XQuery

Dan VODISLAV

CY Cergy Paris Université

Licence Informatique L3

Plan

- Principes, historique
- Modèle de données
- Expressions XQuery
  - Expressions simples
  - Expressions complexes
- Expressions FLOWR
- Fonctions et opérateurs

XQuery

- Langage de requêtes pour données XML
  - L'équivalent de SQL pour les données relationnelles
- XQuery 1.0:
  - W3C Working Draft 02/05/2003 → W3C Recommendation 23/01/2007 → W3C Edited Recommendation 14/12/2010
- XQuery 3.0 (Proposed Recommendation) : XQuery 1.0 + GROUP BY, WINDOW, TRY/CATCH, etc.

Pourquoi un autre langage?

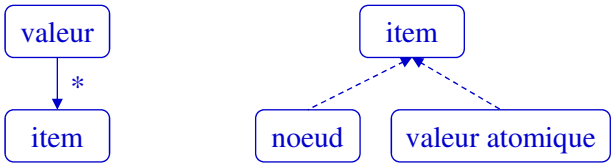
- XPath
  - Seulement extraction de fragments d'arbres
- SQL
  - Le document XML doit être transformé en relations
- SQL + XPath
  - Document (ou fragments) sous forme d'attributs XML (CLOB)
  - Mélange des deux langages → problèmes d'optimisation
- XSLT
  - Pour créer une copie transformée d'un document → pourrait servir à extraire des parties d'un document, donc à interroger un document
  - Mal adapté à de gros volumes de données
- XQuery
  - Vrai langage de requêtes XML

Langage fonctionnel

- XQuery = langage fonctionnel typé
  - Requête = composition d'expressions (fonctions)
  - Chaque expression retourne une valeur (typée) ou une erreur
  - Expressions *sans effets de bord* (sans modification des données)
- Expressions
  - Simples: valeurs atomiques, constructeurs, variables, etc.
  - Complexes: expressions de chemin, expressions FLWOR, etc.
- Éléments de syntaxe
  - On fait la différence entre minuscules et majuscules → les mots-clés du langage (for, let, where, if, …) doivent être écrits en minuscules
  - Commentaires: n'importe où *(: Ceci est un commentaire :)*

Modèle de données de XQuery

- Modèle de données XDM spécifique à XQuery (et XPath 2.0)
  - Basé sur les *séquences ordonnées*
  - ***Valeur*** = séquence ordonnée (liste) d'*items*
  - ***Item*** = *nœud* (tout type DOM) ou *valeur atomique*
    - Document, élément, attribut, texte, … + valeurs atomiques de différents types
  - Chaque nœud et chaque valeur atomique a un ***type*** (XML Schema)
  - Résultat de requête XQuery = valeur = séquence d'items



Séquences

- Pas de distinction entre item et séquence de longueur 1
  - Ex: 39 = (39)
- Une séquence peut contenir des valeurs hétérogènes
  - Ex: (39, "toto", <toto/>)
- Pas de séquences imbriquées
  - Ex: (39, (1, 2), "toto", <toto/>) = (39, 1, 2, "toto", <toto/>)
- Une séquence peut être vide
  - Ex: ()
- L'ordre est important
  - Ex: (1, 2) ≠ (2, 1)

Expressions XQuery simples

- Valeurs atomiques littérales (des types simples XML Schema)
  Ex: *39*, *"toto"*, *3.9*, etc.
- Nœuds XML sous forme littérale
  Ex: *<film annee="2007">*
  <titre>La môme</titre>
  </film>
- Valeurs obtenues par des constructeurs simples
  Ex: *true()*, *false()*, *date("2006-12-08")*
- Collections de documents, documents
  - doc(*uri-document*) → retourne un item de type nœud *Document*
  - collection(*uri-collection*) → retourne une séquence de nœuds *Document*

  Ex: *doc("biblio.xml")*, *collection("cinema/films")*
- Séquences construites
  Ex: *(1, 2, 3, 4, 5)*, *1 to 5*, *(1 to 3, 4, 5)*
- Variables
  - Ont un nom (précédé du signe $): *$x*, *$toto*, etc.
  - Ont une valeur: séquence d'items

Expressions XQuery complexes

- Expressions de chemin (XPath 2.0)
  - Toute expression produisant une séquence de nœuds peut être une étape
- Expressions FLWOR (*For-Let-Where-Order by-Return*) avec définition de variables
- Tests (*If-Then-Return-Else-Return*)
- Fonctions
  - Fonctions prédéfinies: les mêmes que pour XPath 2.0
    - voir « XQuery 1.0 and XPath 2.0 Functions and Operators » *http://www.w3.org/TR/xpath-functions/*
  - Fonctions utilisateur
- Mélange de littéraux et expressions complexes
  - Chaque expression doit être placée entre { } pour qu'elle soit évaluée

