rapport (évaluation des requetes en étoile, dictionnaire et indexation)

Doha CHEMAOU

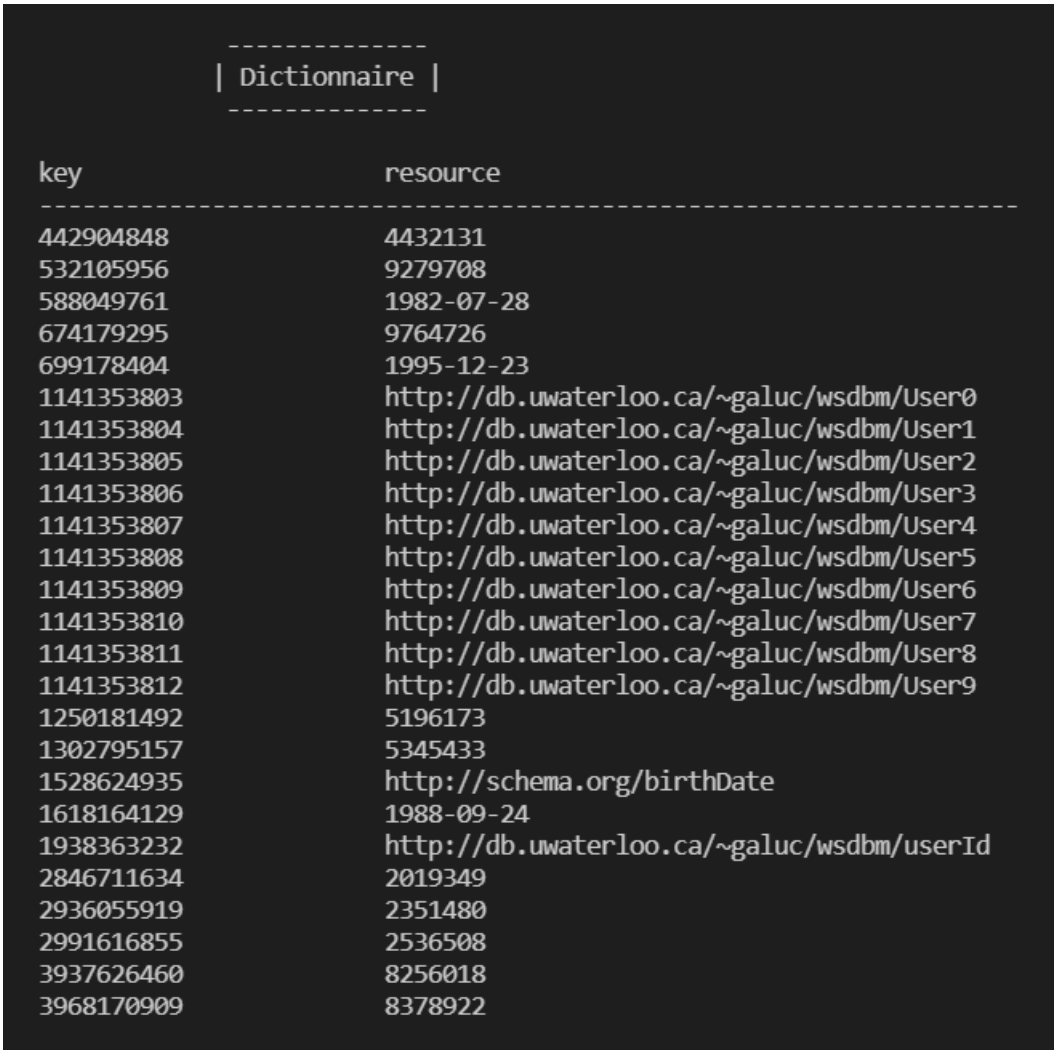
January 2022

1 Introduction

C'est en exécutant le fichier "Main.java" que vous pourrez visionner les résultats de cette première partie.
Dans le terminal, vous verrez une partie consacrée au dictionnaire, une partie consacrée aux différentes indexations, et une partie consacrée aux requêtes.

