Master 2 Génie de l'Informatique Logiciel

Soualmia

20 Fevrier 2022

Résumé

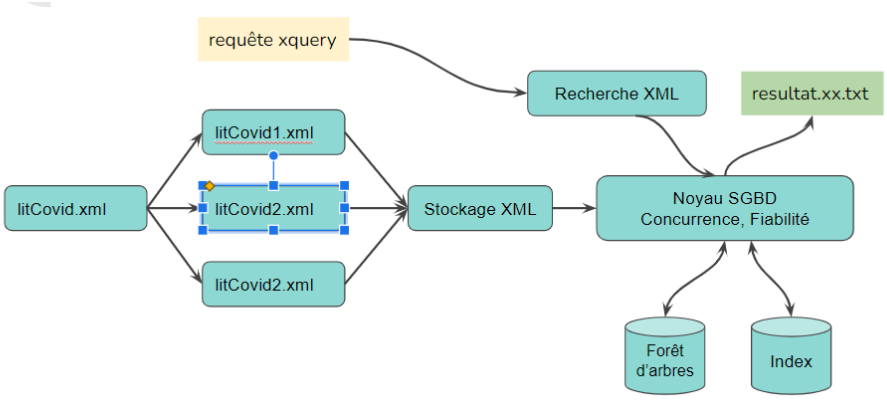
Le projet a pour objectif d'extraire des connaissances à partir de données non structurées et semi-structurées, en particulier des textes biomédicaux. Les données proviennent de plusieurs corpus de textes et sont issues du projet LitCovid et de l'outil PubTator. Le projet se divise en deux parties : la première partie consiste à constituer des collections et à comparer différents SGBD en termes de performances, tandis que la seconde partie consiste à extraire des connaissances à partir des données générées dans la première partie.

I. Big data : import/export, requêtage de données

A. Gestion des données XML → eXistDB

1. Intégration et gestion dans une base XML native

Nous avons choisi d'utiliser eXistBD pour la base de données XML native. Étant donné que le fichier à stocker était trop volumineux, nous l'avons découpé en plusieurs sous-fichiers que nous avons stockés dans une collection eXistDB. Cette collection nous permet d'effectuer des requêtes XQuery sur les fichiers stockés.



a) Le titre et le résumé de chaque PMID

Ci dessous la requête XQuery qui permet de récupérer pour chaque PMID son titre et son résumé :

```
xquery version "3.1";

for $x in doc("/db/bigdata/litcovidBioCXML1.xml")/collection/document[exists(passage/infon[@key="article-id_pmid"])]
let $pmid := $x/passage/infon[@key="article-id_pmid"]/text()
let $title := $x/passage[infon[@key="article-id_pmid"]]/text/text()
let $resume :=
    for $p in $x/passage
    return
        if($p/infon[@key="section_type"]/text()="ABSTRACT") then fn:concat($p/text/text()," ")
        else()
        return concat($pmid," / ", $title, " --- ", string-join($resume), "&#10;" )
```

Resultat → Cf fichier pmid-title-resume-exist.txt

b) La liste des références de chaque PMID

Ci dessous la requête XQuery qui permet de récupérer pour chaque PMID la liste des références :

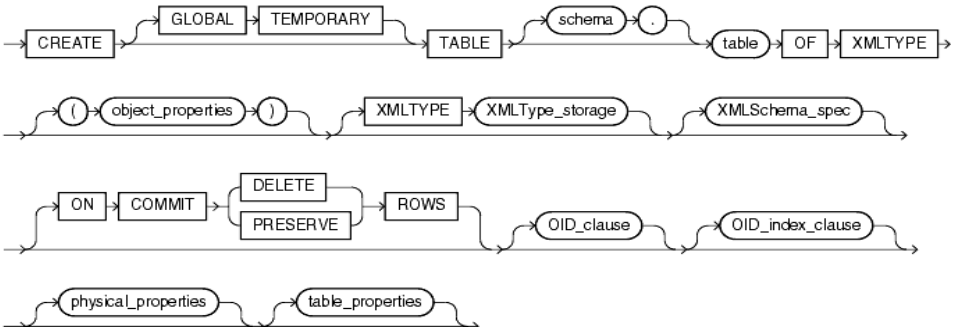
```
xquery version "3.1";

for $x in doc("/db/bigdata/litcovidBioCXML1.xml")/collection/document[child::passage/infon[@key="article-id_pmid"]]
let $pmid := $x/passage/infon[@key="article-id_pmid"]/text()
where $x/passage[child::infon[@key="section_type" and text()="REF"] and child::infon[@key="pub-id_pmid"]]
return
    <result>
        {$pmid}
        {
            for $p in $x/passage[child::infon[@key="section_type" and text()="REF"] and child::infon[@key="pub-id_pmid"]]
            let $ref := $p/infon[@key="pub-id_pmid"]/text()
            return concat(" / ",$ref)
        }
    </result>
```

