Traitement des données NoSQL

INSA Centre Val de Loire, STI8 – EA Big Data, 2018–2019

Sujets de TD [1]

Radu Ciucanu

radu.ciucanu@insa-cvl.fr

Table des matières

Sujet 1 – XML, DTD, XML Schema **2**

Sujet 2 – XPath **4**

Sujet 3 – Bases de données orientées graphes **6**

Sujet 4 – MapReduce **8**

Rendus sur Celene suite aux TD sur machine

La qualité de votre travail pendant les TD sur machine compte pour 40% de la note finale. Après chacun des 4 sujets, vous ferez un rendu sur Celene.

Le nom de votre rendu doit contenir votre nom et prénom. Par exemple, si vous travaillez seul, votre fichier du premier rendu doit être nommé `sujet1_NOM-Prénom`. Similairement, si vous travaillez en binôme, votre fichier doit être nommé `sujet1_NOM1-Prénom1_NOM2-Prénom2`. Un seul rendu par binôme est suffisant. Des erreurs de syntaxe ou dans le nommage du fichier rendu entraîneront automatiquement la perte de points.

– Pour les sujets 1 (XML, DTD, XML Schema), 3 (BD graphes) et 4 (MapReduce), le rendu sera une archive contenant vos solutions aux exercices, c'est-à-dire les différents fichiers que vous avez créés ou modifiés.

– Pour le sujet 2 (XPath), vous pouvez rendre simplement un fichier txt qui contient toutes vos réponses.

1. Plusieurs exercices dans les trois premiers sujets 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.

Sujet 1 – XML, DTD, XML Schema

Pour lire ou modifier le contenu d'un fichier XML, il suffit de l'ouvrir avec un éditeur de texte (e.g., `gedit`). On peut également visualizer un fichier XML dans un navigateur ; si le fichier est bien formé on voit l'arbre du document, sinon on a une erreur. Dans le navigateur il est possible de voir le contenu du fichier initial en demandant à voir le code source de la page.

On va utiliser `xmllint` (disponible sous Linux) pour tester la validité d'un document XML. Par exemple :

— Pour valider un document contre un DTD (qui lui est déjà associé), il suffit de faire

```
xmllint --valid fichier.xml --noout
```

— Pour valider un document contre un XML Schema, il faut explicitement donner le schéma

```
xmllint --schema fichier_schema.xsd fichier.xml --noout
```

Dans les deux cas, l'option `--noout` empêche d'afficher le document XML. Plus d'infos sur l'outil `xmllint` sont disponibles via la commande "`man xmllint`".

Pour plus de détails de syntaxe DTD et XML Schema, n'hésitez-pas de consulter les tutoriels W3Schools :

— http://www.w3schools.com/xml/xml_dtd_intro.asp

— http://www.w3schools.com/xml/schema_intro.asp

Exercice 1

La France, l'Italie et l'Espagne ont pour capitales respectives Paris, Rome et Madrid. Leurs populations respectives sont de 64 102 000 habitants (en 2007), de 58 133 509 et 44 708 964 habitants (en 2006). Structurez ces informations sous la forme d'un document XML bien formé.

Exercice 2

1. Validez le fichier `cd-1.xml` contre le DTD `cd.dtd`.
2. Essayez de valider le fichier `cd-2.xml` contre le même DTD `cd.dtd` et vous allez constater des erreurs. Faites un nombre minimal de changements sur le fichier `cd-2.xml` pour le rendre valide.
3. Validez le fichier `note-1.xml` contre le XML Schema `note.xsd`.
4. Essayez de valider le fichier `note-2.xml` contre le même XML Schema `note.xsd` et vous allez constater des erreurs. Faites un nombre minimal de changements sur le fichier `note-2.xml` pour le rendre valide.

Exercice 3

On désire écrire un `DTD` pour des documents décrivant des familles. Une famille porte un nom et est constituée d'une ou plusieurs personnes. Pour chaque personne de la famille, on a le prénom, l'âge, le poids en kilos (kg) ou le poids en livres (lb), et éventuellement la taille. Les liens de parenté (père et mère) sont gérés grâce à des attributs de type `ID` et `IDREF`.

