Module MLBDA
Master Informatique
Spécialité DAC

Cours 7 – Xquery

Rév. 22/11/2017

Plan

- Concepts des langages de requêtes XML
  - motivations
  - caractéristiques
- Langages de requêtes pour XML
- XQuery
- Exemples de requêtes

Manipulation de données XML

- Produire du XML
  - Données brutes → documents XML
    → données XML
  - Intégrer des données hétérogènes
- Données XML = collection de documents XML
- Document XML = arbre
  - Structure flexible pas toujours connue à l'avance
  - Peut varier selon les documents

Navigation dans des données XML

- Interroger = Naviguer dans un arbre ayant une structure variable
  - Expression régulière sur des chemins
  - Xpath : permet de sélectionner un sous–arbre
    - Mais insuffisant pour être un langage de requêtes
    - car ne permet pas de construire un sous-arbre.

Besoin d'interroger des données XML

- Besoins pour la Recherche d'Information (RI) et les bases de données (SGBD).
- XML pour la RI
  - Recherche « full text » et par contexte
  - Indexation
- XML pour les SGBD
  - Interroger la structure complexe des données
  - Construire une structure complexe
  - Langage déclaratif : expressivité, optimisation, ...

⇒ Requête XML = SQL + RI + Navigation

Expressivité du langage de requêtes

- Requête déclarative: *pattern* + *filtre* + *constructeur*
- Pattern et filtre
  - Navigation dans un arbre, traversée de références
  - Combiner des arbres : jointure
  - Filtre logique : and, or, négation, etc.
- Constructeur :
  - Créer une nouvelle structure d'arbre
  - Tri, regroupement
  - Imbrication: ajouter des niveaux
- Fonctions externes: agrégats, comparaison de chaînes de caractères, etc.

Spécification des besoins: Use Cases (W3C)

- Use Case « XMP » : Experiences and Exemplars

- Autres USE CASES:
  - TREE : requêtes préservant la hiérarchie
  - SEQ : requêtes basée sur des séquences
  - R : accès à des données relationnelles
  - SGML : standard generalized markup language
  - TEXT : recherche full-text
  - NS : requêtes avec des espaces de noms (namespaces)
  - PARTS : recursive parts explosion
  - REF : requêtes utilisant des références
  - FNPARM : requêtes avec fonctions et paramètres

DTD utilisée pour les Use Cases

www.bn.com/bib.xml

```
<!ELEMENT bib (book*)>
<!ELEMENT book (title, (author+| editor+),
                publisher, price)>
<!ATTLIST book year CDATA #REQUIRED>
<!ELEMENT author (last, first)>
<!ELEMENT editor (last, first, affiliation )>

<!ELEMENT title (#PCDATA )>
<!ELEMENT last (#PCDATA )>
<!ELEMENT first (#PCDATA )>
<!ELEMENT affiliation (#PCDATA )>
<!ELEMENT publisher (#PCDATA )>
<!ELEMENT price (#PCDATA )>
```

Document XML utilisé pour les Use Cases

www.bn.com/bib.xml

```
<bib>
    <book year="1994">
       <title>TCP/IP Illustrated</title>
       <author><last>Stevens</last><first>W.</first></author>
       <publisher>Addison-Wesley</publisher>
       <price> 65.95</price>
    </book>

    <book year="1992">
       <title>Advanced Programming in the Unix
environment</title>
       <author><last>Stevens</last><first>W.</first></author>
       <publisher>Addison-Wesley</publisher>
       <price>65.95</price>
    </book>

    <book year="2000">
       <title>Data on the Web</title>
       <author><last>Abiteboul</last><first>Serge</first></author>
       <author><last>Buneman</last><first>Peter</first></author>
       <author><last>Suciu</last><first>Dan</first></author>
       <publisher>Morgan Kaufmann Publishers</publisher>
       <price> 39.95</price>
    </book>

 <book year="1999">
       <title>The Economics of Technology and Content for Digital
TV</title>
       <editor>
            <last>Gerbarg</last><first>Darcy</first>
             <affiliation>CITI</affiliation>
       </editor>
          <publisher>Kluwer Academic Publishers</publisher>
       <price>129.95</price>
    </book>
 </bib>
```

Ce doc est aussi dans le POLY de TD

Requêtes essentielles (1)

