XQuery

Préparé par
M.G. BELKASMI

Introduction

- Comment interroger des documents XML?
- Solutions :
  - SQL: il faut stocker XML dans une BD relationnel
  - règles XSLT : extraction + transformation (règles)

XQuery

- Généralisation de XPATH
- Un surensemble de SQL
  - Les fonctionnalités sur les tables sont étendus pour supporter des opérations similaires sur les collections d'arbres.
  - Projection d'arbres sur des sous-arbres.(ds XPATH)
  - Selection d'arbre et de sous-arbre(ds XPATH)
  - Toute opération valide sur un type de données peut être effectué pour composé une requête XQuery

XQuery

- Des variables sont utilisables dans les requêtes pour mémoriser un arbre ou pour itérer sur des collections d'arbres
- Jointure entre arbre
- Trie des arbres (selon des valeurs d'éléments)
- Requêtes peuvent être imbriquées
- Calcul d'agrégats : count,sum,avg,…
- Définition de fonctions

XQuery

- On gère des documents
- Support des fonctions des systèmes documentaires.
  - Prédicat contains : recherche de mots clés

XQuery 1.O

- Spécification dans www.w3.org/TR/xquery/
- Syntaxe et description informelle
- Une description formelle de la sémantique du langage dans : www.w3.org/TR/xquery-semantic/
- Liste de fonctions et opérateurs pour XQuery : www.w3.org/TR/xpath-fonctions/
- Et une liste d'exemple de requêtes :
- www.w3.org/TR/xquery-use-case/

Introduction

- On a besoin d'un modèle de données pour définir le sens (la sémantique) des expressions XQuery:
- Requête:
  - *document("bib.xml")/bib/book[1]/(editor union author)*
- La requête retourne une séquence d'éléments de type *editeur* ou *auteur* qui sont des enfants du premier élément de type *book* dans le document *bib.xml*.

Modèle de données

- Séquence : collection ordonnées d'items, éventuellement vide.
- Un **ITEM** est un **NOEUD** ou une **VALEUR ATOMIQUE**.
- Valeur atomique : instance de type :string,integer,date,etc.
- Chaque nœud et chaque valeur a un **TYPE**.
- Une **VALEUR** est une **séquence ordonnée d'ITEMS**.

Nœuds et valeurs atomiques

- XQuery est un langage fortement typé :
- Types de nœuds XML:
  - document
  - Élément et attribut
  - Texte
  - Commentaire, instruction de traitement et espace de nom
- Types atomiques de XML Schéma:
  - *xsd:string*, *xsd:int*, *xsd:float*

Exemples de valeurs

- *47*
- ‘’toto’’
- <a/>
- <1,2,3>
- (47, <a/>,’’toto’’)
- Un document XML
- (): liste vide
- ERROR : valeur d’erreur

Remarques

- XQuery ne fait pas la distinction entre un item et une séquence de longueur 1
- Il n’existe pas de séquences imbriqués
- Une séquence peut être vide
- Une séquence peut contenir des valeurs hétérogènes
- Les séquences sont triées
- Il n’existe pas de valeurs nulles
- Chaque séquence a un type

Nœud

- Chaque nœud a une identité.
- Les éléments et les attributs sont annotés par un type (XMLSchema).
- Chaque nœud a un type :
  - type XML Schema : *element a {()}*
  - type inconnu (“anySimpleType”)
- Les nœuds sont triés dans l’ordre du document.

Exemple de valeurs et de leur type

- 47 implique *xsd:int*
- *(1, 2, 3)* implique *xsd:int, xsd:int, xsd:int*
- (47,’’toto’’) implique *xsd:int, xsd:string*
- *<a/>* implique element a {()}
- *<a>toto* implique *element a {text}*
- () implique ()
- ERROR implique ERROR

XQUERY

- XQuery est un langage fonctionnel :
  - Une requête XQuery est une composition *d'EXPRESSIONS*.
- Chaque expression a une valeur.
- Les expressions n'ont pas d'effets de bord.
- Les expressions peuvent être composées librement.
- La valeur *ERROR* est propagée (exception).

