HAI914I - Un moteur d'évaluation de requêtes en étoile

<federico.ulliana@inria.fr>
<guillaume.perution-kihli@inria.fr>

Implementation d'un mini moteur de requêtes en étoile

L'objectif du TD est d'implémenter l'approche hexastore pour l'interrogation des données RDF vue en cours, ainsi que les procédures nécessaires à l'évaluation de requêtes en étoile (exprimées en syntaxe SPARQL) qui seront introduites par la suite.

Le TD suivant sera dédié à l'évaluation des performances du système réalisé. Votre prototype sera comparé avec l'évaluateur de requête du logiciel libre Integraal. [1] De plus, Integraal peut être considéré comme un système "oracle" pour vérifier la correction et complétude de votre moteur, c'est-à-dire, vérifier que votre moteur donne toutes et seules les bonnes réponses à une requête. Celle-ci est évidemment une condition nécessaire à la conduite d'expériences.

Consignes

— Le TD se fait en groupes de 1 ou 2 personnes.
Lorsque vous avez décidé quel sera votre groupe inscrivez-vous dans le tableau correspondant à votre encadrant de TD suivant ce lien :
https://docs.google.com/spreadsheets/d/1bQY-Xba11DNcbCC5gujvPnidhajN-Kcb3O6PjXveF7M/edit?usp=sharing
— Le langage de programmation est imposé : Java.

Le projet est divisé en 3 rendus :	Dates de rendu
1. Dictionnaire, index (Hexastore) : rendu du code (pas de rapport)	15 Novembre
2. Évaluation des requêtes en étoile : code + rapport 3 pages	29 Novembre
3. Analyse des performances : code + rapport 5 pages	13 Décembre

Il n'y aura pas de soutenance. Toutefois, il sera demandé aux groupes d'expliquer le travail réalisé lors des séances de TD. Ainsi, la présence en TD est obligatoire.

Évaluation

Les points suivants seront sujet à évaluation.
— Travail réalisé (code fonctionnel, implémentation des fonctionnalités)

1. https://rules.gitlabpages.inria.fr/integraal-website/

— Qualité du code produit (notamment, taux du couverture du code par des tests unitaires)
— Clarté et concision du rapport

Point de départ du projet logiciel (il est important de lire toute la page)

Squelette du projet

On vous met à disposition un squelette du projet à réaliser dans ce dépôt git :

https://gitlab.etu.umontpellier.fr/p00000415795/nosql-engine-skeleton

Vous pouvez réutiliser ce projet (ce qui est conseillé), ou juste l'étudier comme exemple.
Pour s'assurer que tout fonctionne bien il suffit d'exécuter le programme de la classe `Example.java`.[2]

Déscription des principales classes fournies

`Example.java` montre simplement comment on peut lire les données et les requêtes depuis des fichiers mis à disposition via des parseurs dédiés.

`RDFAtom.java` est la classe representant des triplets RDF à utiliser

`StarQuery.java` est la classe representant des requêtes en étoile à utiliser

`RDFStorage.java` est l'interface qui décrit le contrat (cad, l'ensemble des méthodes supportées) d'un objet Hexastore. **Attention** : votre implémentation doit implémenter cette interface ; vous êtes bien évidemment libres d'étendre l'interface à souhait.

IDE

— Si votre poste informatique ne dispose pas du logiciel Eclipse, vous pouvez l'installer en suivant cette procédure
https://moodle.umontpellier.fr/mod/page/view.php?id=131074
Nous avons besoin de la version intitulée *Eclipse IDE for Java Developers*.
— Vous pouvez également utiliser IntelliJ, soit la version communautaire qui est gratuite, soit la version Ultimate en demandant la licence gratuite pour les étudiants.
https://www.jetbrains.com/idea/download/?section=linux

2. Si vous rencontrez des soucis avec le log, vous pouvez supprimer le fichier `logback.xml` dans le repertoire `java/resources`

Version de Java

Ce projet est prévu pour utiliser Java 21. Libre à vous d'utiliser des versions plus récentes si vous êtes vraiment à l'aise avec l'environnement Java. L'idée est quand même de ne pas perdre de temps avec des considérations techniques.