Resultat → Cf fichier pmid-refs-exist.txt

2. Intégration et gestion dans une base de donnée relationnel objet

   Nous avons opté pour Oracle comme système de gestion de base de données relationnelle. Pour stocker du XML, Oracle propose deux solutions : le stockage natif en utilisant un attribut de type XMLType, et le stockage relationnel. Nous avons décidé de choisir le stockage natif car il permet une gestion plus efficace du XML, ainsi que des requêtes plus performantes sur des données XML complexes, comparé au stockage relationnel qui impose des conversions de données pour les manipuler.



Ci dessous le processus de stockage des documents XML sous oracle :

```
create table litcovid (filename varchar(64) primary key, xml XMLType);

create or replace directory XML_DIR as 'C:\Users\XXX\XXX\Folder\xml';

insert into litcovid VALUES ('litcovidBioCXML1.xml',
xmltype(bfilename('XML_DIR','litcovidBioCXML1.xml'),nls_charset_id('AL32UTF8')));
```

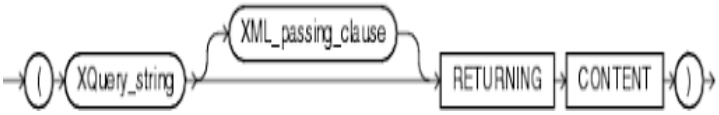
➔ Création de la table “litCovid”;

➔ Ajout d'un répertoire XML_DIR ou les fichiers XML seront stockés;

➔ Insertion de l'enregistrement dans la table "litCovid" avec le nom du fichier "litCovid1.xml" et le contenu xml du fichier est stocké dans le répertoire "XML_DIR".

a) Interrogation des données

Pour l'interrogation des données XML sous oracle, nous avons utilisé la commande "XMLQuery" qui permet d'exécuter une requête XQuery sur une colonne de type XML stockée dans une table. Elle renvoie le résultat de la requête sous forme de type XML. La syntaxe de la commande est la suivante :



(1) Le titre et le résumé de chaque PMID

```
select XMLQUERY (
    'for $x in /collection/document[exists(passage/infon[@key="article-id_pmid"])]
    let $pmid := $x/passage/infon[@key="article-id_pmid"]/text()
    let $title := $x/passage[infon[@key="article-id_pmid"]]/text/text()
    let $resume :=
    for $p in $x/passage
    return
        if($p/infon[@key="section_type"]/text()="ABSTRACT") then fn:concat($p/text/text(),"
        else()
        return concat($pmid," / ", $title,"&#10;" )' passing xml returning content)
from litcovid spool;
```

Resultat : cf fichier → pmid-title-resume-oracle.txt

(2) La liste des références de chaque PMID

```
select XMLQUERY (
    'for $x in /collection/document[exists(passage/infon[@key="article-id_pmid"])]
    let $pmid := $x/passage/infon[@key="article-id_pmid"]/text()
    let $title := $x/passage[infon[@key="article-id_pmid"]]/text/text()
    let $resume :=
    for $p in $x/passage
    return
        if($p/infon[@key="section_type"]/text()="ABSTRACT") then fn:concat($p/text/text()," ")
        else()
        return concat($pmid," / ", $title,"&#10;" )' passing xml returning content)
from litcovid spool;
```

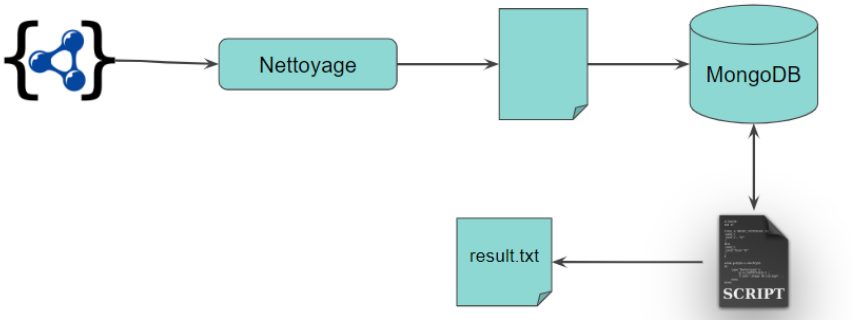
Resultat : cf fichier → pmid-refs-oracle.txt