Écrire le `DTD` correspondant `famille.dtd`. Celle-ci devra accepter par exemple le document suivant qui est disponible dans l'archive sous le nom `famille.xml`. L'attribut `pnumber` est obligatoire.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE famille SYSTEM "famille.dtd">
<famille>
        <nom>Martin</nom>
        <personne pnumber="a1">
                <prenom>Juliette</prenom>
                <age>30</age>
                <poids-kg>58</poids-kg>
        </personne>
        <personne pnumber="a2">
                <prenom>Romeo</prenom>
                <age>31</age>
                <poids-lb>97</poids-lb>
        </personne>
        <personne pnumber="a3" mere="a1" pere="a2">
                <prenom>Max</prenom>
                <age>4</age>
                <poids-kg>12</poids-kg>
                <taille>1.25</taille>
        </personne>
        <personne pnumber="a4" mere="a1" pere="a2">
                <prenom>Marie</prenom>
                <age>3</age>
                <poids-lb>18</poids-lb>
                <taille>1.10</taille>
        </personne>
</famille>
```

Exercice 4

En plein championnat de football, on veut mémoriser les scores et les dates du championnat dans un fichier `XML`.

- Dans le championnat de France, il y a 20 clubs. On mémorisera la liste des clubs, indépendamment du calendrier du championnat. Chaque club possède un nom long (comme `"Olympique de Marseille"`, un nom court (comme `"OM"`).
- Le championnat est divisé en journées, et chaque journée comporte 10 rencontres. On mémorisera la date de chaque journée, mais pas la date de chaque match (parfois un match se joue en avance ou en retard par rapport à la date officielle). Il y a au total 38 journées.
- Les clubs se rencontrent tous 2 fois, une fois à domicile et une fois à l'extérieur. Les 19 premières journées, chaque club rencontre tous les autres (on parle de matches aller). Les 19 journées suivantes, chaque club rencontre aussi tous les autres mais pas sur le même terrain (on parle de matches retour). Une rencontre est donc caractérisée par un couple de (noms courts de) clubs. On mémorisera tous les scores des matches du championnat.

Définissez un `XML Schema` qui permet de représenter le championnat de France de football. La qualité d'un schéma est aussi liée à sa lisibilité. Évitez les imbrications trop profondes (schéma de type *poupées russes*). Un fichier `championnat.xml` est donné à titre d'exemple. Vous pouvez modifier ce fichier pour l'adapter à votre schéma et pour tester celui-ci.

Sujet 2 – 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.

Sujet 3 – Bases de données orientées graphes

Exercice 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
```

Exercice 2 : SPARQL

Nous utilisons Apache Jena [4] pour écrire des requêtes SPARQL. Apache Jena (tout comme Hadoop dans le sujet suivant) 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`).

Sujet 4 – MapReduce

Pour mettre en oeuvre des programmes MapReduce, nous utilisons l'implémentation open-source Hadoop [6]. Hadoop est écrit en Java, qui est d'ailleurs son langage par défaut [7]. Pour des raisons de simplicité, nous utilisons une astuce pour écrire le code avec le langage de programmation de notre choix : Python [8], moins lourd que Java.

C'est pour cela que nous utilisons Hadoop en mode streaming [9], c'est-à-dire nous spécifions les fonctions `map` et `reduce` en tant qu'exécutables qui lisent l'entrée à partir du `stdin` et émettent la sortie sur `stdout`.

— Configurez la variable `JAVA_HOME` en fonction de votre installation Java, par exemple
`export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64`

— Téléchargez Hadoop
https://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-2.9.1/hadoop-2.9.1.tar.gz.
Un lien vers une copie locale est aussi disponible la page Celene du cours.

— Désarchivez `hadoop-2.9.1.tar.gz` et placez-vous dans le répertoire résultat `hadoop-2.9.1`. La commande suivante permet de lancer Hadoop en mode streaming avec les programmes Python `map.py` et `reduce.py`, avec les données d'entrée et de sortie dans les répertoires `INPUT_DIR` et respectivement `OUTPUT_DIR` :

```
bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.9.1.jar \
  -input INPUT_DIR \
  -output OUTPUT_DIR \
  -mapper map.py \
  -reducer reduce.py
```

Exercice 1 : Word count

Pour tester la commande ci-dessus, utilisez le programme `identity.py` en tant que map et reduce, et le répertoire `input-word-count` en tant que répertoire d'entrée. Le programme `identity.py` fait simplement l'application identité, c'est-à-dire il copie l'entrée sur la sortie.

Créez les programmes `word-count-map.py` et `word-count-reduce.py` qui comptent les mots d'un texte en utilisant le paradigme MapReduce. Sur les données d'entrée du répertoire `input-word-count`, le résultat attendu est celui du fichier `expected-outputs/word-count.txt`.

En particulier, vous devez aussi transformer toutes les lettres en minuscules (`The` et `the` sont un même mot) et éliminer les signes de ponctuation (`dog` et `dog.` sont un même mot).

Exercice 2 : Produit matriciel

Implémentez le produit matriciel avec MapReduce, en deux versions :

1. Avec un seul round MapReduce
2. Avec deux rounds MapReduce

Comme format d'entrée pour les deux matrices, vous pouvez utiliser les exemples donnés dans `input-matmul`. Pour ces exemples de matrices, le résultat attendu est celui du fichier `expected-outputs/matmul.txt`. Bien évidemment, le même résultat est attendu pour les deux algorithmes de produit matriciel.

Exercice 3 : Agrégats sur des données Twitter

Nous utilisons le même jeu de données réel issu de Twitter comme dans le sujet précédent. Pour rappel, il s'agit d'une relation binaire `friend/follower`, qui contient environ 15 millions de tuples.

1. Pour chaque utilisateur (colonne `friend` dans les données d'entrée), comptez le nombre de followers. Le résultat attendu est celui du fichier `expected-outputs/aggreg-count.txt`.
2. En utilisant en entrée le résultat de la question précédente, calculez le nombre total d'utilisateurs, le nombre total de relations friend/follower, les valeurs minimale et maximale de nombre de followers pour un utilisateur, et au moins un utilisateur pour lequel chacune de ces valeurs extrêmes est atteinte. Le résultat attendu est celui du fichier `expected-outputs/aggreg-other.txt`.

6. https://hadoop.apache.org/docs/current/
7. http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
8. https://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/
9. https://hadoop.apache.org/docs/r2.9.1/hadoop-streaming/HadoopStreaming.html