Daniel Azevedo Gomes, Sebastien Prud'homme Gateau

TP2 : Évaluation des requêtes en étoile

1. Introduction :

Ce projet s'inscrit dans le cadre de l'UE HAI914I : Gestion des données au-delà de SQL (NoSQL) et a été réalisé par Daniel Azevedo Gomes et Sébastien Prud'homme Gateau.

Le code produit est disponible sur git : https://github.com/dneial/mini-rdf

et est une complétion du squelette de code fourni sur moodle

2. Structures implémentées

Trois structures ont été créées pour répondre aux consignes demandées :

- Un Dictionnaire bi-directionnel, mettant en relation un objet, sujet ou prédicat avec son encodage.
- Un ensemble d'index, dans lesquels seront stockés triplets RDF sous forme arborescente.
- Un Hexastore qui regroupe les 6 index définis précédemment (SPO, SOP, POS, PSO, OSP, OPS) dans l'optique de pouvoir gérer différents types de requêtes.

3. Chargement de la base

Lors de la mise en route de notre application, une opération d'initialisation de la base de données et des structures associées se lance. Lors d'un seul passage sur le fichier contenant nos données on effectue les actions suivantes :

- Encodage et insertion du Sujet, Prédicat et Objet dans le dictionnaire
- Insertion du triplet courant dans le hexastore
    - Ce dernier va gérer lui-même l'ajout de nos éléments dans les index selon leurs structures respectives.

Cette redondance de données va nous permettre de traiter efficacement l'ensemble des requêtes qui nous sera fourni.

4. Lecture des requêtes en entrée.

Une fois la base de données initialisée, on peut procéder au traitement de requêtes SPARQL. Notre programme parse un fichier regroupant des requêtes puis charge chacune d'entre elles en mémoire.

5. Évaluation des requêtes

Les requêtes récupérées sont en étoile, elles sont composées de triplets RDF partageant tous un élément commun à récupérer. Dans le cadre de notre projet toutes les requêtes traitées sont de la forme SPO et les variables à récupérer dans la base sont des Sujets.

Pour chaque pattern reconnu dans notre requête, on récupère le prédicat et l'objet, et grâce à leur indexation, on peut facilement retrouver les sujets en lien avec ces deux éléments. On réalise l'intersection entre les sujets renvoyés par chaque pattern. Dans le cas ou l'un d'entre eux ne renvoie rien, il n'est pas nécessaire de continuer car le retour sera vide. L'intersection entre les ensembles de sujets trouvés est la réponse renvoyée par notre moteur de requête.

6. Options implémentées

Comme demandé, notre moteur supporte les options suivantes en paramètre :

1. Jena

"Apache Jena est un framework Java gratuit et open source pour la construction des applications du web sémantique et de l'annotation sémantique. Il est composé de différentes APIs interagissant ensemble pour traiter les données RDF."

Dans le cadre de ce projet, nous avons utilisé Jena pour vérifier la correction des résultats de notre application. Lancer l'exécution avec l'option "*-Jena*" permet d'utiliser cet outil comme un moteur de requêtes : on lui fournit les mêmes données et requêtes que le moteur implémenté, puis on vérifie simplement que les deux renvoient les mêmes résultats pour toutes les requêtes.

2. Warm

L'option *-warm* a pour vocation de simuler un environnement d'exécution "chaud". Il va permettre au moteur de charger les données et le fichier de requêtes, choisir un échantillon aléatoire de requêtes correspondant au pourcentage passé en paramètre, puis les exécuter. Il est ensuite possible de procéder à une exécution normale de toutes les requêtes chargées.

3. Shuffle

Lancer le moteur avec *-shuffle* modifie aléatoirement l'ordre des requêtes chargées en mémoire.

4. Output

L'option "-*output*" permet de visualiser différentes mesures associées au fonctionnement de notre outil :

- nom du fichier de données
- nom du dossier des requêtes
- nombre de triplets RDF
- nombre de requêtes
- temps de lecture des données (ms)
- temps de lecture des requêtes (ms)
- temps de création du dictionnaire (ms)
- nombre d'index
- temps de création des index (ms)
- temps total d'évaluation du workload (ms)
- temps total (du début à la fin du programme) (ms)

Toutes ces informations sont affichées dans la console puis écrites dans un fichier CSV après l'exécution de notre programme. Le dossier de destination doit être passé en paramètre de cette option.

5. Export des résultats

L'option "-*export_results*" permet de stocker les requêtes exécutées ainsi que l'ensemble des résultats dans un fichier CSV. Le dossier de destination doit également être passé en paramètre de cette option.