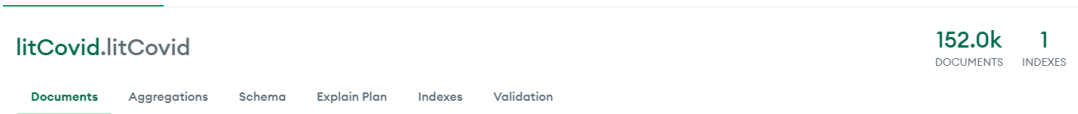
<u>NB :</u> Pour voir toute les requête → cf fichier Projet-DM-BD.sql

B. Gestion des donnés JSON

1. Intégration et gestion dans une base de donnée JSON natif

   Le choix s'est porté sur MongoDB pour stocker du JSON natif, cependant le fichier contenant les données présentait des erreurs et n'était pas valide selon le validateur. Nous avons donc procédé à un nettoyage avant de stocker les données dans notre collection. Une fois stockées, nous avons utilisé un script JavaScript pour se connecter à la base de données et récupérer les données. Le processus comprend les étapes de nettoyage, stockage et interrogation :





   a) Le titre et le résumé de chaque PMID

```
db = connect("mongodb://localhost/big-data");

db.BKA.find({ "passages.infons.article-id_pmid": { $exists: true } }).forEach(
  function (x) {
    var result = "";
    if (x.pmid) {
      result = result + x.pmid + "/";
    }
    for (var i in x.passages) {
      if (x.passages[i].infons.section_type === "TITLE") {
        result = result + " " + x.passages[i].text + "\n";
      }
      if (
        x.passages[i].infons.section_type === "ABSTRACT" &&
        x.passages[i].infons.type === "abstract"
      ) {
        result = result + " " + x.passages[i].text + "\n";
      }
    }
    result = result + "\n";
    print(result);
  }
);

```

Requete : cf fichier → pmid-title-resume.js

Resultat : cf fichier → pmid-title-resume-mongo.txt

b) La liste des références de chaque PMID

```
db = connect("mongodb://localhost/big-data");

db.BKA.find({ "passages.infons.article-id_pmid": { $exists: true } }).forEach(
  function (x) {
    var result = "";
    if (x.pmid) {
      result = result + x.pmid + "/";
    }
    for (var i in x.passages) {
      if (
        x.passages[i].infons.section_type === "REF" &&
        x.passages[i].infons.type === "ref"
      ) {
        if (x.passages[i].infons["pub-id_pmid"]) {
          result = result + " " + x.passages[i].infons["pub-id_pmid"] + "/";
        }
      }
    }
    result = result + "\n";
    print(result);
  }
);

```

Requete : cf fichier → pmid-refs.js

Resultat : cf fichier → pmid-refs-mongo.txt

```
PS C:\Users\ahmad\Downloads\mongosh\bin> .\mongosh.exe --file .\pmid-refs.js > pmid-refs.txt
```

NB : Pour l'exécution il faut d'abord installer le shell de mongodb

C. Comparaison méthode d'intégration

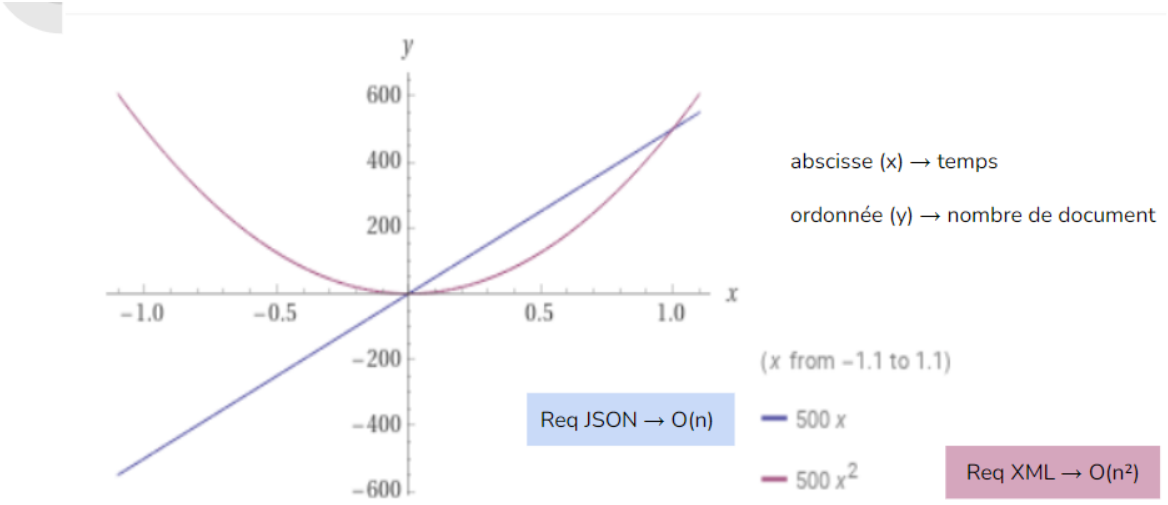
Pour la comparaison des différentes méthodes d'intégration, il conviendra de dresser un tableau comparatif en se basant sur les critères suivants :