- **Sélection et extraction** : *tous les titres des ouvrages publiés par Eyrolles depuis 2000*
- **Flattening** : l'arbre XML de la base est « mis à plat » (ex: *aplatir la structure imbriquée book (title, author) en faisant apparaître un n-uplet (title, author) par auteur de livre*)
- **Préserver la structure** : afficher la base dans sa version originale (*regrouper les livres par titre*)
- **Changer la structure par imbrication de requête** : *lister la base par auteur*
- **changer la structure par opérateur de regroupement :** *classement des livres par auteur*

Requêtes essentielles (2)

- **Combiner plusieurs sources de données** : joindre la base des livres et celle des prix pour avoir les livres et leurs prix.
- **Indexer les éléments de la structure** : *lister les livres par leur titre et les deux premiers auteurs* (et un élément `et al` s'il y a plus que deux auteurs)
- **Trier les résultats** : *titre des livres par ordre alphabétique*
- **Accès approximatif par les éléments (tags)** : *sélectionner les livres dont une des balises contient l'expression régulière '*or' (author, editor) et dont la valeur est 'Martin'*
- **Accès approximatif par le contenu** : *retrouver les sections ou les chapitres traitant de XML (indépendamment du niveau d'imbrication)*

XQuery

- Spécification du W3C (v1.0, oct.2004 → v. 3.1 mars 2017)
  - inspiré de SQL
  - satisfait les requêtes des Use Cases
  - construit au-dessus de Xpath

- Une requête en XQuery est une expression qui
  - lit un ou plusieurs documents XML (ou des fragments)
  - **renvoie une séquence d'arbres** (fragments XML)

Expressions Xquery

- Plan des diapos qui suivent:
  - expressions simples
  - expressions de chemins
  - comparaisons
  - construction d’éléments
  - expressions FLWOR (prononcer flower)
  - conditions
  - quantificateurs
  - types de données

Expressions simples

- Valeurs atomiques : **`32, "coucou"`**
- Valeurs construites : **`true(), false(), integer("12"), date ("01/01/2017")`**
- Variables : *$nom* (chaîne de caractères précédée de $)
- Opérateurs sur les éléments
  - logiques : *and or*
  - arithmétiques : + - * *div mod*
  - comparaison :
    - Valeurs *eq, ne, lt, le, gt, ge*
    - Générale : =, !=, <, < =, >, >=
    - ordre sur les noeuds : << , >> (precedes, follows)

Expressions de chemin

Ex : `document("bib.xml")//book//author[last="martin"]`

sélectionne les éléments de type **author** (filtrée sur le nom des auteurs, élément fils **last**).

- Toutes les expressions XPath sont des expressions de Xquery.

```
/bib/book[last()]
./child::author[position() >1]
//book[@year= "2002"]/author/last
```

- Le **contexte** d'une expression de chemin est:
  - **Un document:** `document("bib.xml")//book/author`
  - **Une Variable:** `$alice/last`
  - **Le contexte courant:** `.`

Séquences (1)

- Une séquence est une collection ordonnée de zéro ou plusieurs items (nœud et/ou valeur atomique)
- On peut construire, ou filtrer des séquences
- Construction : l'opérateur virgule « `,` » évalue chacune des opérandes et concatène les séquences résultats, dans l'ordre, en une seule séquence résultat.
  - `()` est une séquence vide
  - `(10, 1, 2, 3, 4)` est une expression dont le résultat est une séquence de 5 entiers
  - `(10, (1,2), (), (3,4))` renvoie la séquence `(10,1,2,3,4)`
  - `($prix, $prix * 0.8 )` renvoie la séquence 10, 8 si `$prix` a la valeur 10.

Séquences (2)

- Filtre :

  `$produits` contient une séquence de produits,
  `$produits[prix > 100]` les produits dont le prix est supérieur à 100.

- Combiner des séquences:

  `union, intersect, except`

  Exple: `$s1 union $s2`

Comparaisons de valeurs

- Les opérateurs **`eq, ne, lt, le, gt, ge`** permettent de comparer des valeurs simples
  - **`$book1/title eq "Data on the Web"`**
    - renvoie **`true`** ssi **`$book1`** a exactement un élément fils **`title`** dont la valeur est la chaîne "**`Data on the Web`**"

- Exple d'inégalité
  - **`$book1/price gt 100`**
    - renvoie **`true`** ssi le prix est > 100

- Le nom de l'élément n'est pas comparé
  - **`<a>5</a> eq <a>5</a>`** renvoie **`true`**
  - **`<last>Bob</last> eq <first>Bob</first>`**
    renvoie **`true`**

