Big Data – Gestion de données NoSQL

Master IMIS/MIAGE – INSA CVL & Univ. Orléans

Sujets de TD [1]

Radu Ciucanu

radu.ciucanu@insa-cvl.fr

Table des matières

Compte rendu de TD. A la fin des 8h de TD sur machine, il faut déposer sur Celene un seul fichier = une archive contenant vos solutions aux différents exercices :

- XPath : un fichier `.txt` avec les réponses aux questions (Exercice 1) et toutes les requêtes (Exercices 2–5)
- XQuery : un fichier `.xq` par requête
- SQL : un script `.sql`
- SPARQL : un fichier `.sparql` par requête

Ce rendu sera noté en tant que contrôle continu intégral.

Le nom de votre rendu doit contenir votre nom et prénom. Par exemple, si vous travaillez seul, votre fichier doit être nommé `NOM-Prénom`. Similairement, si vous travaillez en binôme, votre fichier doit être nommé `NOM1-Prénom1_NOM2-Prénom2`. Un seul rendu par binôme est suffisant. Des erreurs de syntaxe dans les requêtes ou dans le nommage du fichier rendu entraîneront automatiquement la perte de points.

1. Plusieurs exercices sont largement hérités de :
   - Yves Roos. Cours de *Langages Avancés pour les Bases de Données*, Université Lille 1, 2016.
   - Farouk Toumani. Cours de *Bases de Données Avancées*, Université Blaise Pascal, Clermont-Ferrand, 2014.

1 BD orientées documents

1.1 XPath

Pour exécuter une requête `Q` sur un document `document.xml`, il suffit d'exécuter la commande suivante :

```
xmllint --xpath "Q" document.xml
```

Une alternative est d'ouvrir le document en mode `shell` et ensuite d'exécuter les requêtes :

`ledit xmllint --shell document.xml` et ensuite `xpath Q`

Pour plus de détails sur la syntaxe `XPath`, n'hésitez-pas de consulter les tutoriels W3Schools :

http://www.w3schools.com/xml/xpath_intro.asp

Exercice 1. Cet exercice concerne le document `contacts.xml`. Exécutez chacune des requêtes `XPath` suivantes et à chaque fois expliquez dans une phrase le résultat retourné. Avant d'exécuter une requête, essayez de deviner ce qui va se passer.

1. `/Contacts`
2. `/Contacts/Person`
3. `//Person[Firstname="John"]`
4. `//Person[Email]`
5. `/Contacts/Person[1]/Firstname/child::text()`
6. `/Contacts/Person[1]/Firstname/text()`
   Comparez avec le résultat de la requête précédente.
7. `/Contacts//Address[@type="home"]//Street/child::text()`
8. `/Contacts//Address[@type="home" and City="London"]`
9. `/Contacts//Address[@type="work" and City="Dublin"]/parent::node()/Lastname/text()`
10. `/Contacts//Address[@type="work" and City="Dublin"]/../Lastname/text()`
    Comparez avec le résultat de la requête précédente.
11. `/Contacts[.//Address[@type="work" and City="Dublin"]]//Lastname/text()`
    Comparez avec le résultat de la requête précédente.
12. `/Contacts//Address[@type="work"]/ancestor::node()`
13. `/Contacts/Person[Lastname="Smith"]/following-sibling::node()/Lastname/text()`
14. `/Contacts/Person[following-sibling::node()/Lastname="Dunne"]/Lastname/text()`

Exercice 2. On considère des documents `XML` correspondant à la description d'une collection de CD audio. Le fichier `cd.xml` donne un exemple de document contenant une seule entrée (un seul CD). Une collection est un document valide vis-à-vis du DTD `cd.dtd` :

```
<!ELEMENT CDlist      (CD+)>
<!ELEMENT CD          (composer, performance+, publisher, length?)>
<!ELEMENT performance (composition, soloist?, (orchestra, conductor)?)>
<!ELEMENT composer    (#PCDATA)>
<!ELEMENT publisher   (#PCDATA)>
<!ELEMENT length      (#PCDATA)>
<!ELEMENT composition (#PCDATA)>
<!ELEMENT soloist     (#PCDATA)>
<!ELEMENT orchestra   (#PCDATA)>
<!ELEMENT conductor   (#PCDATA)>
```

Trouvez les requêtes `XPath` qui retournent les informations suivantes.

1. Toutes les compositions.
2. Toutes les compositions ayant un `soloist`.
3. Toutes les performances avec un seul `orchestra` mais pas de `soloist`.
4. Tous les soloists ayant joué avec le `London Symphony Orchestra` sur un CD publié par `Deutsche Grammophon`.
5. Tous les CDs comportant des performances du `London Symphony Orchestra`.

Exercice 3. Le fichier `booker.xml` contient une liste de livres (les gagnants du Booker Prize) avec leur auteur et l'année de l'obtention du prix. Trouvez les requêtes `XPath` qui retournent les informations suivantes.