- ➔ Temps de chargement : mesure du temps nécessaire pour charger les données dans la base de données.
- ➔ Temps d'extraction : mesure du temps nécessaire pour extraire les données de la base de données.
- ➔ Temps d'export : mesure du temps nécessaire pour exporter les données de la base de données.
- ➔ Complexité de la requête : mesure de la complexité de la requête nécessaire pour extraire les données.

Ci dessous le tableau comparatif :

SGBD	Type de donnée	Performance de chargement	Performance de requête	Facilité d'utilisation
ExistDB	XML	Rapide	Rapid	Facile
Oracle (XMLType)	XML	Moyen	Rapide	Moyen
MongoDB	JSON	Rapide	Rapide	Facile
Oracle (JSONType)	JSON	Moyen	Rapide	Moyen

Dans l'ensemble, tous les SGBD utilisés ont montré de bonnes performances de chargement et de requête pour les données non structurées au format XML ou JSON. Cependant, pour les données XML, ExistDB ont montré des performances plus rapides que Oracle (XMLType). En ce qui concerne la facilité d'utilisation, les SGBD MongoDB, ExistDB ont été considérés comme plus faciles à utiliser que Oracle (XMLType ou JSONType). En fin de compte, le choix du SGBD dépendra des besoins spécifiques de l'application et des préférences de l'utilisateur.



Nous recommandons le choix de de MongoDB car les requêtes sont plus optimisés (***la fonction find() de MongoDB est optimisée pour effectuer des recherches efficacement dans des grandes collections de données.***) → complexité temporel en O(n) alors que les requêtes XML ont une complexité en O(n2)

II. Fouille de donnée : Apriori

Nous avons développé un **script python** qui réalise une analyse des règles d'association sur des données textuelles (fichier pmid-refs.txt) et extrait des ensemble fréquent en utilisant l'algorithme Apriori.

AssociationRule.py

```python
from apyori import apriori
import pandas as pd
import os.path

def database_transaction_matrix(references):
    # Crée une matrice binaire pour les données de référence
    # Les clés sont les identifiants des références et les valeurs sont les éléments associés à chaque référence
    keys = list(references.keys())
    values = list(set([val for sublist in references.values() for val in sublist]))
    df = pd.DataFrame(0, index=keys, columns=values)
    for k, vals in references.items():
        for v in vals:
            df.at[k, v] = 1
    return df

def apriori_frequent_itemsets(input_file, output_file, support_threshold):
    # Analyse des règles d'association à partir des données textuelles
    if not os.path.isfile(input_file):···

    references = {}
    with open(input_file, 'r') as input_txt:
        # Itère sur chaque ligne du document pour extraire les références
        for line in input_txt:···

    data = database_transaction_matrix(references)
    data.to_csv('input_data.csv', index=False)

    # Analyse des règles d'association à partir de la matrice binaire
    frequent_itemsets = pd.DataFrame(list(apriori(data.astype('bool'), min_support=support_threshold, use_colnames=True)))

    if len(frequent_itemsets) == 0:···

    frequent_itemsets.to_csv(output_file, index=False)

# Exemple d'utilisation
apriori_frequent_itemsets("./pmid-refs-mongo.txt", "./result1.txt", 0.5)
```

La fonction **database__transaction_matrix** prend en entrée un dictionnaire de références, où chaque clé est un identifiant unique pour une référence et chaque valeur est une liste d'éléments associés. La fonction crée ensuite une matrice binaire où les lignes représentent les références et les colonnes représentent les éléments, avec un **1** indiquant que l'élément est associé à la référence et un **0** indiquant qu'il ne l'est pas.

La fonction apriori_frequent_itemsets prend trois arguments : **input_file**, qui est le nom d'un fichier texte contenant les données de référence ; **output_file**, qui est le nom du fichier où écrire les ensembles fréquents ; et **support_threshold**, qui est le seuil de support minimum pour l'algorithme **d'Apriori**. La fonction lit les données du fichier d'entrée et utilise **database__transaction_matrix** pour créer une matrice binaire représentant les données. Elle applique ensuite l'algorithme d'Apriori pour extraire les ensembles fréquents de la matrice et écrit les résultats dans le fichier de sortie.