  Ex. *<comedies>*
  {doc("films.xml")//film[@genre="comedie"]}
  </comedies>

Opérations

- Opérations arithmétiques
  - Ex: 1+2, 3.14-$x, $y mod 2
- Opérations sur les séquences
  - Concaténation: virgule
    - Ex: (1, 2), (‘a’, <toto/>) = (1, 2, ’a’, <toto/>)
  - Union, intersection, différence
- Opérateurs de comparaison
  - Valeurs atomiques
  - Nœuds
  - Séquences
- Opérateurs booléens
  - and, or, not, …

Exemple

- Document *bib.xml*

```
<bib>
  <book title="Comprendre XSLT">
    <author><la>Amann</la><fi>B.</fi></author>
    <author><la>Rigaux</la><fi>P.</fi></author>
    <publisher>O'Reilly</publisher>
    <price>28.95</price>
  </book>
  <book year="2001" title="Spatial Databases">
    <author><la>Rigaux</la><fi>P.</fi></author>
    <author><la>Scholl</la><fi>M.</fi></author>
    <author><la>Voisard</la><fi>A.</fi></author>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price>35.00</price>
  </book>
  <book year="2000" title="Data on the Web">
    <author><la>Abiteboul</la><fi>S.</fi></author>
    <author><la>Buneman</la><fi>P.</fi></author>
    <author><la>Suciu</la><fi>D.</fi></author>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price>39.95</price>
  </book>
</bib>
```


Expressions de chemin

- Requête:

```
doc("bib.xml")//author
```

- Résultat:

```
<author><la>Amann</la><fi>B.</fi></author>,
<author><la>Rigaux</la><fi>P.</fi></author>,
<author><la>Rigaux</la><fi>P.</fi></author>,
<author><la>Scholl</la><fi>M.</fi></author>,
<author><la>Voisard</la><fi>A.</fi></author>,
<author><la>Abiteboul</la><fi>S.</fi></author>,
<author><la>Buneman</la><fi>P.</fi></author>,
<author><la>Suciu</la><fi>D.</fi></author>
```

- Le résultat est une *séquence* de noeuds XML

Expressions de chemins XPath 1.0 et 2.0

- XPath 1.0
  - expression de chemin = suite d'*étapes* séparées par '/'
  - étape = un *axe*, un *filtre* et une *séquence de prédicats*
- XPath 2.0 (fonctionnel)
  - expression de chemin = suite d'*expressions d'étape*
  - expression d'étape =
    - soit une étape XPath 1.0 (axe, filtre, prédicats)
    - soit une expression XQuery (simplifiée)

Exemple: expressions de chemin

- `doc("bib.xml")/bib//book[1]/publisher`
  - L'éditeur du premier livre
- `doc("bib.xml")//(book union author union publisher)`
  - tous les livres, auteurs et éditeurs
- `doc("bib.xml")//book/(* except price)`
  - pour chaque livre tous les éléments fils sauf le prix.
- `doc("bib.xml")//book[every $a in author satisfies contains($a/la,'a')]`
  - tous les livres dont tous les auteurs ont la lettre 'a' dans leur nom de famille

Construction de nœuds XML

- Cas 1: nom connu, contenu construit par une expression
  - Requête:

```
<auteurs>
  { doc("bib.xml")//book[2]/author/la }
</auteurs>
```

  - Résultat:

```
<auteurs>
  <la>Rigaux</la>
  <la>Scholl</la>
  <la>Voisard</la>
</auteurs>
```

- La requête fait partie d'un document XML, le résultat est transformé en fragment XML

Construction de nœuds XML (suite)

- Cas 2: le nom et le contenu sont calculés
  - Constructeurs d'élément et d'attribut
    - `element` *{* expr-nom *} {* expr-contenu *}*
    - `attribute` *{* expr-nom *} {* expr-contenu *}*
  - Requête:

```
element { doc("bib.xml")//book[1]/name(@*[1]) } {
  attribute { doc("bib.xml")//book[1]/name(*[3]) }
    { doc("bib.xml")//book[1]/*[3] }
}
```

  - Résultat:

```
<title publisher="O'Reilly"/>
```

Différence de séquences

- On garde les éléments de la première séquence qui n'apparaissent pas dans la seconde séquence
  - Élimination des doublons
- Exemple: séquences de noeuds
  - Requête:

```
<livre>
  Tous les sous-elements sauf les auteurs:
  { doc("bib.xml")//book[1]/(* except author) }
</livre>
```