2 Dictionnaire

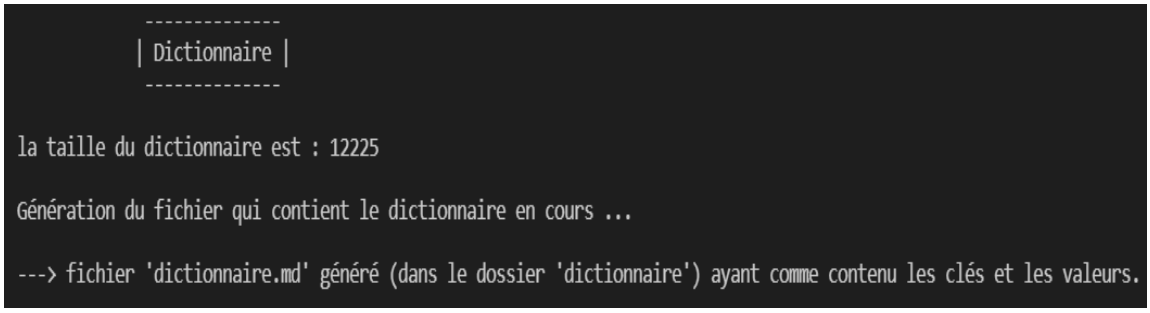
Afin de ne pas ralentir le processus d'exécution, pour le fichier "sample_data.nt" , le dictionnaire est directement affichée dans le terminal :



| Dictionnaire |

key	resource
442904848	4432131
532105956	9279708
588049761	1982-07-28
674179295	9764726
699178404	1995-12-23
1141353803	http://db.uwaterloo.ca/~galuc/wsdbm/User0
1141353804	http://db.uwaterloo.ca/~galuc/wsdbm/User1
1141353805	http://db.uwaterloo.ca/~galuc/wsdbm/User2
1141353806	http://db.uwaterloo.ca/~galuc/wsdbm/User3
1141353807	http://db.uwaterloo.ca/~galuc/wsdbm/User4
1141353808	http://db.uwaterloo.ca/~galuc/wsdbm/User5
1141353809	http://db.uwaterloo.ca/~galuc/wsdbm/User6
1141353810	http://db.uwaterloo.ca/~galuc/wsdbm/User7
1141353811	http://db.uwaterloo.ca/~galuc/wsdbm/User8
1141353812	http://db.uwaterloo.ca/~galuc/wsdbm/User9
1250181492	5196173
1302795157	5345433
1528624935	http://schema.org/birthDate
1618164129	1988-09-24
1938363232	http://db.uwaterloo.ca/~galuc/wsdbm/userId
2846711634	2019349
2936055919	2351480
2991616855	2536508
3937626460	8256018
3968170909	8378922

alors que pour le fichier "100K.nt", le résultat est écrit dans le fichier intitulé "dictionnaire.md". Dans le terminal vous verrez :



2.1 L'implémentation

Soit la variable suivante :

static BiMap $< Long, String >$ key_resource ;

Celle-ci est le dictionnaire en question qui a comme clé les hashcodes et comme valeurs les ressources.

J'utilise ces trois variables :

static List $< String >$ subjects = new ArrayList<>();

static List $< String >$ objects = new ArrayList<>();

static List $< String >$ predicates = new ArrayList<>();

afin de stocker dans chacune d'elles les sujets, objets et prédicats respectivement.

Je remplis ces trois variables dans cette méthode :

public void handleStatement(Statement st)