1. Le titre du cinquième livre dans la liste.
2. L'auteur du sixième livre dans la liste.
3. Le titre du livre qui a gagné en `2000`.
4. Le nom de l'auteur du livre intitulé `Possession`.
5. Le titre des livres dont `J M Coetzee` est l'auteur.
6. Le nom de tous les auteurs qui ont obtenu un prix depuis `1995`.
7. Le nombre total de prix décernés.

Exercice 4. Quelques recettes ont été extraites de différents livres. A partir de celles-ci, on a conçu deux `DTD` différents permettant de décrire ces recettes de cuisine. Les `DTD` sont disponibles dans les fichiers `recettes1.dtd` et `recettes2.dtd`. Les documents correspondants sont disponibles dans les fichiers `recettes1.xml` et `recettes2.xml`. D'abord, visualisez les `DTD` et les documents correspondants pour en comprendre la structure.

Pour chacun des deux documents, donnez les requêtes `XPath` permettant d'obtenir :

1. Les éléments titres des recettes.
2. Les noms des ingrédients.
3. L'élément titre de la deuxième recette.
4. La dernière étape de chaque recette.
5. Le nombre de recettes.
6. Les éléments recette qui ont strictement moins de 7 ingrédients.
7. Les titres des recettes qui ont strictement moins de 7 ingrédients.
8. Les recettes qui utilisent de la farine.
9. Les recettes de la catégorie entrée.

Exercice 5. Le fichier `iTunes-Music-Library.xml` contient une bibliothèque musicale au format de sauvegarde prévu par le logiciel `iTunes`. Ce format est un peu particulier, comme vous pouvez le constater en consultant le `DTD` qui lui est associé. La structure est finalement assez pauvre, chaque propriété étant définie par un couple clé (élément `key`), valeur (élément `string` ou `integer`). Il est quand même assez facile de comprendre à quoi correspond chacune des propriétés ; il est donc possible d'exploiter ce fichier et d'extraire des informations avec des requêtes `XPath`.

Donnez les requêtes `XPath` permettant d'obtenir :

1. Le nombre de morceaux (*tracks* hors *PlayLists*) de la bibliothèque.
2. Tous les noms d'albums.
3. Tous les genres de musique (*Jazz*, *Rock*, ...)
4. Le nombre de morceaux de Jazz.
5. Tous les genres de musique mais en faisant en sorte de n'avoir dans le résultat qu'une seule occurrence de chaque genre.
6. Le titre (*Name*) des morceaux qui ont été écoutés au moins 1 fois.
7. Le titre des morceaux qui n'ont jamais été écoutés.
8. Le titre du (ou des) morceaux les plus anciens (renseignement *Year*) de la bibliothèque.

1.2 XQuery

Pour exécuter une requête `query.xq` afin de produire un fichier résultat `output.xml`, il suffit d'exécuter la commande suivante :

```
xqilla -o output.xml query.xq
```

Pour plus de détails sur la syntaxe `XQuery`, n'hésitez-pas de consulter les tutoriels W3Schools :

http://www.w3schools.com/xml/xquery_intro.asp

Exercice 1. Ecrire un programme `XQuery` qui, à partir du fichier `maisons.xml`, calcule, pour chaque maison, sa superficie totale. La sortie du programme sera un fichier `HTML` dont la visualisation correspond à la capture écran suivante :

Maisons	Surfaces (m2)
Maison 1	125
Maison 2	28
Maison 3	57.5

Exercice 2. On considère les trois fichiers `xml` suivants :

- `plant_catalog.xml` est un catalogue de plantes ;
- `plant_families.xml` qui indique à quelle famille appartiennent certaines plantes ;
- `plant_order.xml` est une commande de plantes.

1. Donnez un programme `XQuery` qui produit à partir des fichiers `plant_catalog.xml` et `plant_families.xml` un document `XML` en ajoutant dans chaque élément `PLANT` apparaissant dans `plant_catalog.xml` un élément `FAMILY` qui donne le nom de la famille à laquelle appartient la plante comme dans l'exemple ci-dessous :

```
<PLANT>
      <COMMON>Bloodroot</COMMON>
      <BOTANICAL>Sanguinaria canadensis</BOTANICAL>
      <ZONE>4</ZONE>
      <LIGHT>Mostly Shady</LIGHT>
      <PRICE>$2.44</PRICE>
      <AVAILABILITY>031599</AVAILABILITY>
      <FAMILY>Papaveraceae</FAMILY>
</PLANT>
```

2. Donnez un programme `XQuery` qui regroupe les éléments `PLANT` du fichier `plant_catalog.xml` en fonction du contenu de leur élément `LIGHT`. Vous devez obtenir un document identique au fichier `exposure.xml` disponible dans l'archive.
3. Donnez un programme `XQuery` qui réalise les 2 opérations des questions précédentes en classant en outre les éléments `LIGHT` par ordre alphabétique du contenu des éléments `EXPOSURE` et en classant les éléments `PLANT` par ordre alphabétique du contenu des éléments `COMMON`.
4. Donnez un programme `XQuery` qui calcule le montant total de la commande décrite dans `plant_order.xml` en donnant le résultat dans un élément `PRICE`. On doit obtenir :

```
<PRICE>663.2</PRICE>
```

2 BD orientées graphes

2.1 SQL

Malgré l'existence d'une multitude de systèmes spécialisés pour la gestion de données orientées graphes, beaucoup de requêtes que l'on veut spécifier sur les graphes sont exprimables en SQL. Dans ce cas-là, on peut utiliser pour simplicité un SGBD relationnel plutôt qu'un système NoSQL spécialisé.

