Exercice de Synthèse
« Document & Graph »

L'objectif de cet exercice est de faire une synthèse entre les datastore MongoDB et Neo4J. Il s'agit de développer une application Java qui reposera sur 2 datastores : 1 document et 1 graph.

Si nécessaire, avant de continuer sur la machine virtuelle (contenant la V3 de Neo4J) exécuter l'instruction suivante en tant que root (à ne faire qu'une fois) :

```
mv    /var/log/mongdb    /var/log/mongodb
```

1. Exécution de Neo4J & de MongDB

Pour développer cette application vous devrez utiliser le driver java pour MongoDB et Neo4J dans votre application Java.

Veuillez vous référer aux supports TPs précédents pour mettre en place l'environnement. N'oubliez pas de lancer les services nécessaires….

2. Présentation du fichier de données

Pour cette application nous allons manipuler une base de données correspondant à des données concernant des publications scientifiques. La base proposée est un sous-ensemble (999 articles) de la collection DBLP (http://dblp.uni-trier.de/).

Les nœuds disponibles sont :

- Les articles scientifiques caractérisés par :
  - Un titre
  - Une année de publication
- Les auteurs de ces articles caractérisés par :
  - Un nom
- Les journaux dans lesquels les articles sont publiés :
  - Un nom

Les relations existantes entre ces nœuds sont :

- « *Apparaitre* » traduisant le fait qu'un article apparaît dans un journal. Cette relation possède un attribut « *volume* » indiquant le volume du journal dans lequel l'article apparaît.
- « *Ecrire* » traduisant qu'un auteur a écrit un article. Cette relation ne possède pas d'attribut.

3. Mise en place des données

La base de données fournie doit être intégrée à **Neo4J** de la manière suivante :

- Supprimer la BDD précédente (**rm -rf /var/lib/neo4j/data/databases/graph.db**)
- Lancer le service neo4j (**service neo4j start**)
- Copier l'intégralité des fichiers de l'archive fournie (**fichiersPublications.tar**) sur la machine virtuelle

  Sur la machine virtuelle :
- Décompressez l'archive (**tar -xvf fichiersPublications.tar**) – cela va créer un répertoire nommé « fichiers ».
- Entrer dans ce répertoire (**cd fichiers**)
- Exécuter la procédure d'importation (**./import.sh**)

L'insertion peut prendre un peu de temps… **Patientez…..**
Si une erreur apparait, relancer le script…..

3. Développement de l'application Java

(On aurait pu traiter cette application différemment... Mais ce serait moins « marrant »…)

3.1. Mettre en place un datastore MongoDB

Sur la base des articles disponibles dans Neo4J, écrire une méthode permettant de construire une base de documents MongoDB qui va créer pour chaque article dans Neo4J un document ayant le format suivant dans une nouvelle collection nommée **`index`** :

```
{
      idDocument :  <idNoeud>,
      motsCles : [ mot1,  mot2, mot3….]
}…
```

L'*idDocument* correspond à l'id du nœud (*idNoeud*) présent dans Neo4J.
Sur MongoDB, la db sera nommée **`dbDocuments`** et la collection portera le nom **`index`**.
Les motsClés sont construits sur la base des mots du titre de l'article (en minuscules et découpage en fonction des caractères non numériques présents dans les titres : , **‘ - : ; . ( ) + [] {} ? ! (espace)**
Pour réaliser ce traitement nous vous conseillons d'utiliser un objet de type `StringTokenizer`. On fera également appel à la méthode `trim()` de `String` pour supprimer les espaces superflus.
Attention aux chaînes vides.

Rappels :
Passage en minuscules :
```
String chaineEnMinuscule = chaine.toLowerCase() ;
```
Découpage de la chaîne :
```
// Découpe en fonction des caractères , ou .
StringTokenizer st = new StringTokenizer (chaineADecouper, ",.");

while (st.hasMoreTokens()){
      // Récupération du mot suivant
      String mot = st.nextToken();
      // Traite le mot…..
}
```
Suppression des espaces superflus (en début / fin de chaîne) :
```
String nouvelleChaine = chaine.trim();
```

3.2. Mettre en place un index sur le tableau de motsClés dans MongoDB

Mettre en place un index (***ensureIndex***) sur le tableau nommé *motClés* dans MongoDB. *Remarque : lorsqu'un index est mis en place sur un champ tableau, cet index sera fait sur chaque valeur du tableau.*

```
db.index.ensureIndex({motsCles : 1})
```

Permet de mettre en place un index par ordre croissant sur les valeurs du tableau *motsCles*.

Cet index permettra de retrouver rapidement les documents possédant certains mots.

