Les bases de données NoSQL basées sur XML

L. KERKENI

IUT de Laval - Dpt. MMI

2019-2020

Plan

Introduction

Pourquoi utiliser une base de données XML plutôt que relationnelle ?

1. XML et ses dérivés deviennent des standards dans l'échange d'information.
2. Internet devient une base de données à lui seul.
3. XML comporte l'information et le modèle d'information dans la même syntaxe.
4. Les types de données sont extensibles.
5. Les schémas sont flexibles, évolutifs, mais aussi optionnels.
6. Un fichier XML peut contenir des éléments vides ou absents.
7. En base de données relationnelle, la structure est figée, uniforme et répétitive.
8. En XML, chaque page est différente.
9. En XML, la recherche peut s'effectuer sans connaître la structure.

Introduction

Pourquoi les requêtes XML sont différentes de celles en SQL ?

1. Le résultat d'une requête XML peut être de type différent et de structure complexe. Par exemple : *//*[couleur="rouge"]* peut retourner une cerise, une voiture, ...
2. On manipule deux types d'objet et parfois en même temps : éléments et valeurs atomiques.
3. Une requête XML peut effectuer des transformations structurelles. Par exemple : modifier l'ordre d'une hiérarchie d'éléments.
4. L'ordre en XML est important. On peut demander le 3ème fils.

Un langage de requêtes pour XML

- XPath : Langage de recherche au sens extraction d'une valeur dans un fichier XML. Mais il est insuffisant pour en faire un véritable langage de requêtes (par exemple, jointures impossibles).
- XSLT : Permet la construction et le filtrage, mais n'est pas un langage de requête, c'est un langage de transformation.
- XQuery : Recommandation du W3C depuis janvier 2007.

Plan

Les expressions XQuery
Règles XQuery

- XQuery comme tout langage XML est sensible à la casse.
- Une expression XQuery a une valeur et ne génère pas d'effet de bord.
- Les expressions peuvent se composer.
- Une expression peut générer une erreur.
- Des commentaires peuvent être insérés :

```
(: exemple de commentaire :)
```

Une requête XQuery

Une requête est une expression qui :

- Lit une séquence de fragments XML ou de valeurs atomiques.
- Retourne une séquence de fragments XML ou de valeurs atomiques.

Une expression XQuery peut prendre les formes suivantes :

- Expression de chemin (XPath)
- Constructeurs d'objets
- Expression FLWOR
- Expression de listes
- Condition
- Expression quantifiée
- Expression de type de données
- Fonctions

La structure d'une requête

Une requête est composée :

- D'un prologue contenant :
  - Les déclarations d'espaces de nommage.
  - Les importations de schémas.
  - Les importations de modules.
  - Les définitions de fonctions.
  - Les déclarations de variables globales et externes.
  - Le contrôle pour la gestion des espaces.
- D'un corps contenant :
  - Une expression qui définit le résultat.

La syntaxe “Flower”

Il s’agit des différentes clauses disponibles pour faire des requêtes (FLWOR) :

- For : Association de variables et d’éléments (avec itération)
- Let : Assignation de valeur(s) à une variable (sans itération)
- Where : Spécification de critères sur le résultat
- Order by : Tri du résultat
- Return : Spécification du résultat

La syntaxe “Flower” (2)

```
for variable in expression (: Au moins un FOR ou un LET :)
let variable := expression (: Possibilité de plusieurs FOR et LET :)
where expression (: Optionnelle :)
order by expression (: Optionnelle :)
return expression (: Obligatoire :)
```

FOR

La clause FOR fonctionne comme en algorithmique et propose un traitement pour chaque valeur obtenue.

Exemple

```
for $e in doc("apogee.xml")//ETUDIANT
```

Permet d'effectuer un traitement, une requête pour chaque étudiant du fichier *apogee.xml*.

LET

La clause LET fonctionne différemment et renseigne une variable avec l'ensemble des réponses possibles.

