

Université De Montpellier  
Faculté Des Sciences



**Niveau :** Master 2

**Module :** Gestion des données au delà de SQL (NoSQL)

**HAI914I**

---

## TP : Moteur KnowledgeGraph RDF

---

*Supervisé par :*  
M. Federico Ulliana  
M. Rodriguez Olivier  
M. François Scharffe

*Réalisé par :*  
ALLOUCH Yanis  
KACI Ahmed

2021/2022

# Table des matières

1	Création du dictionnaire . . . . .	2
2	Création des indexes . . . . .	2
3	Évaluation des requêtes en étoiles . . . . .	3
3.1	Détails de l'implémentation de la première étape de l'évaluation de requête . . . . .	3
3.2	Détails de l'implémentation de la deuxième étape de l'évaluation de requête . . . . .	3
3.3	Détails de l'implémentation de la troisième étape de l'évaluation de requête . . . . .	4
3.4	Détails de l'implémentation de la quatrième étape de l'évaluation de requête . . . . .	4
4	Options d'exécution implémentées . . . . .	4
5	Analyse des bancs d'essais . . . . .	5
6	Hardware et Software . . . . .	7
7	Métriques, Facteurs et Niveaux . . . . .	7
8	Évaluation des performances . . . . .	8
9	Plan des tests . . . . .	8

# 1 Création du dictionnaire

Pour l'implémentation du dictionnaire, nous avons créé une classe *Dictionary* contenant une *HashMap* où chaque élément est composé, d'une part, d'une clé de type *String* correspondant soit au sujet, au prédicat ou à l'objet d'une ressource RDF. D'autre part, la valeur de chaque élément de la *HashMap* est un objet de type *Long* représentant un index qui sera utilisé pour l'implémentation de l'hexastore index.

Nous avons aussi utilisé dans cette classe *Dictionary* une seconde *HashMap* qui représente l'inverse du dictionnaire (l'index comme clé et le sujet ou le prédicat ou l'objet comme valeur) afin de récupérer un élément à partir de son index lors de la restitution des résultats des requêtes.

# 2 Création des indexes

Pour l'implémentation des six types de l'index *Hexastore* nous avons réalisé une architecture basée sur le polymorphisme dans laquelle on possède une classe abstraite nommée *HexastoreIndex* qui est la super classe de six classes concrètes à savoir : *SOPIndex*, *SPOIndex*, *POIndex*, *PSOIndex*, *OPSIndex*, *OSPIIndex*.

Dans la classe abstraite *HexastoreIndex*, nous représentons la structure générale de l'index comme suit :

```
1 Map<Long, Map<Long, List<Long>>> mapIndex;
```

Listing 1 – Représentation général de l'index Hexastore

Cette classe possède aussi une méthode abstraite permettant d'ajouter un élément dans cette structure selon l'ordre spécifier dans la classe qui l'implémente, par exemple dans le cas où on instancie un *SPOIndex* l'implémentation de cette méthode permettra d'ajouter dans la *mapIndex* des éléments ayant la structure suivante :

```
1 <IndiceSujet, Map<IndicePredicat, List<IndiceObjet>>>
```

Listing 2 – Structure d'un élément ajouté à l'index

Nous avons aussi implémenté dans les classes *POIndex* et *OPSIndex* une méthode permettant de récupérer la liste des index des sujets correspondant à un index de prédicat et un index d'objet fournis en paramètre. Cette méthode sera utilisée pour l'obtention des résultats des requêtes.

## 3 Évaluation des requêtes en étoiles

Afin de réaliser l'évaluation des requêtes nous avons créé les trois classes suivantes :

- La classe *ParsingAndDictionaryProcessor* contenant les méthodes permettant de réaliser le chargement de la base de triplets et la création d'un dictionnaire pour les ressources.
- La classe *IndexingProcessor* contenant les méthodes permettant de réaliser la création des index.
- La classe *QueryProcessors* contenant les méthodes permettant la lecture des requêtes en entrée et le calcul de leurs résultats qui sont accessibles pour la visualisation dans le fichier intitulé *export\_query\_result.csv*.