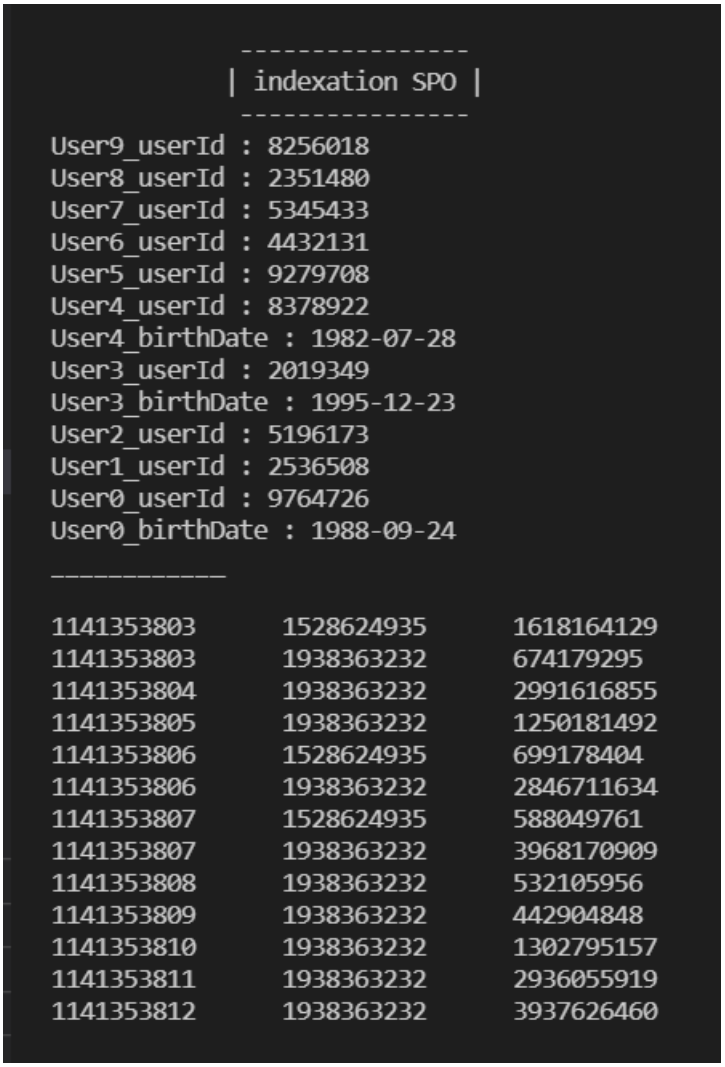
qui se trouve dans le fichier "MainRDFHandler.java". C'est grâce à ces variables que je remplis le dictionnaire.
Vous allez remarquer la présence de cette variable :

public static BiMap$< Long, String >$ resource_key;

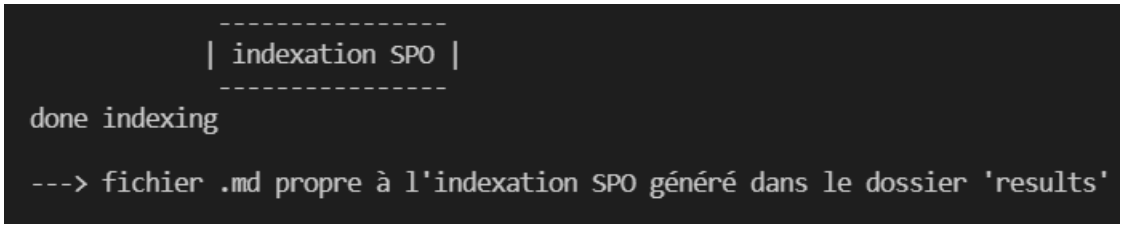
qui est utilisée afin d'avoir une recherche de complexité O(1) et ainsi éviter de parcourir tout le dictionnaire à la recherche du hashcode d'une ressource quelconque. (variable utilisée dans l'indexation)

3 Indexation

Même idée qu'avant, dans le but de ne pas ralentir l'exécution, pour le fichier "sample_data.nt" l'affichage sera dans le terminal:



alors que pour le fichier "100K.nt" le résultat sera écrit dans un fichier .md . Dans le terminal vous verrez le message suivant :



3.1 L'implémentation

Je mets toutes les méthodes qui ont un comportement commun quelque soit l'indexation dans une classe abstraite intitulée "Indexation".
La méthode :

public void fillsTriplets(List< *String* >subjects, List< *String* >objects

, List< *String* > predicates)
est abstraite dans cette classe et est implémentée dans chacune des classes filles : OPS, OSP, POS, PSO, SOP, SPO.
La variable :

static List< *List* < *String* >> triplets = new ArrayList<>();

est remplie dans chacune des classes filles ; dans la classe OPS , on trouvera que "triplets" contient (objet, prédicat, sujet) alors que dans la classe SPO on trouvera qu'elle contient (sujet, prédicat, objet).
cette variable est parcourue afin de remplir cette variable :

public static HashMap< *Long*, *HashMap* < *Long*, *List* < *Long* >>>indexations;