Dans cet exercice, nous utilisons SQLite [2]. Pour le démarrer, il faut simplement taper `sqlite3` dans un terminal. Voici une liste de commandes utiles, qui concernent le terminal dans lequel `sqlite3` est ouvert :

- `.read fichier.sql` – pour exécuter un fichier `fichier.sql` qui contient des commandes SQL
- `.separator ";"` – pour changer de séparateur (vous pouvez remplacer ";" par le séparateur de votre choix)
- `.import f R` – pour importer dans la relation `R` des données csv stockées dans le fichier `f`
- `.help` – pour voir la liste des commandes SQLite

Nous utilisons un jeu de données réel issu de Twitter [3]. Il s'agit d'une relation binaire `friend/follower`, qui contient environ 15 millions de tuples, donnés dans `higgs-social_network.edgelist`. Cela encode un graphe orienté, où les noeuds sont des utilisateurs. Un arc entre deux utilisateurs encode le fait que le premier a comme follower sur Twitter le deuxième.

Créez un script SQL qui importe le jeu de données et calcule des statistiques simples. Votre script retournera :

```
nb total de relations friend/follower :  14855842
nb utilisateurs qui ont au moins un follower :  456626
nb utilisateurs qui suivent au moins qqn :  370341
nb max de followers per utilisateur :  1259
-- exemple utilisateur avec nb max de followers :  13115
nb min de followers per utilisateur :  1
-- exemple utilisateur avec min de followers :  17
```

2.2 SPARQL

Nous utilisons Apache Jena [4] pour écrire des requêtes SPARQL. Apache Jena est écrit en Java et a besoin de Java pour fonctionner [5].

Désarchivez `apache-jena-3.1.1.tar.gz` et placez-vous dans le sous-répertoire `bin`. Pour exécuter une requête `Q` sur un graphe `G`, il suffit de faire :

```
./sparql --data G --query Q
```

Attention à utiliser les chemins corrects vers les deux fichiers. Pour plus de détails sur les requêtes SPARQL sous Apache Jena, vous pouvez consulter https://jena.apache.org/tutorials/sparql.html.

Considérons un graphe RDF qui contient des informations sur des personnes. Le graphe est donné sous format Turtle dans le fichier `human.ttl`.

1. (a) Expliquez le résultat de la requête suivante.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT (COUNT(*) as ?c)  WHERE
{
  ?x rdf:type ?t
}
```

   (b) Donnez une requête qui retourne le type de John.

2. Expliquez le résultat de la requête suivante.

2. https://www.sqlite.org/index.html
3. https://snap.stanford.edu/data/higgs-twitter.html
4. https://jena.apache.org/
5. `sudo apt-get install openjdk-8-jre-headless`

```
PREFIX humans: <http://www.inria.fr/2007/09/11/humans.rdfs#>
SELECT * WHERE {
  ?x humans:hasSpouse ?y
}
```

3. Donnez une requête qui retourne toutes les personnes qui ont plus de 20 ans. La fonction `xsd:integer(param)` convertit son paramètre `param` de string en entier.
4. (a) Donnez une requête qui extrait toutes les personnes (`Person`) avec leur pointure.
   (b) Modifiez cette requête pour extraire toutes les personnes et, si elle est disponible, leur pointure. Hint : utilisez `OPTIONAL`.
   (c) Modifiez cette requête pour extraire toutes les personnes et, si elle est disponible, leur pointure à condition que celle-ci soit supérieure à 8.
   (d) Écrire une requête pour extraire toutes les personnes dont la pointure est supérieure à 8 ou dont la taille de chemise est supérieure à 12.
5. (a) Formulez une requête pour trouver toutes les propriétés de John.
   (b) Demandez une description de John en utilisant la clause SPARQL prévue pour cela (i.e., `DESCRIBE`).
6. (a) Donnez tous les couples (parent, enfant).
   (b) Formulez une requête pour trouver les personnes qui ont au moins un enfant.
   (c) Modifiez cette requête pour éliminer les doublons.
   (d) Donnez une requête pour trouver les hommes (`Man`) qui n'ont pas d'enfant.
   (e) Donnez une requête pour trouver les femmes (`Woman`) mariées, avec éventuellement leurs enfants.
7. Donnez une requête qui retourne les paires de personnes différentes qui ont la même taille de chemise.
8. (a) Construisez la clôture symétrique de la relation `hasFriend`.
   (b) Construisez la clôture transitive de la clôture symétrique de la relation `hasFriend`.
9. Recherchez toutes les personnes qui ne sont pas des hommes (`Man`).