Comparaisons d'arbres

- Ne **pas** utiliser **`eq`** pour comparer des arbres :

- Si **`$a`** est un auteur (contenant les éléments first et last)
  - **`$a eq "SmithDan"`** : ambiguïté sur les valeurs de first et last
  - **`$a/last eq "Smith" and $a/first eq "Dan"`** : correct

- OU utiliser la fonction deep-equals
  - **`deep-equals($a1,$a2)`**

Comparaisons générales

- Comparer des **séquences**
  - Les opérateurs =, !=, <, <=, >, >= s'appliquent à des séquences de longueur quelconque.
  - **`$s1 = $s2`**
    - **renvoie vrai s'il existe x dans `$s1` et y dans `$s2` tq `x eq y`**
- Exemples :
  - **`$book1/author/last = "Ullman"`**
    - vrai s'il existe un fils **`author`** ayant un fils **`last`** dont la valeur est la chaîne de caractère **`Ullman`**
  - **`(1,2) = (4,3,1)`** renvoie **`true`**
  - **`(1,2) != (2,3)`** renvoie **`true`**

Comparaisons de noeuds

- Les opérateurs **`is, <<, >>`** permettent de comparer deux nœuds, par leur identité ou par leur ordre dans le document

- Si l'une des opérandes est la séquence vide, le résultat est une séquence vide.

- Une comparaison avec **`is`** renvoie **`true`** si les deux nœuds ont la même identité (càd s'il s'agit du même nœud)

- Une comparaison avec **`<<`** (resp. **`>>`**) renvoie **`true`** si le nœud de l'opérande gauche précède (resp. suit) le nœud de l'opérande droite dans l'ordre du document.

Exemple

```
<a>5</a> is <a>5</a>
```

renvoie **`false`** (chaque nœud construit a sa propre identité)

```
//book[year= "2002"] is //book[title= "Data on the
  Web"]
```

renvoie **`true`** si les deux expressions correspondent au même nœud.

```
//produits[ref="123"] << //produits[ref="456"]
```

renvoie **`true`** si le nœud de gauche apparaît avant le nœud de droite dans le document.

Construction d'éléments

- Il est possible de construire des éléments à l'intérieur des requêtes, soit directement en XML, soit en utilisant des expressions Xquery, entre {}.

```
<book isbn="isbn-1234567890">
    <titre>100 ans de solitude</titre>
    <auteur>
        <prenom>Gabriel</prenom>
        <nom>Garcia Marquez</nom>
    </auteur>
</book>
```

Crée un élément **book**, avec un **titre** et un **auteur**, un **nom** et un **prenom**. Il a sa propre identité.

Construction d’éléments

```
<req>{ $b/titre }</req>
```

Seules les expressions entre {} sont évaluées. Les variables doivent être liées à un fragment.

Résultat

Si $b est liée à l'élément

```
<book isbn="isbn-1234567890">
        <titre>100 ans de solitude</titre>
        <auteur>
                <prenom>Gabriel</prenom>
                <nom>Garcia Marquez</nom>
        </auteur>
</book>
```

La requête :

```
<p> En vacances, j'ai lu <req>{ $b/titre }</req> </p>
```

Construit le résultat :

```
<p> En vacances, j'ai lu :
  <req> <titre>100 ans de solitude</titre> </req>
</p>
```

Construction d'éléments

On peut aussi construire des éléments et des attributs de la façon suivante :

```
element book
{ attribute isbn {"isbn-1234567890" },
  element titre {"100 ans de solitude"},
  element auteur {
     element first {"Gabriel"},
     element last {"Garcia Marquez" }
     }
}
```

Le nom et le contenu des éléments et des attributs peuvent être calculés par des expressions.

Expression FLWOR

FOR ... LET ... WHERE ...ORDER BY … RETURN

Exemple : personnes ayant édité plus de 100 livres

```
FOR $p IN document("bib.xml")//publisher
LET $b:=document("bib.xml")//book[publisher = $p]
WHERE count($b) > 100
RETURN $p
```

FOR **itère** sur une liste ordonnée d'éléments publisher désignée par **$p.**

LET **affecte** une liste d'éléments **book** à la variable **$b.**

On a une liste de n-uplets **($p,$b).**

WHERE filtre cette liste pour ne retenir que les n-uplets souhaités.