Le traitement de cette variable se fait dans la classe abstraite contrairement à "triplets" qui se fait dans les classes filles.
"indexations" est utilisée afin d'avoir ce genre de résultat : objet_prédicat : sujet ou bien sujet_prédicat : objet , ...
Exemple : userId_User9 : 8256018 "indextions" quant à elle ne contient que les hashcodes. On peut aisin avoit un affichage du genre :

1528624935 1141353803 1618164129

Grâce à la variable "resource_key", je peux récupérer la clé d'une ressource que je récupère de "triplets". Ainsi, j'arrive à remplir "indexations" avec les hashcodes.
Quand il y a beaucoup de données , vous allez trouver dans les .md des informations sous ce format :

userId_User9 : 8256018

4 Requêtes en étoile

Les résultats des requêtes sont écrits dans le fichier "results.md" que vous pouvez trouver dans le dossier "queries_results". Voici l'affichage que vous verrez dans le terminal (que ce soit pour le fichier "sample_data.nt" ou "100K.nt" :

```
-------------------------
Génération du fichier des résultats en cours ...

---> fichier 'results.md' généré ayant comme contenu les requetes et leur résultat.
```

4.1 L'implémentation

Les requêtes peuvent aller d'une condition à 4.
Avec la présence de prédicats et d'objets dans les conditions, les sujets sont donc les résultats des requêtes.
Je remplis une liste nommée "branches" par les prédicats et les objets correspondant à chaque requête.
Je parcours cette liste afin de remplir une seconde liste nommée "keys"

qui va contenir les hashcodes des prédicats et objets correspondants.
Par la suite , je parcours cette liste de hashcodes, en avançant par 2 à chaque fois (sachant que deux éléments successifs dans cette liste correspondent à un prédicat et un objet liés à une condition).
Donc pour chaque condition , je vais parcourir les "triplets" (rappelez vous que les triplets ne contiennent que les hashcodes) qui sont reliés biensûr à l'indexation SPO. je récupère le hashcode du sujet, prédicat et objet depuis les "triplets". Si le prédicat et objet (récupérés depuis "keys") sont égaux à ceux que je viens de récupérer, alors je récupère le sujet.
Pour les requêtes à une seule condition, le traitement dont je vais parler n'est pas aussi important. Par contre , quand on a plus d'une condition , et afin d'éviter de continuer à reparcourir le dictionnaire et l'indexation alors que la requête n'a pas de solution un certain mécanisme a été implémenté .
Je récupère tout d'abord les sujets qui sont résultats de la première condition. Je les stoque dans une liste "subjects".
Si cette liste est vide , alors je ne passe pas à la deuxième condition car cela veut dire que la première condition n'a pas de résultats. et comme le résultat de la requête c'est l'intersection des résultats de toutes les conditions, ensemble vide intersection n'importe quel autre ensemble reste vide.
Dans le cas contraire, où pour la première condition on arrive à trouver des sujets, on passe à la deuxième condition.
Pour la deuxième condition, je stoque dans une variable nommée "common_subjects" les sujets si ils sont dans "subjects" . A la fin du parcourt, "common_subjects" n'aura que les sujets qui sont résultats de la première et deuxième condition. Si "common_subjects" est vide, cela veut dire qu'il n'y aucun sujet en commun entre les conditions 1 et 2 et donc la requête n'a pas de solution (pas la peine de voir la suite) .
Dans le cas contraire, je passe à "subjects" le contenu de "common_subjects" et je vide "commo,_subjects".

Si pour la condition 3 "common_subjects" est vide , cela voudra dire que la requête n'a pas de solution.
Si pour la condition 4, "common_subjects" est vide, cela veut aussi dire que la requête n'a pas de solution, sachant que je ne suis passée à la condition 4 qu'après que "common_subjects" resultant de la condition 3 ne soit pas vide.
Ce n'est que quand "common_subjects" de la dernière condition (quand on a plus d'une condition) ou bien qu'on "subjects" n'est pas vide quand on a qu'une condition, que la requete a un résultat.