Nous avons implémenté une classe permettant de charger l'intégralité des ressources RDF de la base, afin de l'utiliser dans l'implémentation du dictionnaire et des différents index. Nous avons choisi cette solution au lieu de remplir directement le dictionnaire et les index lors de la lecture des données dans le but de respecter le cahier des charges où il est mentionné que la première étape de l'évaluation des requêtes consiste en le chargement de la base de triplets et création d'un dictionnaire pour les ressources, puis dans la deuxième étape la création des index. Par ailleurs, nous avons effectué ce choix, car cette séparation des étapes permet d'avoir un code plus compréhensible dans lequel il est facile de faire les différents calculs des temps d'exécution tout en prenant en compte que les données chargées en mémoire, fournies dans le cadre de ce projet, ne sont pas volumineuses au point de saturer la RAM.

### 3.1 Détails de l'implémentation de la première étape de l'évaluation de requête

Dans cette étape qui consiste en le chargement de la base de triplets et création d'un dictionnaire pour les ressources. Nous avons implémenté dans la classe *ParsingAndDictionaryProcessor* une méthode qui réalise cette fonctionnalité en faisant appel à deux autres méthodes : une pour le chargement de la base et l'autre pour la création du dictionnaire en se basant sur les données chargées.

### 3.2 Détails de l'implémentation de la deuxième étape de l'évaluation de requête

Dans cette étape qui consiste en la création des index. Nous avons implémenté dans la classe *IndexingProcessor* une méthode qui réalise cette fonctionnalité en lui fournissant la BDD chargée, le dictionnaire rempli et le type de l'index souhaité.

### 3.3 Détails de l'implémentation de la troisième étape de l'évaluation de requête

Dans cette étape, nous utilisons la méthode *parseQueries* de la classe *QueryProcessors* pour lire les requêtes en entrées.

### 3.4 Détails de l'implémentation de la quatrième étape de l'évaluation de requête

Dans cette étape qui consiste en l'accès aux données et la visualisation des résultats. À chaque fois qu'une requête est lue, nous appelons la méthode *processAQuery* qui calcule le résultat attendu en commençant par retrouver l'ensemble des solutions pour le premier patron de triplet qui servira comme base pour filtrer ultérieurement les valeurs récupérées pour le deuxième patron, et ainsi de suite. Et ce, en utilisant le dictionnaire et les index pour l'accès aux données. Pour la visualisation des résultats, nous exportant ses derniers dans un fichier CSV nommé *export\_query\_result.csv* qui est créé dans le dossier *data* du projet après son exécution.

## 4 Options d'exécution implémentées

Nous avons implémenté toutes les options d'exécutions, proposés dans le sujet. C'est-à-dire, les options suivantes :

1. queries,
  - Le fichier contenant les requêtes.
2. data,
  - Le fichier contenant les données.
3. output,
  - Le fichier CSV contenant les temps d'exécutions.
4. export\_query\_results.
  - Le fichier contenant les résultats des requêtes.