RETURN construit pour chaque n-uplet la valeur résultat.

FOR et LET

- **L'itération `FOR $var in` exp**.
  - A chaque itération: affecte à la variable **`$var`** un item de la séquence **exp**

- La clause **`LET $var:=` exp affecte** la variable **`$var`** avec le résultat de **exp** (qui peut être une valeur ou une séquence)

- Les clauses FOR et LET peuvent contenir plusieurs variables, et peuvent apparaître plusieurs fois dans une requête (utile pour la jointure).

Exemples

```
let $s := (<un/>, <deux/>, <trois/>)
return <out>{$s}</out>
```

Résultat :

```
<out>
   <un/>
   <deux/>
   <trois/>
</out>
```

```
for $s in (<un/>, <deux/>, <trois/>)
return <out>{$s}</out>
```

Résultat :

```
<out> <un/> </out>
<out> <deux/> </out>
<out> <trois/> </out>
```

WHERE

- La clause **`WHERE`** **exp** permet de filtrer le résultat par rapport au résultat booléen de **exp**.

```
for $x in document("bib.xml")//book
where $x/author/last = " Ullman "
return $x/title
```

Renvoie les titres des livres dont Ullman est auteur

ORDER BY et RETURN

La clause return est évaluée une fois pour chaque n-uplet du flot de données. Le résultat de ces évaluations est concaténé.

En l'absence de clause ORDER BY, l'ordre est déterminé par les clauses FOR et LET.

ORDER BY permet de réordonner les n-uplets dans l'ordre croissant (ascending) et décroissant (descending).

```
for $e in $employees
order by $e/salary descending
return $e/name

for $b in $books/book
order by $b/price, b/title
return $b
```

Conditionnelle

IF ... THEN ... ELSE

```
<books>
{for $x in document("bib.xml")//book
Where $x/author/last = " Ullman "
Return
If ($x/@year > "2005")
Then <book>{$x/title} "est un livre récent" </book>
Else ()
}
</books>
```

Quantificateurs

- **SOME ... IN ... SATISFIES**
- **EVERY ... IN ... SATISFIES**

`SOME $x in` **expr1** **`SATISFIES`** **expr2** signifie qu'il existe AU MOINS un nœud renvoyé par **expr1** qui satisfait **expr2.**

`EVERY $x in` **expr1** **`SATISFIES`** **expr2** signifie que TOUS les nœuds renvoyés par **expr1** satisfont **expr2**.

```
Every $b in document("bib.xml")//book satisfies $b/@year
```

Renvoie **`true`** si tous les livres ont un attribut **`year`.**

```
some $b in document("bib.xml")//book
   satisfies $b/@year >2003
```

Renvoie **`true`** si au moins un livre a un attribut dont la valeur est supérieure à 2003.

Exemple

On suppose qu'un **book** contient plusieurs éléments **resume**.

```
FOR $b IN document("bib.xml")//book
WHERE SOME $p IN $b//resume SATISFIES
(contains($p, "sailing") AND contains($p, "windsurfing"))
RETURN $b/title
```

Titre des livres mentionnant à la fois *sailing* et *windsurfing* dans le même élément *resume*.

```
FOR $b IN document("bib.xml")//book
WHERE EVERY $e IN $b//editor SATISFIES $e/affiliation="UPMC"
RETURN $b/title
```

Titre des livres dont tous les éditeurs sont affiliés à l'UPMC.

Types

XQuery supporte les types de données de XML Schema, types simples et complexes.

INSTANCEOF : renvoie true si la valeur du premier opérande est du type du deuxième opérande.

TYPESWITCH .. CASE ..DEFAULT ..: branchement en fonction du type

```
typeswitch($customer/billing-address)
case $a as element(*, USAddress) return $a/state
case $a as element(*, CanadaAddress) return
$a/province
default return "unknown"
```

CAST : force un type

```
xs:date("2000-01-01")
```

Exemple (1)

Livres publiés par Addison-Wesley depuis 1991, avec l'année et le titre

```
<bib>
{ for $b in document("www.bn.com")/bib/book
   where $b/publisher = "Addison-Wesley" and $b/@year
   > 1991
   return <book year= "{$b/@year}">
                 {$b/title}
          </book>
}
</bib>
```

Résultat :

```
<bib>
   <book year= "1994">
       <title> Bases de données </title>
   </book>
<book year= "1999">
       <title> Langages de requêtes XML </title>
   </book>
</bib>
```