Dépendances Java avec Maven

Le projet utilise Maven pour la gestion des dépendances Java. Ainsi, utilisez l'import approprié pour votre IDE (e.g., "Import Maven Project" pour Eclipse, IntelliJ).

Le projet contient un fichier special `pom.xml` qui permet d'importer toutes bibliothèques nécessaires au projet ; il n'est normalement pas nécessaire d'y toucher.

Si Eclipse ne reconnaît pas ce fichier `pom.xml` et n'arrive pas à charger les dépendances, c'est qu'il y a un problème avec son plugin maven (m2e).
On peut alors tenter de chercher le menu suivant (il peut ne pas exister) :

```
Fenêtre "Package Explorer" (toute à gauche)/
clic droit sur le projet/
Configure/Convert to Maven Project.
```

Bibliothèques externes

Le projet utilise plusieurs bibliothèques, notamment :

— rdf4j[3] 3.7.3 pour lire les données rdf et requêtes sparql. Pour en apprendre davantage sur la bibliothéque RDF4J, il faudra se référer à la documentation[3] ainsi qu'à sa Javadoc[4].

— Integraal[5] 1.6.0 qui sert de base pour les implémentations et sera utilisée pour comparaison dans la phase *d'analyse des performances*

Tests unitaires

Un ensemble de tests unitaires sont mis à disposition dans `src/test/java`. Ces tests vous permettent de mieux comprendre le comportement attendu des objets.

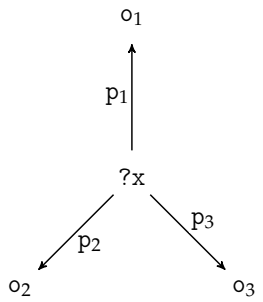
Jeu de données et requêtes

Dans le répertoire `data/` du projet, on vous met à disposition des fichiers permettant de réaliser des micro-tests (`sample_data.nt` et `sample_query.queryset`) ainsi que des fichiers contenant plus de données et de requêtes (`100K.nt` et `STAR_ALL_workload.queryset`).

3. https://rdf4j.org/documentation
4. https://rdf4j.org/javadoc/latest/
5. https://rules.gitlabpages.inria.fr/integraal-website/

Les requêtes en étoile

Les requêtes en étoile constituent la base des requêtes SPARQL (et plus en général des requêtes sur des graphes). Une requête SPARQL en étoile est composée de n patrons de triplet tous partageant une même variable commune. Un exemple de requête en étoile est `SELECT ?x WHERE { ?x` p_1 o_1 `. ?x` p_2 o_2 `. ?x` p_3 o_3 `}` dont la représentation graphique est la suivante.



Voici quelques exemples d'interrogations exprimables par des requêtes en étoile.

Q_1 : *quels écrivains ayant reçu un prix Nobel sont aussi des peintres ?*

Q_2 : *quels sont les artistes nés en 1904 ayant vécu à Paris ?*

Q_3 : *qui sont les amis d'Alice qui aiment le cinéma ?*

Pour Q_1 nous aurons la requête

```
SELECT ?x WHERE { ?x type peintre . ?x won_prize Nobel . ?x type écrivain }
```

Similairement, pour Q_2, nous aurons

p_1 `= type,` o_1 `= artiste,` p_2 `= lives_in,` o_2 `= Paris,` p_3 `= birth_year,` o_3 `= 1904`

Enfin, pour Q_3, nous aurons

```
SELECT ?x WHERE {?x friendOf Alice . ?x likes Movies}
```

Bien entendu, le système résultant du projet doit être capable de gérer des requêtes étoile avec n branches, et n'importe quelle valeur pour les constantes p_i et o_i. Enfin, on assumera que les variables n'apparaissaient que dans les sujets (et jamais au niveau des propriétés ni des objets).