## 5 Analyse des bancs d'essais

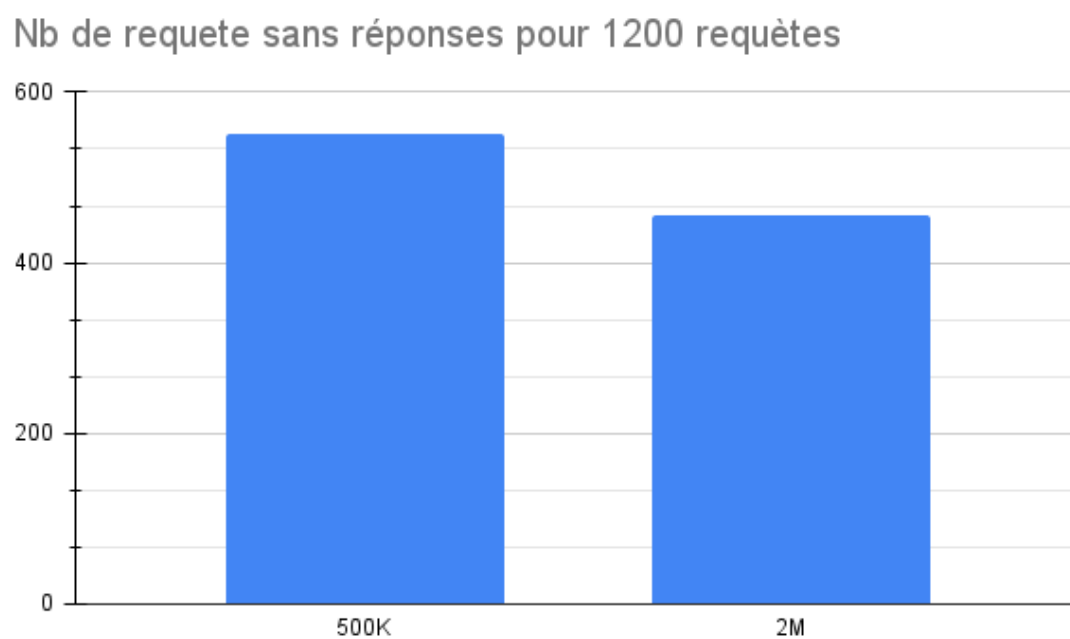


FIGURE 1 – le nombre de réponses aux requêtes sur une instance de 500K et de 2M triples