Expression

- Littéraux (valeurs atomiques): *46*, *“Salut”*
- Valeurs construites: *true()*, *date(“2002-10-23”)*
- Variables: *$x*, *$y*, ...
- Opérateurs arithmétiques: + - * div mod
- Séquences: *(1,2)*, *(1, “toto”, <toto/>)*
- Union, intersection, différence de séquences de *nœuds*

Expression

- Expressions de chemins: XPath 2.0
- Opérateurs de comparaison pour valeurs atomiques, nœuds et séquences
- Expressions logiques: *and or not()*
- Appels de fonctions
- Constructeurs de nœuds
- Boucles (for-let-where-return)
- Tests (if-then-return-else-return)

DTD exemple

<!ELEMENT bib (book)>*
<!ELEMENT book (author+ | editor+),
publisher, price)>
<!ATTLIST book year CDATA #IMPLIED
title CDATA #REQUIRED>
<!ELEMENT author (la, fi)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT la (#PCDATA)>
<!ELEMENT fi (#PCDATA)>
<!ELEMENT affiliation (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT price (#PCDATA)>

Exemple : bib.xml

```
<bib>

<book title="Comprendre XSLT">
<author><la>Auteur1</la><fi>B.</fi></author>
<author><la>Auteur2</la><fi>P.</fi></author>
<publisher>O'Reilly</publisher>
<price>28.95</price>
</book>

<book year="2001" title="Spatial Databases">
<author><la>Auteur2</la><fi>P.</fi></author>
<author><la>Auteur4</la><fi>M.</fi></author>
<author><la>Auteur5</la><fi>A.</fi></author>
<publisher>Morgan Kaufmann Publishers</publisher>
<price>35.00</price>
</book>
...
</bib>
```

Expression simple

- Requête: *1+2*
  - Résultat: *3*
- Requête: *(1, (2, 6), "toto", <toto/>)*
  - Résultat: *1, 2, 6, toto, <toto/>*

Expression de chemins

- Requête: *document("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>

Expression de chemin XPATH 2.0

- Dans XPath 2.0, chaque étape est une expression XQuery (#XPATH 1.0) :
  - *//book[1]/author*
  - *//book[1]/publisher*
  - *//book[1]/(author,publisher)* (impossible dans XPath 1.0)
  - *//book[author/la=''Auteur4'']*
  - *//book[author/la=''Suciu'']/publisher*
  - *//book[position() lt last()]*

Construction de nœuds XML

- Nom et contenu connu:
- Requête:

```
<livre auteur="E. Zola">
   <titre>Germinal</titre>
</livre>
```

- Résultat:
  - *<livre auteur="E.Zola"><titre>Germinal</titre></livre>*

Construction de nœuds XML

Nom connu, contenu calculé: *{ expr }*

- Requête:

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

- Résultat:

```
<?xml version="1.0"?>
<auteurs>
    <la>Auteur1</la>
    <la>Auteur2</la>
    <la>Auteur3</la>
</auteurs>
```

Construction de nœuds XML

- Nom et contenu calculé:
  - *element {* expr-nom *} {* expr-contenu *}*
  - *attribute {* expr-nom *} {* expr-contenu *}*
- Requête:

  element {concat("a_",string(1+4))} {9+3}
- Résultat

  <a_5>12</a_5>

Construction de nœuds XML

- Requête:

```
element livre {
attribute {concat("auteur_",
string(1+2))} {
"toto"
}
}
```

- Résultat:

```
<livre auteur_3="toto"/>
```

Union de séquence de nœuds

- Requête:
  <Livre>
  Les auteurs et le prix du premier livre:
  { document("bib.xml")//book[1]/(author union price) }
  </Livre>
- Résultat:
  <Livre>
  Les auteurs et le prix du premier livre:
  <author><la>Auteur1</la><fi>B.</fi></author>
  <author><la>Auteur2</la><fi>P.</fi></author>
  <price>28.95</price>
  </Livre>
- Les trois opérateurs *union*, *intersect* et *except* éliminent les duplicats.

Différence de séquence de nœuds

- Requête:

  <livre>

  Tous les sous-elements sauf les auteurs:

  { document("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>

Construction de séquence

- Requête:

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

  <livre>
  Le prix suivi des auteurs:
  <price>28.95</price>
  <author><la>Auteur1</la><fi>B.</fi></author>
  <author><la>Auteur2</la><fi>P.</fi></author>
  </livre>
- On a changé l'ordre des nœuds (#union)

Opérateurs arithmétique

- opérateurs unaires : *+ -*
- opérateurs binaires : *+ - * div mod*
- Requête:

  *document("bib.xml")//book[1]/price * 1.2*
- Résultat:

  3.474E1

Transformation nœud valeur

- Requête:

  <auteurs>
  { document("bib.xml")//book[2]/author/la }
  <noms>
  { document("bib.xml")//book[2]/author/xf:string(la) </noms>
  </auteurs>