Exemple

```
let $e := doc("apogee.xml")//ETUDIANT
```

Permet d'effectuer un traitement, une requête sur la liste des étudiants du fichier *apogee.xml*.

Combinaison FOR et LET

Les clauses sont combinables, notamment lorsque plusieurs fichiers XML sont utilisés.

Exemple

```
for $f in doc("universite.xml")//FORMATION
let $e := doc("apogee.xml")//ETUDIANT[@formation = $f/@id]
```

Permet d'obtenir la liste des formations de l'Université avec la liste des étudiants pour chacune d'elle.

RETURN

La clause RETURN formate le résultat de la requête pour chaque itération.

Exemple 1

```
<ETUDIANTS>{
for $e in doc("apogee.xml")//ETUDIANT
return $e/NOM}
</ETUDIANTS>
```

Exemple 2

```
let $e := doc("apogee.xml")//ETUDIANT/NOM
return <NOMS>{$e}</NOMS>
```

Attention

Il faut toujours veiller à ce qu'une requête retourne un élément XML et non une collection.

WHERE

La clause WHERE sélectionne les valeurs (comme en SQL).

Exemple

```
<ETUDIANTS>{
for $e in doc("apogee.xml")//ETUDIANT
where $e/CODE_POSTAL = "72000"
return $e/NOM}
</ETUDIANTS>
```

Remarque

Il est possible d'utiliser les opérateurs logiques classiques.

ORDER BY

La clause ORDER BY permet de trier le résultat de la requête.

Exemple

```
<ETUDIANTS>{
for $e in doc("apogee.xml")//ETUDIANT
order by $e/NOM
return $e/NOM}
</ETUDIANTS>
```

Remarque

Il est possible d'inverser l'ordre en indiquant *descending* juste après le critère de tri.

Les expressions conditionnelles

Il est possible de formater le résultat de la requête selon différentes valeurs. Pour cela, il existe le classique “Si . . .Alors . . .Sinon”.

Exemple

```
<ETUDIANTS>{
for $e in doc("apogee.xml")//ETUDIANT
return
if ($e/../@nom = "DUT SRC 1")
then <PREMIERE_ANNEE> {$e} </PREMIERE_ANNEE>
else <SECONDE_ANNEE> {$e} </SECONDE_ANNEE>}
</ETUDIANTS>
```

Les expressions quantifiées

Quantificateur SOME

```
<DOCUMENTS>{
for $a in doc("srcnews.xml")//ARTICLE
where some $p in $a/PARAGRAPHE satisfies
(contains($p,"Laval") and contains($p,"XML"))
return $a/TITRE}
</DOCUMENTS>
```

Traduction

La requête retourne les titres des articles dont au moins un paragraphe comporte les mots “Laval” et “XML”.

Les expressions quantifiées (2)

Quantificateur EVERY

```
<DOCUMENTS>{
for $a in doc("srcnews.xml")//ARTICLE
where every $p in $a/PARAGRAPHE satisfies
(contains($p,"Laval") and contains($p,"XML"))
return $a/TITRE}
</DOCUMENTS>
```

Traduction

La requête retourne les titres des articles dont tous les paragraphes comportent les mots “Laval” et “XML”.

Les requêtes imbriquées

Il est possible d'imbriquer les requêtes à l'intérieur d'une clause RETURN :

Exemple

```
<HOTELS>{
for $v in
distinct-values(doc("guideroutard.xml")//HOTEL//VILLE)
return
   <VILLE>
    <NOM>{$v}</NOM>
    <HOTEL>
     for $h in doc("guideroutard.xml")//HOTEL
     where $h/VILLE = $v
     return {$h}
    </HOTEL>
   </VILLE>}
</HOTELS>
```

Les requêtes imbriquées (2)

Il est possible d'imbriquer les requêtes à l'intérieur d'une clause WHERE :

Exemple

```
<HOTELS>{
for $h in doc("guideroutard.xml")//HOTEL
where $h/VILLE IN
   for $r in doc("guideroutard.xml")//RESTAURANT
   where $r/@type = "asiatique"
   return {$r/VILLE/text()}
return $h}
</HOTELS>
```