### Les requêtes avec un même nombre de patrons

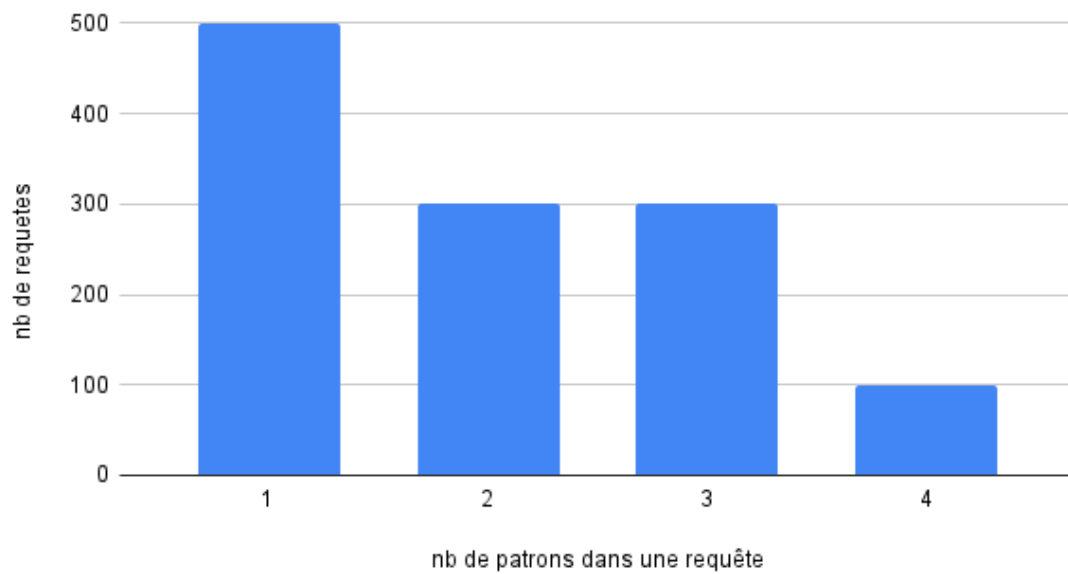


FIGURE 2 – le nombre de requêtes ayant zéro réponses

### Les doublons dans les requêtes

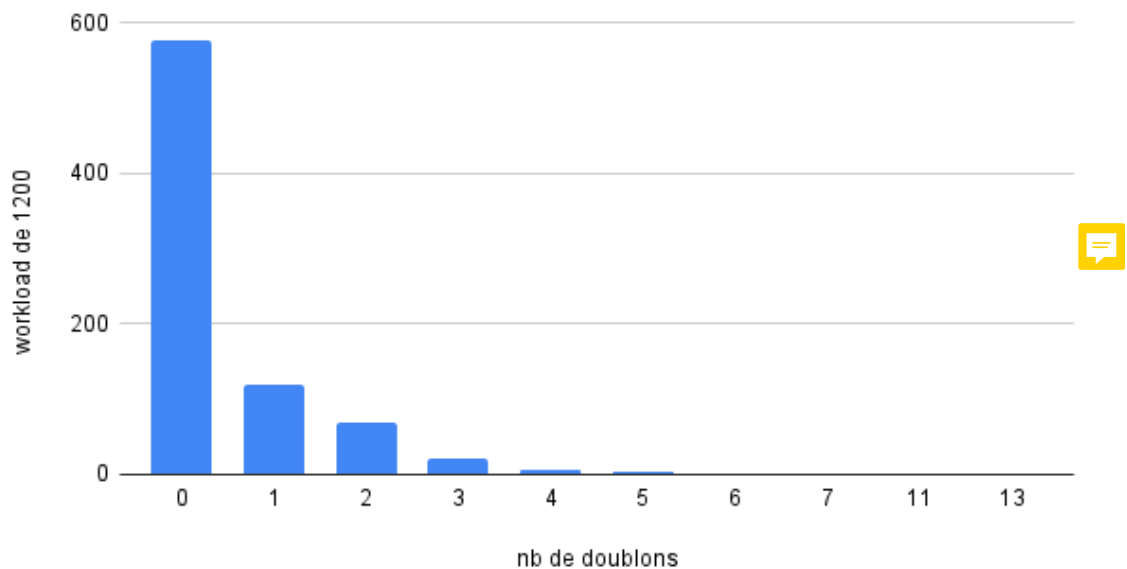


FIGURE 3 – le nombre de requêtes avec un même nombre de conditions (patrons de requêtes)

## 6 Hardware et Software

Nous allons utiliser le hardware suivant :



- Processor : AMD Ryzen 5 2600X Six-Core Processor 3.60 GHz,
- Installed RAM : 32,0 GB,
- System type : 64-bit operating system, x64-based processor,
- Hard Drives :
  - SSD : 120 GB SATA

Nous allons utiliser le software suivant :


- OS : Ubuntu 20.04 LTS,
- Notre application est codé en Java,
- Java JDK : 1.8.

## 7 Métriques, Facteurs et Niveaux


Les métriques sont les suivantes :

1. Temps d'exécution, 
2. Nombre de requêtes,
3. Qualité de la réponse (% de réponses valides), 
4. Rendement (query par sec.).

Les facteurs sont les suivants :

- CPU,
- RAM, 
- Hard drives,
- Workload (queries),
- Dataset (data).

Les niveaux en fonction des facteurs sont les suivants :


- Workload :
  1. 1200 requetes, 
  2. 10K requetes,
  3. 100K requetes.
- Dataset :
  1. 100K,
  2. 500K,
  3. 2M,



Les facteurs importants sont les suivants par ordre d'importance, parce que nous l'avons observé sur les résultats des temps d'exécution :


- Workload,
- Dataset.

Les facteurs secondaires sont les suivants puisqu'on ne peut pas influencer leurs niveaux :

- CPU,
- RAM, 
- Hard drives.

## 8 Évaluation des performances

Les métriques cold :

1. Temps d'exécution, 
2. Qualité de la réponse (% de réponses valides),
3. Rendement (query par sec.).

Les métriques warm :

Les métriques sont les suivantes :



1. Temps d'exécution,
2. Qualité de la réponse (% de réponses valides),
3. Rendement (query par sec.).

## 9 Plan des tests