  - Résultat:

```
<livre>
  Tous les sous-elements sauf les auteurs:
  <publisher>O'Reilly</publisher>
  <price>28.95</price>
</livre>
```


Concaténation de séquences

- Différences avec l'union
  - La concaténation préserve l'ordre
  - L'union élimine les doublons
- Exemple
  - Requête:

```
<livre>
  Le prix suivi des auteurs:
  { doc("bib.xml")//book[1]/(price,author) }
</livre>
```

  - Résultat:

```
<livre>
  Le prix suivi des auteurs:
  <price>28.95</price>
  <author><la>Amann</la><fi>B.</fi></author>
  <author><la>Rigaux</la><fi>P.</fi></author>
</livre>
```

Transformation nœud → valeur

- Exemple: utilisation de la fonction prédéfinie *string()*
  - Requête:

```
"Les auteurs du premier livre sont",
doc("bib.xml")//book[1]/author/string(la)
```

  - Résultat: la séquence

```
Les auteurs du premier livre sont, Amann, Rigaux
```


Comparaison de valeurs atomiques avec *eq*

- Requête:

```
doc("bib.xml")//book/author[la eq "Scholl"]
```

- Résultat:

```
<author><la>Scholl</la><fi>M.</fi></author>
```

- Requête:

```
doc("bib.xml")//book[author/la eq "Scholl"]
```

- Résultat:

```
ERROR
```

  - Erreur, car l'expression *author/la* (dans le prédicat) retourne une *séquence* de longueur > 1 !

- Autres comparaisons de valeurs atomiques: *lt*, *gt*, *le*, *ge*, *ne*

Comparaison de séquences avec =

- ***s1 = s2*** ⇔ il *existe* un élément dans ***s1*** égal à un élément dans ***s2***
  - Sémantique existentielle
- Exemple:
  - Requête:

```
count(doc("bib.xml")//book[author/la =
                    ("Scholl", "Rigaux", "Abiteboul")])
```

  - Résultat: `3`
- Autres opérations de comparaison (<, <=, !=, >, >=)
  - Remarques:
    - s1 = s2 et s1 != s2 peuvent être vraies en même temps !
    - Si s1=s2 et s2=s3, il n'est pas nécessairement vrai que s1=s3 !

Comparaison par la position avec <<

- ***n1 << n2*** ⇔ ***n1*** apparaît avant ***n2*** dans le document
- Exemple
  - Requête:

```
<livre>
  { doc("bib.xml")//book[author[la="Abiteboul"] <<
                         author[la="Suciu"]]/@title }
</livre>
```

  - Résultat:

```
<livre title="Data on the Web"/>
```

Comparaison de nœuds avec *is*

- ***n1 is n2*** si ***n1*** est identique à ***n2***
  - Comparaison de l'identité des nœuds, pas de leur valeur
- Exemple
  - Requête:

```
doc("bib.xml")//book[author[2] is author[last()]]
```

  - Résultat:

```
<book title="Comprendre XSLT">
  <author><la>Amann</la><fi>B.</fi></author>
  <author><la>Rigaux</la><fi>P.</fi></author>
  <publisher>O'Reilly</publisher>
  <price>28.95</price>
</book>
```


Expressions FLOWR

- Une expression FLOWR (on dit "flower")
  - Itère sur des séquences (**f**or)
  - Définit des variables (**l**et)
  - Trie les résultats (**o**rder by)
  - Applique des filtres (**w**here)
  - Construit et retourne un résultat (**r**eturn)
- **for** et **return** sont obligatoires

Itération : *for*

- *for* **$var** *in* **exp**
  - affecte à la variable **$var** *successivement* chaque item dans la séquence retournée par **exp**.

- Requête:

```
for $a in doc("bib.xml")//author[la eq "Voisard"]
return $a
```

- Résultat:

```
<author><la>Voisard</la><fi>A.</fi></author>
```


Affectation d'ensembles: *let*

- *let* **$var** *:=* **exp**
  - affecte à la variable **$var** *la séquence entière* retournée par **exp**
- Requête:

```
for $b in doc("bib.xml")//book[1]
let $al := $b/author
return <livre nb_auteurs="{count($al)}">
          { $al }
       </livre>
```

- Résultat:

```
<livre nb_auteurs="2">
  <author><la>Amann</la><fi>B.</fi></author>
  <author><la>Rigaux</la><fi>P.</fi></author>
</livre>
```

Trier avec *order by*

- ***expr1*** *order by* ***expr2*** *(ascending | descending)?*
  - trier les éléments de la séquence retournée par l'expression *expr1* selon les valeurs retournées par **expr2**
- Requête:

```
<livres>
  { for $b in doc("bib.xml")//book
    order by $b/@year
    return <livre> { $b/@title, $b/@year } </livre>
  }
</livres>
```