L'évaluation des requêtes

Nous identifions quatre étapes fondamentales dans l'évaluation des requêtes en étoile.

1. Le chargement des triplets (déjà implémenté) et leur encodage dans un dictionnaire (à implémenter)
2. Creation de l'hexastore avec ses indexes
3. Lecture des requêtes en entrée (en utilisant le parser de requête en étoile déjà implémenté)
4. Accès aux données et visualisation des résultats.

Important : Pour chaque étape que vous devez implémenter, vous devez au préalable écrire des tests unitaires pour chaque méthode. Écrivez le résultat attendu pour différentes entrées, en faisant attention à bien traiter tous les cas limites, puis rédigez le test unitaire correspondant avant d'implémenter la méthode. Le test unitaire vous permettra de vérifier la correction de votre implémentation. Vérifiez bien que votre test couvre toutes les branches d'exécution de la méthode en utilisant la fonctionnalité de couverture du code (*code coverage*) de votre IDE.

Nous allons maintenant détailler chaque étape.

Le dictionnaire (rendu 15 Novembre)

Le dictionnaire associe un entier à chaque ressource de la base RDF, afin d'en permettre un stockage compact. Par exemple, on voit les triplets `<Bob, knows, Bob>` et `<1,2,1>` indistinctement avec la correspondance {(1,Bob) , (2,knows)}. Le dictionnaire doit vous permettre d'encoder et décoder les triplets de manière efficace.

→Le dictionnaire est utilisé par la méthode `add(RDFAtom a)` de l'Hexastore.

L'index (rendu 15 Novembre)

L'index permet une évaluation efficace des requêtes et est adapté au système de persistance choisi.

Dans ce projet, on vous demande d'implémenter l'approche hexastore pour l'indexation des données.

→L'index est utilisé par la méthode `add(RDFAtom a)` de l'Hexastore pour ce qui concerne l'insertion de données, ains que par la méthode `match(RDFAtom a)` pour ce qui concerne l'interrogation des données.

L'accès aux données (rendu 29 Novembre)

L'accès aux données se fait par les structures de données mises en oeuvre. Une fois retrouvé l'ensemble des solutions pour le premier patron de triplet, cela nous servira pour filtrer ultérieurement les valeurs récupérées pour le deuxième patron, et ainsi de suite. Il est demandé d'implémenter une méthode spécifique pour l'évaluation des requêtes en étoile.

→L'évaluation des requêtes est utilisée par la méthode `match(StarQuery q)` de l'Hexastore.

Vérification de correction et complétude (rendu 29 Novembre)

Vous devez implémenter une procédure qui compare la liste des résultats de votre système avec ceux de Integraal, qu'on utilise ici comme un "oracle". Il s'agit d'une étape fondamentale pour ensuite passer à l'analyse des performances du système. Pour la comparaison, vous pouvez utiliser le SimpleInMemoryGraphStore pour le stockage des triplets, et utiliser la méthode "asFOQuery" de la classe "StarQuery" pour la convertir en requête évaluable par l'évaluateur générique. Un exemple est fourni dans la classe Example.

Consignes :

— Implémenter indexation, et accès aux données du moteur de requêtes.
— Les données seront stockées en mémoire vive (ram). [6]

Lecture des entrées et export des résultats :

1. Comme déjà expliqué, vous trouverez sur le dépôt git un squelette du programme à réaliser que vous pouvez réutiliser comme base de votre projet. Vous y trouverez comment lire un fichier de données dans la classe Example.
2. Important : votre système doit être orienté vers l'évaluation d'un *ensemble de requêtes* (et non pas d'une seule requête) sur un (seul) fichier de données. Les résultats seront exportés dans un répertoire dédié.

6. rien nous empêche de prévoir un système de persistance en mémoire secondaire comme extension possible du travail.