Universités de Montpellier

Faculté des Sciences de Montpellier

Département d'informatique

HAI916I

Gestion des données au-delà de SQL

Moteur KnowledgeGraph RDF

Rendu N°2

Réalisé Par :

RAMDANE Souhaib (M2 GL)

MEFLAH Wided (M2 IASD)

Encadré par :

Mr. Federico Ulliana

2023/2024

Table des matières

1. Introduction

Au sein de ce rapport, nous entreprenons d'expliquer en détail le mécanisme d'évaluation des requêtes issues d'un fichier spécifique sur un ensemble de données défini. Cette démarche requiert la mise en œuvre d'un dictionnaire ainsi que la configuration d'un index hexastore. Nous allons explorer en profondeur cette procédure, mettant en lumière les différentes étapes et les méthodes utilisées pour cette évaluation.

2. Création du dictionnaire

Pour créer le dictionnaire, une classe nommée Dictionary a été mise en place. Cette classe contient une TreeMap où chaque entrée est constituée d'une clé de type String, représentant le sujet, le prédicat ou l'objet d'une ressource RDF, et d'une valeur de type Integer. Cette valeur correspond à un index utilisé ultérieurement pour l'implémentation de l'hexastore index.

De plus, dans la classe Dictionary, une seconde TreeMap est utilisée pour représenter l'inverse du dictionnaire. Ici, l'index est utilisé comme clé et le sujet, le prédicat ou l'objet est associé comme valeur. Cette configuration permet de récupérer un élément à partir de son index lors de la restitution des résultats des requêtes.

Ci-dessous, un exemple de données codées et décodées utilisant notre index :

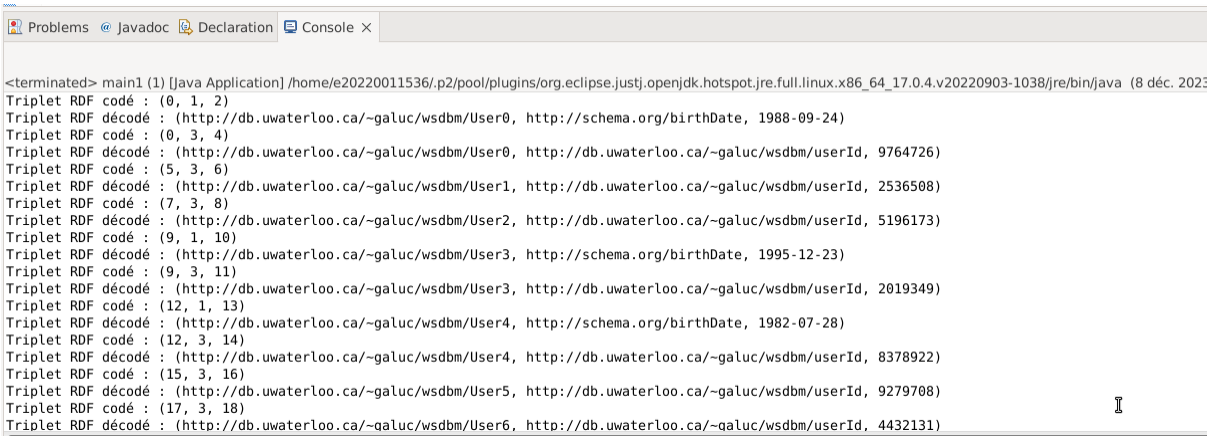

Figure 1 : Trace de la création du dictionnaire des données

3. Création des indexes

Pour mettre en place l'implémentation des six types de l'index Hexastore, nous avons créé une classe nommée Triplet qui représente un type d'hexastore (SPO, OPS, etc.). Cette classe contient la structure suivante, représentant notre triplet :

```
private final Map<Integer, Map<Integer, List<Integer>>> tripletsIndex;
```

La classe Triplet offre une solution robuste pour la gestion des triplets, avec des fonctionnalités optimisées pour l'insertion et la récupération.

De plus, nous avons conçu la classe Hexastore pour représenter un index hexastorique, une structure de données optimisée destinée au stockage et à la recherche de triplets dans un ensemble de données. Cette classe utilise six instances de la classe Triplet, chacune correspondant à une permutation différente des composants du triplet (sujet, prédicat, objet).

```
public class Hexastore {

    private static Hexastore indexCollection_instance = null;
    Triplet SPO = new Triplet();
    Triplet SOP = new Triplet();
    Triplet PSO = new Triplet();
    Triplet POS = new Triplet();
    Triplet OSP = new Triplet();
    Triplet OPS = new Triplet();
```

4. Évaluation des requêtes en étoiles

Dans le processus d'évaluation de nos requêtes, plusieurs étapes clés sont mises en œuvre. Tout d'abord, lors de la phase de data parsing, un dictionnaire est créé et rempli pour assurer la correspondance entre les valeurs textuelles des triplets et leurs représentations codées. Cette étape préliminaire est cruciale pour le codage ultérieur des paramètres de requête.

Ensuite, lors de la lecture et du traitement de chaque requête, le processus de traitement comprend trois étapes essentielles. Tout d'abord, chaque paramètre de la requête (sujet, prédicat, objet) est chiffré en utilisant le dictionnaire établi précédemment. Ensuite, en fonction des éléments à retourner dans la requête, le programme sélectionne le Hexastore correspondant parmi les six instances disponibles. Dans notre cas, nous avons déterminé l'utilisation de POS, qui organise les triplets selon la permutation prédéfinie prédicat-objet-sujet car les requêtes utilisées ne cherchent à retourner que des sujets.