3.3. Mise en place d'une « structure miroir » sur MongoDB

Dans l'application Java, écrire un traitement/méthode correspondant à un « ForEach » permettant d'obtenir, à partir de la collection « `index` », la structure suivante dans la collection **`indexInverse`** :

```
{
      mot : <mot>,
      documents : [ idDoc1, idDoc2, … idDoc3 ]
}…
```

Où mot correspond à un mot apparaissant dans le tableau *motsCles*. Les différents *idDoc* correspondent aux identifiants des articles contenant le mot.

Remarque : attention au type de retour de « `get(motsCles)` » (BasicBSONList → ArrayList... Pensez au transtypage (cast)

Pour obtenir ce traitement, un algorithme possible peut être :

```
Pour chaque document d de la collection index faire
  Pour chaque mot m de d.motsCles faire
    Si un document dex existe déjà dans indexInverse avec le mot m alors
      On ajoute d.idDocument dans l'attribut dex.documents ;
    Sinon
      On crée un document vierge dv avec
           dv.mot = m et dv.documents = [d.idDocument] ;
      Insérer dv dans la collection indexInverse ;
    Fin si ;
  Fin pour ;
Fin pour ;
```

*Remarque : pour la recherche du document existant avec le mot **m** dans **indexInverse** on pourra utiliser, pour effectuer la recherche, la méthode **findOne()** plutôt que **find()**.*

Remarque : sauvegarder un document existant dans la collection on pourra utiliser une instruction du type :

```
database.getCollection("indexInverse").replaceOne(
                Filters.eq("_id", docExistant.get("_id")),
                docExistant
);
```

*Vérification : vous devriez obtenir 2846 documents dans **indexInverse**.*

Mettre en place un index croissant sur les mots issus de **IndexInverse**.

3.4. Recherche de documents

Implanter une méthode Java permettant de rechercher dans **indexInverse** un mot dans MongoDB et afficher par ordre alphabétique les titres des documents dans Neo4J.
A titre d'exemple, vous rechercherez le mot « with » (il y a 72 articles).

*Indication : vous pourrez construire la requête à envoyer à Neo4J sur la base des valeurs contenues dans l'attribut « **documents** » présent dans le document MongoDB (s'il existe). Se souvenir qu'il existe l'opérateur « **in** » dans Neo4J.*

3.5. Auteurs ayant écrit le plus d'articles

Implanter une méthode Java permettant d'afficher à l'écran, par ordre décroissant de nombre d'articles écrits puis par ordre croissant de nom les 10 auteurs ayant écrit le plus d'articles.

Le résultat attendu doit correspondre à :

```
7 - Joost Engelfriet
4 - Ching-Hsien Hsu
4 - Péter Kacsuk
3 - Albert Y. Zomaya
3 - Amir Pnueli
3 - Dunbing Tang
3 - Frank J. Seinstra
3 - Henri E. Bal
3 - Joanna Kolodziej
3 - Kenli Li
```

3.6. Recherche de documents avancée

Implanter une méthode Java permettant de rechercher dans **indexInverse** plusieurs mots et afficher par ordre alphabétique les titres des documents dans Neo4J. Les documents seront triés par nombre décroissant de mots de la requête contenus dans les documents.
A titre d'exemple, vous rechercherez le mot « with, systems, new ».
On se limitera aux 10 premiers documents.

Pour cette question nous vous demandons d'utiliser une agrégation lors de l'interrogation de MongoDB. Cette agrégation aura une forme proche de celle-ci :

```
db.indexInverse.aggregate( [
      {$match :
      {
            mot : {$in : ['with', 'systems']}
      }
      },
      {$sort :
      {
            mot : -1
      }
      }
] )
```

Bien entendu cette opération d'agrégation n'est pas complète. Il manque **3** étapes importantes et les valeurs doivent être adaptées au problème posé…

Résultat attendu :

```
232668 New and efficient conditional e-payment systems with transferability. 3
232402 Scheduling tasks of a parallel program in two-processor systems with use of cellular automata. 2
231035 New perspectives for the future interoperable enterprise systems. 2
232326 A performance comparison of current HPC systems: Blue Gene/Q, Cray XE6 and InfiniBand systems. 2
232222 New scoring formula to rank hypervisors' performance complementing with statistical analysis using DOE. 2
230702 A new compensation pixel circuit with all-p-type TFTs for AMOLED displays. 2
233563 Applying different control approaches for resources with high and low utilisation: a case study of the pro
230960 Visual acuity and contrast sensitivity screening with a new iPad application. 2
230467 General quantitative specification theories with modal transition systems. 2
231868 A review of Multi-Agent Systems techniques, with application to Columbus User Support Organisation. 2
```