Les informations du prologue

La déclaration de version

```
xquery version "1.0" encoding "iso-8859-15";
```

La déclaration de module

```
module namespace math="http//example.org/math-functions";
```

Le contrôle des espaces

```
declare boundary-space preserve;
declare boundary-space strip;
```

L'url de référence

```
declare base-uri "http//example.org";
```

Les informations du prologue (2)

L'importation de schéma

```
import schema namespace soap=
"http//www.w3.org/2003/05/soap-envelope" at
"http//www.w3.org/2003/05/soap-envelope/";
```

L'importation de module

```
import module namespace math=
"http//example.org/math-functions";
```

La déclaration d'espaces de nommage

```
declare namespace fn=
"http//www.w3.org/2005/xpath-functions";
```

La déclaration de variables

```
declare variable $x as xsinteger := 7;
```

Plan

Les fonctions internes

XQuery possède déjà un certain nombre de fonctions “classiques” :

- Agrégation : count(), avg(), max(), min(), sum()
- Contexte : last(), position(), ...
- Numériques : abs(), round(), ...
- Manipulation de chaînes : substring(), contains(), string-length(), ...
- Autres : exists(), distinct-values(), reverse(), index-of(), ...

Exemple

```
<ETUDIANTS>{
for $c in
distinct-values(doc("apogee.xml")//ETUDIANT/CODE_POSTAL)
let $e := doc("apogee.xml")//ETUDIANT[CODE_POSTAL=$c]
return <VILLE><CP>{$c}</CP>{$e/NOM}</VILLE>}
</ETUDIANTS>
```

Définir de nouvelles fonctions

XQuery permet de créer ses propres fonctions et de les regrouper dans des modules. Les fonctions sont basées sur les types XPath pour les arguments et les types XML Schéma pour les valeurs de retour.

Exemple

```
declare function depth($n as node()) as xsdinteger
{
if (empty($n/*)) then 1
else max(for $c in $n/* return depth($c)) + 1
};
```

Attention

Les fonctions ne peuvent pas être surchargées.

Plan

Insertion

- Expression : `update insert expr ( into | following | preceding ) exprSingle`
- Insère le contenu spécifié dans `expr` au sein du nœud indiqué par `exprSingle`. `exprSingle` et `expr` correspondent à des ensembles de nœuds. Si `exprSingle` contient plus d'un nœud élément, la modification sera appliquée sur chacun des nœuds.
- Options :
  - into : Le contenu est ajouté après le dernier nœud spécifié.
  - following : Le contenu est ajouté immédiatement après le nœud spécifié dans `exprSingle`.
  - preceding : Le contenu est ajouté avant le nœud spécifié dans `exprSingle`.
- Exemple : `update insert <email type="office"> andrew@gmail.com </email> into //address[fname="Andrew"]`

Mise à jour

- Replace
  - Expression : `update replace expr with exprSingle`
  - Remplace les nœuds retournés par `expr` avec les nœuds de `exprSingle`.
  - Exemple : `update replace //fname[. = "Andrew"] with <fname>Andy</fname>`
- Value
  - Expression : `update value expr with exprSingle`
  - Remplace le contenu de tous les nœuds de `expr` avec les valeurs de `exprSingle`.

Suppression et renommage

- Supression
  - Expression : `update delete expr`
  - Supprime tous les nœuds de l'expression `expr` du document.
  - Exemple : `for $city in //address/city return update delete $city`
- Renommage
  - Expression : `update rename expr as exprSingle`
  - renomme les nœuds de `expr` en utilisant la valeur de type String du premier item de `exprSingle` comme nouveau nom du nœud.

Plan

La librairie SAXON

Cette librairie JAVA permet de tester ses requêtes et d'interroger un ou plusieurs fichiers.