En troisième lieu, pour chaque motif de triplet extrait de la requête, l'accès à la structure POS est effectué pour récupérer les sujets correspondants.

La phase cruciale de l'intersection intervient ici, où les réponses de chaque paire de motifs sont comparées. Si l'intersection est vide, cela signifie qu'il n'y a pas de réponse, sinon, les résultats sont décodés à l'aide du dictionnaire et exportés dans un fichier csv selon la structure suivante: (num_requete, requete , notre resultats , resultats jena).

```
-----------------------------------------------Une requete-----------------------------------------------
        ----  Condition  ------------
              o         = 1988-09-24
              p         = http://schema.org/birthDate
              o Chiffre =  2
              p Chiffre =  1
              reponse indiv  =  [0]
        ---- END Condition  ------------
        ----  Condition  ------------
              o         = 9764726
              p         = http://db.uwaterloo.ca/~galuc/wsdbm/userId
              o Chiffre =  12222
              p Chiffre =  5
              reponse indiv  =  [0]
        ---- END Condition  ------------
la reponse coder est  =  [0]
la reponse decoder est  =  [http://db.uwaterloo.ca/~galuc/wsdbm/User0]
-----------------------------------------------END requete-----------------------------------------------
```

Figure 2 : Trace d'évaluation des requêtes en étoiles

Nous avons comparé nos résultats avec ceux produits par Jena en examinant à la fois le nombre total de résultats retournés et le nombre de résultats par requête. Nous avons constaté qu'ils étaient identiques.

	A	B	C	
	Num	Requete	Notre Resultat	Result Jena
2	1	SELECT ?v0 WHERE {?v0 <http://schema.org/eligibleRegion> <http://db.uwaterloo.ca/~galuc/wsdbı	[http://db.uwaterloo.ca/~galuc/wsdbm/Offer642, http://db.uwaterloo.ca/~galuc/wsdbm/Offer195	[http://db.uwaterloo.ca/~galuc/wsdbm/Offer
3	2	SELECT ?v0 WHERE {?v0 <http://schema.org/eligibleRegion> <http://db.uwaterloo.ca/~galuc/wsdbı	[]	[]
4	3	SELECT ?v0 WHERE {?v0 <http://schema.org/eligibleRegion> <http://db.uwaterloo.ca/~galuc/wsdbı	[http://db.uwaterloo.ca/~galuc/wsdbm/Offer469, http://db.uwaterloo.ca/~galuc/wsdbm/Offer821	[http://db.uwaterloo.ca/~galuc/wsdbm/Offer
5	4	SELECT ?v0 WHERE {?v0 <http://schema.org/eligibleRegion> <http://db.uwaterloo.ca/~galuc/wsdbı	[http://db.uwaterloo.ca/~galuc/wsdbm/Offer559, http://db.uwaterloo.ca/~galuc/wsdbm/Offer424	[http://db.uwaterloo.ca/~galuc/wsdbm/Offer
6	5	SELECT ?v0 WHERE {?v0 <http://schema.org/eligibleRegion> <http://db.uwaterloo.ca/~galuc/wsdbı	[http://db.uwaterloo.ca/~galuc/wsdbm/Offer63, http://db.uwaterloo.ca/~galuc/wsdbm/Offer558,	[http://db.uwaterloo.ca/~galuc/wsdbm/Offer
7	6	SELECT ?v0 WHERE {?v0 <http://schema.org/eligibleRegion> <http://db.uwaterloo.ca/~galuc/wsdbı	[http://db.uwaterloo.ca/~galuc/wsdbm/Offer797, http://db.uwaterloo.ca/~galuc/wsdbm/Offer356	[http://db.uwaterloo.ca/~galuc/wsdbm/Offer
8	7	SELECT ?v0 WHERE {?v0 <http://schema.org/eligibleRegion> <http://db.uwaterloo.ca/~galuc/wsdbı	[]	[]
9	8	SELECT ?v0 WHERE {?v0 <http://schema.org/eligibleRegion> <http://db.uwaterloo.ca/~galuc/wsdbı	[http://db.uwaterloo.ca/~galuc/wsdbm/Offer558, http://db.uwaterloo.ca/~galuc/wsdbm/Offer739	[http://db.uwaterloo.ca/~galuc/wsdbm/Offer
10	9	SELECT ?v0 WHERE {?v0 <http://schema.org/eligibleRegion> <http://db.uwaterloo.ca/~galuc/wsdbı	[http://db.uwaterloo.ca/~galuc/wsdbm/Offer319, http://db.uwaterloo.ca/~galuc/wsdbm/Offer397	[http://db.uwaterloo.ca/~galuc/wsdbm/Offer
11	10	SELECT ?v0 WHERE {?v0 <http://schema.org/eligibleRegion> <http://db.uwaterloo.ca/~galuc/wsdbı	[http://db.uwaterloo.ca/~galuc/wsdbm/Offer186, http://db.uwaterloo.ca/~galuc/wsdbm/Offer818	[http://db.uwaterloo.ca/~galuc/wsdbm/Offer

Figure 3 : Extrait du fichier CSV des résultats

5. Conclusion

Dans les prochaines étapes, nous allons analyser les performances de notre prototype. Nous allons spécifiquement évaluer et comparer les performances de notre système en utilisant le benchmark WatDiv.