Module MLBDA
Master Informatique
Spécialité DAC

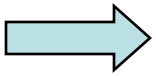
Cours 7 – XQuery

Langages de requêtes XML

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

Interrogation de documents XML

- extraire rapidement un sous-ensemble de données
- transformer des données en documents XML
- intégrer des données provenant de sources hétérogènes
- retrouver des documents (full text)
- générer des nouveaux documents, dériver de nouvelles présentations à partir d'un document
- indexer des documents
- effectuer des recherches par contexte
- requêtes sur documents structurés (données imbriquées, etc.)



Bases de données et Recherche d'Information

Fonctionnalités BD

- Tout ce qu'offre SQL
  - déclaratif
  - indépendant du système
  - support de quantificateurs existentiels et universels
  - support d'agrégats
  - recherche d'un élément dans un document
  - combiner des informations provenant de plusieurs documents
  - doit pouvoir être optimisé
  - support de types simples et complexes (SQL3)
  - ...

Fonctionnalités RI

- support d’opérations sur la structure des documents
- transformation et création de structures XML
- recherche d’informations dans le document
- recherche efficace de texte dans de grandes collections de documents

Syntaxe XML

- Une seule racine, imbrication correcte : structure de graphe
- Structure exacte des données pas toujours connue, ou pouvant varier selon les données

Il est plus facile d'utiliser une forme navigationnelle d'interrogation, basée sur des expressions de chemin.

⇒ Langage de requêtes = SQL + RI + Navigation (SQL généralisé à XML)

Caractéristiques du langage de requêtes (1)

Caractéristiques essentielles :

- une formulation de la requête en trois parties : *pattern* + *filtre* + *constructeur*
- possibilité d'imbriquer des requêtes, de les grouper, d'indexer, de faire du tri (permet de restructurer un document)
- disposer d'un opérateur de jointure, permettant de combiner des données provenant de diverses portions de documents
- interroger sans connaître nécessairement la structure du document (accès à des données imbriquées de façon arbitraire)

Caractéristiques du langage de requêtes (2)

Fonctionnalités utiles :

- pouvoir vérifier l'absence d'une information
- permettre l'utilisation d'alternative dans les requêtes (|)
- pouvoir faire appel à des fonctions externes (agrégats, comparaison de chaînes de caractères, etc.)
- utilisation d'opérateurs de navigation (simplifie la manipulation des données par référence(ID, IDREFS))