- Résultat:

  <auteurs>
  <la>Auteur3</la>
  <la>Auteur4</la>
  <la>Auteur5</la>
  <noms>Auteur3 Auteur4 Auteur5</noms>
  </auteurs>

Expression de comparaisons

- Comparaison de valeurs atomiques:
  - *eq ne gt ge lt le*
- Comparaison de séquences de valeurs avec quantificateur existentiel:
  - *= != > >= < <=*
- Comparaison de nœuds:
  - *is isnot*
- Comparaison de nœuds par rapport à leur position dans le document:
  - *<< (avant) >> (après)*

Comparaison de valeurs atomique

- Requête:

  distinct-values(document("bib.xml")//book/author[la eq "Auteur4"])
- Résultat:

  <author><la>Auteur4</la><fi>P.</fi></author>

- Requête:

  document("bib.xml")//book[author/la eq "Auteur4"]
- Résultat:

  ERROR

- L'expression *author/la* dans le prédicat ne retourne pas une valeur atomique mais une *séquence de valeurs*.

Comparaison de séquence

- Comparaison de séquences : S1=S2 s'il existe au moins un élément dans S1 qui est égal à un élément dans s2
- Requête :

  document("bib.xml")//book[author/la = "Auteur4"]
- Résultat:

  <book year="2001"
  title="Spatial Databases">
  <author><la>Auteur3</la><fi>P.</fi></author>
  <author><la>Auteur4</la><fi>M.</fi></author>
  <author><la>Auteur5</la><fi>A.</fi></author>
  <publisher>Morgan Kaufmann Publishers</publisher>
  <price>35.00</price>
  </book>

Comparaison de nœuds

- Comparaison de l'identité de deux nœuds :
- Requête:

  document("bib.xml")//book[author[2] is author[last()]]
- Résultat:

  <book title="Comprendre XSLT">

  <author><la>Auteur1</la><fi>B.</fi></author>

  <author><la>Auteur2</la><fi>P.</fi></author>

  <publisher>O'Reilly</publisher>

  <price>28.95</price>

  </book>

Comparaison par la position

- Comparaison de la position de deux nœuds :n1<<n2(n1>>n2) si n1 apparaît avant (après) n2 dans le document

- Requête:

  document("bib.xml")//book[author[la="Abiteboul"] <<author[la="Suciu"]]

- Résultat:

  <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>

Boucles : for-return

- La clause *let* **$var** *:=* **exp** affecte la variable **$var** avec la séquence “entière” retournée par **exp**.
- Requête:

```
for $b in document("bib.xml")//book[1]
let $a1 := $b/author
return <livre nb_auteurs="{count($a1)}">
{ $a1 }
</livre>
```

- Résultat:

```
<livre nb_auteurs="2">
<author><la>Auteur1</la><fi>B.</fi></author>
<author><la>Auteur2</la><fi>P.</fi></author>
</livre>
```

Boucles : for-where-return

- La clause *where* **exp** permet de filtrer le résultat par rapport au résultat booléen de l'expression **exp** (= prédicat dans l'expression de chemin).
- Requête:

```
for $a in document("bib.xml")//author
where $a/la eq "Abiteboul"
Return $a
```

- Résultat:

```
<?xml version="1.0"?>
<author>
<la>Abiteboul</la>
<fi>S.</fi>
</author>
```

Tests : if-then-else

- Requête:

```
<livres>
{   for $b in document("bib.xml")//book
    where $b/author/la = "Auteur2"
    return
       If ($b/author[1]/la eq "Auteur2")
       then <livre prem="true"> {$b/@title} </livre>
                    else <livre> {$b/@title} </livre>
}
</livres>
```

- Résultat:

```
<?xml version="1.0"?>
<livres>
    <livre title="Comprendre XSLT"/>
    <livre prem="true" title="Spatial Databases"/>
</livres>
```

Quantification existentielle

- some *$var* in *expr1* satisfies *expr2*
  - Retourne true s'il existe au moins un nœud retourné par l'expression *expr1* qui satisfait l'expression *expr2*.
- Requête:

  for $b in document("bib.xml")//book

  where some $a in $b/author satisfies $a/la = "Auteur1"

  return <livre>{$b/@title}</livre>
- Résultat:

  <livre title="Comprendre XSLT"/>

Quantification Universelle

- every *$var* in *expr1* satisfies *expr2*
  - Retourne true ssi tous les nœuds retournés par l'expression *expr1* satisfont l'expression *expr2*
- Requête:

  for $a in document("bib.xml")//author

  where every $b

  in document("bib.xml")//book[author/la = $a/la]

  satisfies $b/publisher="O'Reilly"

  return $a
- Résultat:

  <author><la>Auteur1</la><fi>B.</fi></author>

Construction d'éléments

- Requête:

```
for $b in document("bib.xml")//book[2]
return
element livre
{ element annee { string($b/@year) },
for $e in $b/@*
where name($e) != "year"
return element {name($e)} {string($e)} }
```

- Résultat:

```
<?xml version="1.0"?>
<livre>
<annee>2001</annee>
<title>Spatial Databases</title>
</livre>
```

Construction d’attributs

- Requête:

  <livres>
  { for $t in document("bib.xml")//book/@title
  return element livre {attribute titre {string($t)}}
  }
  </livres>

- Résultat:

  <livres>
  <livre titre="Comprendre XSLT"/>
  <livre titre="Spatial Databases"/>
  <livre titre="Data on the Web"/>
  </livres>

Trier avec sortBy

- **Expr1** sortby **Expr2** (ascending | descending)?
  - Trier les éléments de la séquence retournée par l'expression *Expr1* par les valeurs retournées par **Expr2**.
- Requête:

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

- Résultat:

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

jointure

Fichier d'adresses: *"addr.xml"*

```
<addresses>
    <person>
        <name>Auteur1</name>
        <country>France</country>
        <institution>CNAM</institution>
    </person>
    <person>
        <name>Auteur2</name>
        <country>France</country>
        <institution>CNAM</institution>
    </person>
    <person>
        <name>Auteur3</name>
        <country>Germany</country>
        <institution>FU Berlin</institution>
    </person>
</addresses>
```

Jointure : requête

- Requête:

```
for $b in document("bib.xml")//book
return element livre {
attribute titre {$b/@title},
for $a in $b/author
return element auteur {
attribute nom {$a/la},
for $p in document("addr.xml")//person
where $a/la = $p/name
return attribute institut {$p/institution} } }
```

Jointure : résultat

- Résultat:

```
<livre titre="Comprendre XSLT">
        <auteur nom= " Auteur1" institut="CNAM"/>
        <auteur nom= " Auteur2"/>
</livre>,
<livre titre="Spatial Databases">
        <auteur nom= " Auteur2"/>
        <auteur nom= " Auteur3" institut="CNAM"/>
        <auteur nom= " Auteur4" institut="FU Berlin"/>
</livre>,
<livre titre="Data on the Web">
        <auteur nom="Abiteboul"/>
        <auteur nom="Buneman"/>
        <auteur nom="Suciu"/>
</livre>
```

Fonctions et opérateurs

- Les nombreux fonctions et opérateurs permettent :
  - l'accès au type et le nom d'un nœud
  - la construction, comparaison et la transformation de valeurs
  - l'agrégation des valeurs d'une séquence
  - la transformation du type d'une valeur (cast)

Fonctions et opérateurs

- Les fonctions et opérateurs sont
  - typées (XML schema) et
  - manipulent des séquences et des valeurs typées (XML schema) : entiers, chaînes de caractères, dates, . . .

Exemple : fonction avg

- Requête:

  for $p in distinct-values(document("bib.xml")//publisher)

  let $l := document("bib.xml")//book[publisher = $p]

  return element publisher {

  attribute name {string($p)},

  attribute avg_price { avg($l/data(price)) }}

- Résultat :

  <publisher name="O'Reilly" avg_price="28.95"/>,

  <publisher name="Morgan Kaufmann Publishers" avg_price="37.48"/>

Fcts séquences

- distinct-nodes(item* *Seq1*) => item* :
  - élimination de duplicats en comparant l'identité des nœuds
- distinct-values(item* *Seq1*) => item* :
  - élimination de duplicats en comparant la valeur
- index-of(item* *Seq*, item *Search*) =>
  - unsignedInt* : retourne les positions des nœuds ou des valeurs dont la valeur est identique au Paramètre search

Exemple : index-of

- Requête:

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

- Résultat:

```
<?xml version="1.0"?>
<books>
<book title="Comprendre XSLT" no="1"/>
<book title="Spatial Databases" no="2"/>
<book title="Data on the Web" no="3"/>
</books>
```

Déf. de fonctions

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

define function NombreAuteurs(book $b)

returns xsd:integer {

return count($b/author)

}

- Le résultat est de type xsd:int.