- Résultat:

```
<livres>
  <livre title="Comprendre XSLT"/>
  <livre title="Data on the Web" year="2000"/>
  <livre title="Spatial Databases" year="2001"/>
</livres>
```


Sélection : *where*

- *where* **exp**
  - permet de filtrer le résultat par rapport au résultat booléen de l'expression **exp**
- Requête:

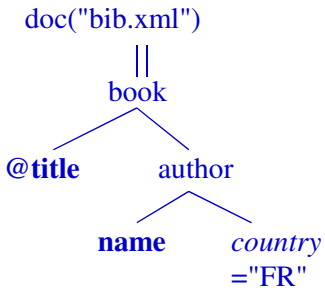
```
for $a in doc("bib.xml")//book
where $a/author[1]/la eq "Abiteboul"
return <livre>{$a/@title}</livre>
```

- Résultat:

```
<livre title="Data on the Web"/>
```

Requêtes motif d'arbre

- Les requêtes les plus courantes
  - Correspondent aux requêtes relationnelles de sélection/projection
- Exemple : *noms des auteurs français et titres des livres qu'ils ont publiés*
- Principe
  - Une variable par nœud de jonction
  - Conditions dans le "where"
  - Résultats dans le "return"



- Équivalent en XQuery

```
for $b in doc("bib.xml")//book, $a in $b/author
where $a/country="FR"
return <resultat>
         {$a/name}
         {$b/@title}
       </resultat>
```


Tests: *if-then-else*

- Requête:

```
<livres>
  { for $b in doc("bib.xml")//book
    where $b/author/la = "Rigaux"
    return if ($b/@year > 2000)
           then <livre recent="true"> {$b/@title}
  </livre>
           else <livre> {$b/@title} </livre>
  }
</livres>
```

- Résultat:

```
<livres>
  <livre title="Comprendre XSLT"/>
  <livre recent="true" title="Spatial Databases"/>
</livres>
```

Quantification

- *some* ***$var*** *in* ***expr1*** *satisfies* ***expr2***
  - *il existe au moins un* noeud retourné par l'expression *expr1* qui satisfait l'expression *expr2* (quantification existentielle).
- *every* ***$var*** *in* ***expr1*** *satisfies* ***expr2***
  - *tous* les nœuds retournés par l'expression *expr1* satisfont l'expression *expr2* (quantification universelle)
- Requête:

```
for $a in doc("bib.xml")//author
where every $b
            in doc("bib.xml")//book[author/la = $a/la]
            satisfies $b/publisher="Morgan Kaufmann Publishers"
return string($a/la)
```

- Résultat:

```
Scholl, Voisard, Abiteboul, Buneman, Suciu
```


Fonctions et opérateurs

- Ils permettent
  - L'accès au nom et au type d'un nœud
  - La construction, comparaison et transformation de valeurs
  - L'agrégation des valeurs d'une séquence
  - …
- Typage
  - Les fonctions et opérateurs sont *typés* (XML Schema)
  - Ils manipulent des séquences et des valeurs typées

Exemple: fonction *avg*

- Requête:

```
for $p in distinct-values(doc("bib.xml")//publisher)
let $l := doc("bib.xml")//book[publisher = $p]
return element publisher {
                  attribute name {string($p)},
                  attribute avg_price { avg($l/price) }
                }
```

- Résultat :

```
<publisher name="O'Reilly"
           avg_price="28.95"/>,
<publisher name="Morgan Kaufmann Publishers"
           avg_price="37.48"/>
```


Exemple: fonction *index-of*

- Requête:

```
<books> {
  let $bl := doc("bib.xml")//book
  for $b in $bl
  return <book> {
           $b/@title,
           attribute no {index-of($bl, $b)}
         }
        </book>
 }
</books>
```

- Résultat:

```
<books>
  <book title="Comprendre XSLT" no="1"/>
  <book title="Spatial Databases" no="2"/>
  <book title="Data on the Web" no="3"/>
</books>
```

Fonctions utilisateur

- XQuery permet à l'utilisateur de définir ses propres fonctions
- Exemple

```
define function NombreAuteurs(book $b)
returns xsd:integer {
  return count($b/author)
}
```

- Le résultat est de type `xsd:integer`

Bibliographie spécifique

- XQuery sur le site W3C
  `http://www.w3.org/TR/xquery`
  `http://www.w3.org/XML/Query`
- Différents types de requêtes XQuery
  `http://www.w3.org/TR/xquery-use-cases`
- J. Melton, J. Buxton, *Querying XML*, Morgan Kaufmann