Le fichier XQuery

```
xquery version "1.0";
<LISTE> {for $e in doc("apogee.xml")//ETUDIANT
return $e}
</LISTE>
```

L'exécution

`java -cp saxon9.jar net.sf.saxon.Query` *monfichier.xq*

Référence

`http://saxon.sourceforge.net/`

Une base de données : eXist

- C'est une base de données native XML et open source qui peut gérer jusqu'à 2^{31} documents.
- Elle possède un processus d'indexation pour améliorer ses capacités de recherche.
- Elle implémente XQuery avec des extensions pour la recherche en texte intégral.
- Elle implémente XUpdate, l'équivalent de XQuery pour les mises à jour.
- Elle est basée sur Java donc multi-plateformes et possède une communauté d'utilisateurs avec liste de diffusion.
- Elle possède un serveur et un client permettant l'administration de la base.
- Elle est conçue pour le développement d'applications Web (Protocole XML-RPC).
- Elle utilise le protocole HTTP pour dialoguer donc elle est compatible avec de nombreux langages (Python, Java, Php, .. .).

Une base de données : eXist (2)

Les particularités :

- eXist gère des collections.
- Dans une collection, il y a des documents XML, XML Schéma.
- Les droits des utilisateurs sont fonction des collections.
- Il est possible d'appeler des fonctions Java dans des requêtes XQuery (désactivé par défaut).
- document() permet de sélectionner plusieurs documents, car il peut prendre plusieurs chemins en argument. Sans arguments, il prend tous les documents de la base.
- collection() permet de sélectionner une collection et toutes ses sous-collections. xcollection() permet de ne prendre en compte que la collection.

Une base de données : eXist (3)

document()

```
document()//ETUDIANT
document('/db/test/abc.xml', '/db/test/def.xml')//titre
```

collection()

```
collection('/db/src')//ETUDIANT
```

xcollection()

```
xcollection('/db/src')//ETUDIANT
```

L'API XML :DB pour une interaction avec JAVA

- Cette API propose une interface vers les bases de données permettant la gestion de XML (en natif ou non).
- Elle comporte des objets *drivers*, *collections*, *services* et *resources*.
- Elle manipule deux types de ressources : *XMLResource* et *BinaryResource*.

L'API XML :DB - Exemple de récupération de document

```
public class RetrieveExample {
  protected static String URI = "xmldb:exist://localhost:8080/exist/xmlrpc";

  public static void main(String args[]) throws Exception {
   String driver = "org.exist.xmldb.DatabaseImpl";

   // initialize database driver
   Class cl = Class.forName(driver);
   Database database = (Database) cl.newInstance();
   DatabaseManager.registerDatabase(database);

   // get the collection
   Collection col = DatabaseManager.getCollection(URI + args[0]);
   col.setProperty(OutputKeys.INDENT, "no");
   XMLResource res = (XMLResource)col.getResource(args[1]);
   if(res == null)
    System.out.println("document not found!");
   else
    System.out.println(res.getContent()); }
}
```

L'API XML :DB - Exemple de requête

```
public class QueryExample {
  public static void main(String args[]) throws Exception {
   String driver = "org.exist.xmldb.DatabaseImpl";
   Class cl = Class.forName(driver);
   Database database = (Database)cl.newInstance();
   DatabaseManager.registerDatabase(database);

   Collection col = DatabaseManager.getCollection(
    "xmldb:exist://localhost:8080/exist/xmlrpc/db");
   XPathQueryService service =
    (XPathQueryService) col.getService("XPathQueryService", "1.0");
   service.setProperty("indent", "yes");

   ResourceSet result = service.query(args[0]);
   ResourceIterator i = result.getIterator();
   while(i.hasMoreResources()) {
    Resource r = i.nextResource();
    System.out.println((String)r.getContent());
   }
  }
}
```

Plan

Références

- W3C : http ://www.w3.org/TR/xquery/
- eXist : http ://exist-db.org
- SAXON : http ://saxon.sourceforge.net
- Fonctions internes XQuery :
  http ://www.w3.org/TR/xquery-operators/