La fonction **apriori** est importée depuis la bibliothèque apyori, qui fournit une implémentation de l'algorithme d'Apriori. La fonction prend une matrice binaire en entrée, ainsi qu'un seuil de support minimum, et renvoie une liste d'ensembles fréquents.

Les ensembles fréquents résultants sont stockés dans un objet **Pandas** DataFrame et écrits dans un fichier à l'aide de la méthode to_csv.

NB : le script vérifie si le fichier d'entrée existe avant de tenter de le lire, et affiche un message d'erreur et retourne si le fichier n'existe pas. Si aucun ensemble fréquent n'est trouvé avec le seuil de support spécifié, le script affiche un message indiquant cela et retourne.

Exécution de l'algorithme :

```
PS C:\Users\Ahmadou\mining> python .\AssociationRule.py
```

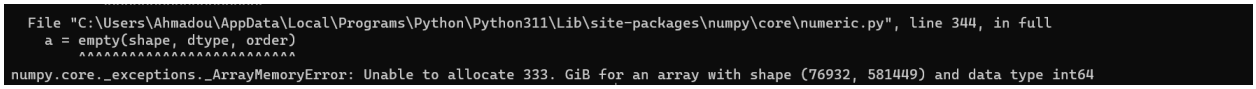
L'algorithme va nous générer le fichier result1.txt qui contient le résultat de l'exécution de l'algorithme Apriori.

```
items,support,ordered_statistics
frozenset({'\x00'}),1.0,"[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'\x00'}), confidence=1.0, lift=1.0)]"
frozenset({' '}),0.9982078853046595,"[OrderedStatistic(items_base=frozenset(), items_add=frozenset({' '}), confidence=0.9982078853046595, lift=1.
frozenset({'1'}),0.5985663082437276,"[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'1'}), confidence=0.5985663082437276, lift=1.
frozenset({'2'}),0.7634408602150538,"[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'2'}), confidence=0.7634408602150538, lift=1.
frozenset({'3'}),0.8315412186379928,"[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'3'}), confidence=0.8315412186379928, lift=1.
frozenset({'4'}),0.53584229390681,"[OrderedStatistic(items_base=frozenset(), items_add=frozenset({'4'}), confidence=0.53584229390681, lift=1.0)]"
```

résultat exécution apriori (cf result1.txt)

Les résultats présentent les ensembles d'éléments uniques qui ont été trouvés ainsi que leur support (la fréquence à laquelle ils apparaissent dans l'ensemble de données) et les statistiques d'association qui mesurent la probabilité d'apparition conjointe des éléments. Le support des éléments simples (tels que '\x00', ' ', '1', '2', '3', '4', '9') est présenté en pourcentage, ce qui signifie que ces éléments apparaissent respectivement dans 100%, 99,8%, 59,9%, 76,3%, 83,1%, 53,5% et 50% des enregistrements. Les ensembles d'éléments multiples (tels que '\x00', ' ' ou '\x00', '1') ont également été trouvés et leur support est présenté ainsi que les statistiques d'association. Par exemple, le premier ensemble '\x00', ' ' apparaît dans 99,8% des enregistrements et il a une confidence de 0,998, ce qui signifie que s'il y a '\x00' dans un enregistrement, il y aura probablement aussi ' ' dans cet enregistrement. Le lift est de 1,0, ce qui signifie qu'il n'y a pas de corrélation entre ces deux éléments.

NB : Pour l'exécution de l'algorithme nous avions travaillé sur une portion de notre fichier qui contient les pmdi et les références qui leurs sont associés, car lorsqu'on a essayé sur le fichier d'origine (32 Mo) on a un dépassement de capacité.



L'erreur indique que votre programme a essayé d'allouer plus de mémoire qu'il n'en est disponible dans votre système. Dans ce cas particulier, le programme a essayé d'allouer 333 gigaoctets de mémoire pour stocker un tableau NumPy qui a une forme de (76932, 581449) et un type de données int64. La quantité de mémoire disponible dans votre système dépend de plusieurs facteurs, tels que la quantité de mémoire physique (RAM) installée sur votre ordinateur et la quantité de mémoire virtuelle disponible. Dans ce cas, il est probable que votre système n'ait pas suffisamment de mémoire pour allouer la quantité de mémoire demandée par votre programme.