DTD Exemple

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 )>
```

10 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*

10 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)*

Use Cases (W3C)

- Use Case « XMP » : Experiences and Exemplars
- Use Case « TREE » : requêtes préservant la hiérarchie
- Use Case « SEQ » : requêtes basée sur des séquences
- Use Case « R » : accès à des données relationnelles
- Use Case « SGML » : standard generalized markup language
- Use Case « TEXT » : recherche full-text (recherche de chaînes de caractères dans un document XML)
- Use Case « NS » : requêtes utilisant des espaces nominaux (namespaces)
- Use Case « PARTS » : recursive parts explosion
- Use Case « REF » : requêtes utilisant des références
- Use Case « FNPARM » : requêtes utilisant des fonctions et paramètres

Langages de requêtes pour XML

- Buts :
  - recherche d'informations dans les documents
  - travaille directement sur la structure XML
- Bases de réflexion
  - SQL
  - XML-QL (T&T)
  - YATL (INRIA)
  - Lorel (Stanford)
  - XQL
  - Quilt (IBM)
- http://www-db.research.bell-labs.com/user/simeon/xquery.ps

XQuery

- Spécification du W3C (version 1.0, oct.2004)
  - inspiré de SQL
  - satisfait les contraintes émises (requêtes essentielles, Use Cases)
  - Construit au-dessus de XPath
    - Remarque : Xpath n'est pas un langage de requêtes, car le contenu d'un élément (fragment correspondant au sous-arbre de l'élément) destination n'est pas extrait.
- Une requête en XQuery est une expression qui
  - lit un ensemble de documents XML (ou des fragments)
  - renvoie une séquence de fragments XML bien formés

XQuery

- Requête sur un arbre : parcours de l'arbre
- Variables : *$nom*
- Appels de fonction : *nomfonction(...)*
- 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)
- Notion de séquences (listes)
- Construction d'éléments

Requête Xquery

Une requête Xquery est une composition d'expressions. Elle renvoie une séquence d'éléments ou de valeurs (fragments de documents XML).

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

est une expression de chemin (XPath) qui renvoie une séquence d'éléments de type **`author`** (filtrée sur le nom des auteurs, élément fils **`last`**).

Expressions XQuery

- Principales formes des expressions XQuery:
  - expressions simples
  - expressions de chemins
  - comparaisons
  - construction d'éléments
  - expressions FLOWR (flower) : **`for, let, where, order by, return`**
  - conditions
  - quantificateurs
  - types de données

Expressions simples

- Valeurs atomiques : `32, "coucou"`
- Valeurs construites : `true(), false(), integer("12"), date ("29-11-2013")`
- Référence à une variable (chaîne de caractères précédée de $) : `$var`
- Expression de contexte : `"."`
  - `document("bib.xml")//book[count(./author)>1]`
- Appel de fonction :
  - `fonction-sans-argument()`
  - `fonction-avec-deux-arguments(1, 2)`

Expressions de chemin

- Toutes les expressions XPath sont des expressions de Xquery.
  - `/bib/book[last()]`
  - `child::author[position() >1]`
  - `book[@year= "2002"]/author/last`

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 d'items (nœuds et/ou valeurs).
- 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, $prix)` renvoie la séquence 10, 10 si `$prix` a la valeur 10.

Séquences (2)

- Filtre : si **`$produits`** contient une séquence de produits, **`$produits[prix >100]`** renvoie les produits dont le prix est supérieur à 100.
- Les séquences peuvent être combinées grâce aux opérateurs **`union, intersect, except`**
  - **`$var1 union $var2`** renvoie l'union des séquences de nœuds désignées par les variables **`$var1`** et **`$var2`**.

Comparaisons de valeurs

- Les opérateurs **`eq, ne, lt, le, gt, ge`** permettent de comparer des valeurs simples
  - **`$book1/author eq "Ullman"`** renvoie **`true`** ssi **`$book1`** a exactement un élément fils **`author`** dont la valeur est la chaîne de caractère **`Ullman`**.
  - **`//produit[prix gt 100]`** contient un prédicat filtrant les produits dont le prix est supérieur à 100. Si l'élément **`produit`** n'a pas de sous-élément **`prix`**, la valeur du prédicat est une séquence vide, et le produit n'est pas sélectionné.
  - **`<a>5</a> eq <a>5</a>`** renvoie **`true`**
  - **`<a>5</a> eq <b>5</b>`** renvoie **`true`**

Comparaisons générales

- Les opérateurs `=, !=, <, <=, >, >=` s'appliquent à des séquences de longueur quelconque.
- La comparaison renvoie la valeur **`true`** s'il existe un item dans le premier opérande qui correspond à un item dans le deuxième opérande.
- Exemples :
  - **`$book1/author = "Ullman"`** renvoie **`true`** s'il existe un élément fils **`author`** dont la valeur est la chaîne de caractère **`Ullman`**
  - **`(1,2) = (2,3)`** 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)

```
/books/book[isbn= "1234567890"] is
  /books/book[call= "QA67.7 C123"]
```

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

```
<exemple>
<p> Ceci est une requête. </p>
<req> $b/titre </req>
<p> Ceci est le résultat de la requête. </p>
<req>{ $b/titre }</req>
 </exemple>
```

Seules les expressions entre {} sont évaluées. Les variables doivent être liées aux fragments appropriés.

Résultat

Si $b désigne l'élément

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

Le résultat est :

```
<exemple>
<p> Ceci est une requête. </p>
 <req> $b/titre </req>
<p> Ceci est le résultat de la requête. </p> <req>
   <titre>100 ans de solitude</titre>
   </req>
</exemple>
```

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 FLOWR

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 génère une liste ordonnée de liens de noms d'éditeurs désignée par **$p.**

LET associe à chacun de ces liens un autre lien de la liste des éléments **book** avec cet éditeur avec **$b.** On a une liste ordonnée de n-uplets **($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

- La clause **`FOR $var in`** **exp** affecte la variable **`$var`** successivement avec chaque item de la séquence renvoyée par **exp.**

- La clause **`LET $var:=`****exp** affecte la variable **`$var`** avec la séquence entière renvoyée par **exp.**

- 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[price < 100]
order by $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

```
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 de tous les livres mentionnant à la fois *sailing* et *windsurfing* dans le même élément *resume*.

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

Titre des livres mentionnant *sailing* dans chaque élément *resume*.

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,
    $t in $b/title
    $a in $b/author
Return
  <result>
  { $t}
  { $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>
</results>
```

Exemple (3)

```
<results>
{
    for $b in doc
  distinct(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>
</results>
```

Jointure

Pour chaque auteur, liste de ses livres.

```
<results>
  {
    for $a in distinct(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>
```

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 (restriction, projection, jointure, imbrication, restructuration, agrégation, tri, quantificateurs, etc.)

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

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}
       {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>
```