Exemple (2)

Liste de toutes les paires (titre, auteur), chaque paire étant contenue dans un élément result.

```
<results>
{
for $b in document("www.bn.com")/bib/book,
    $a in $b/author
Return
  <result>
  { $b/title}
  { $a }
  </result>
}
</results>
```

Résultat

```
<results>
    <result>
        <title>TCP/IP Illustrated</title>
        <author> <last>Stevens</last> <first>W.</first> </author>
    </result>
    <result>
        <title>Data on the Web</title>
        <author> <last>Abiteboul</last> <first>Serge</first>
   </author>
    </result>
    <result>
        <title>Data on the Web</title>
        <author><last>Buneman</last>
   <first>Peter</first></author>
    </result>
    <result>
        <title>Data on the Web</title>
        <author><last>Suciu</last> <first>Dan</first> </author>
    </result>

    etc …
</results>
```

Exemple (3)

```
<results>
{
    for $b in document("bib.xml")/bib/book
    return
        <result>
            { $b/title }
            { $b/author  }
        </result>
}
</results>
```

Résultat

```
<results>
    <result>
        <title>TCP/IP Illustrated</title>
        <author><last>Stevens</last><first>W.</first></author>
    </result>
  <result>
        <title>Data on the Web</title>
  <author><last>Abiteboul</last><first>Serge</first></author>
  <author><last>Buneman</last><first>Peter</first></author>
   <author><last>Suciu</last><first>Dan</first></author>
    </result>

    etc …

</results>
```

Jointure

Pour chaque auteur, liste de ses livres.

```
<results>
  {
    for $a in distinct-values
   (document("bib.xml")//author)
    return
        <result>
            { $a }
            {for $b in document("bib.xml")/bib/book,
                  $ba in $b/author
             where $ba = $a
             return $b/title
            }
        </result>
  }
</results>
```

Exemples

Le document *carnet.xml* est un carnet (c) de personnes (p).

Chaque personne a un nom (n), un age (a) et une ville de résidence (v).

```
<?xml version="1.0">
<c>
  <p n="paul" a="60" v="Paris"/>
  <p n="martin" a="30" v="Paris"/>
  <p n="jean" a="60" v="Nice"/>
  <p n="claude" a="50" v="Lyon"/>
</c>
```

Que renvoie la requête suivante ?

```
<resultat>
  {for $a in distinct-values(document("carnet.xml")//p/@a)
    return
       <age a='{$a}' >
       {document("carnet.xml")//p[@a=$a]}
       </age>
  }
</resultat>
```

Combien d’éléments <proche> renvoie la requête ?

```
<resultat>
{ for $c in document("carnet.xml")/c, $p1 in $c/p, $p2 in
   $c/p
  where  $p2/@v=$p1/@v
  return
  <proche> {$p1} {$p2} </proche>
 }
</resultat>
```

Et celle-ci ?

```
<resultat>
 { for $c in document("carnet.xml")/c, $p1 in $c/p, $p2 in
   $c/p
   where $p2/@v=$p1/@v and $p1/@n != $p2/@n
   return
      <proche> {$p1} {$p2} </proche>
  }
</resultat>
```

Que fait cette requête ? Que renvoie-t-elle ?

```
<resultat>
 { for $v in distinct-values(document("carnet.xml")//p/@v)
   for $p in document("carnet.xml")//p[@v=$v]
   where every $x in document("carnet.xml")//p[@v=$v] satisfies
   $x/@a <= $p/@a
   return $p
 }
</resultat>
```

Conclusion

XML : structure d'arbre

- navigation grâce à XPath
- caractérisation des sous-arbres grâce aux axes

Requêtes :

- travaillent sur les sous-arbres construits
- génèrent un sous-arbre extrait ou calculé

XQuery :

- langage très puissant, comprenant toutes les fonctionnalités de SQL

Liens

- www.w3.org/TR/xquery
- www.w3.org/TR/xquery-requirements
- www.w3.org/TR/xquery-use-cases
- http://www-db.research.bell-labs.com/user/simeon/xquery.ps

Biblio
sur les langages de requêtes pour XML

- XML Query Languages: Experiences and Exemplars (ed. M. Fernandez, J. Simeon, P. Wadler)
- Bases de réflexion
  - SQL
  - XML-QL (T&T)
  - YATL (INRIA)
  - Lorel (Stanford)
  - XQL
  - Quilt (IBM)
- …