
Interrogation de documents XML

Interrogation de documents XML

Comment interroger des documents XML ?

Solutions :

- SQL : il faut stocker XML dans une BD relationnel
- XPath : extraction de fragments d'arbres
- XQuery : vrai langage de requêtes pour XML
- XSLT : extraction + transformation (règles)

XPath : Résumé

XPath est un langage pour extraire des noeuds dans un arbre XML :

- On navigue dans l'arbre grâce à des axes de navigation.
- Un chemin de navigation est une séquence d'étapes.
- Dans chaque étape on choisit un axe, un filtre et éventuellement des prédicats.
- Le résultat d'une étape (d'une séquence d'étapes) est une séquence de noeuds.

XSLT : Résumé

XPath est un langage pour transformer des arbres XML :

- On définit des règles de transformation qui transforme un fragment d'arbre en un autre fragment (copie).
- Les fragments à transformer sont choisis par des expressions XPath.
- Les règles de transformations à appliquer à un fragment est choisie par une expression XPath.
- Le résultat d'une transformation est un document XML.

Exemple: Requête SQL/XML avec extraction XPath

```
SQL> select P.PODOCUMENT.extract(  
2         '/PurchaseOrder//User').getClobVal()  
3 from PURCHASEORDER P  
4 where P.PODOCUMENT.extract(  
5         '/PurchaseOrder//Date/text()'  
6         ).getStringVal() = '12/11/02'
```

```
P.PODOCUMENT.Extract('/PurchaseOrder//User').GETClobVal()
```

```
-----
```

```
<User>KING</User>
```

XQuery

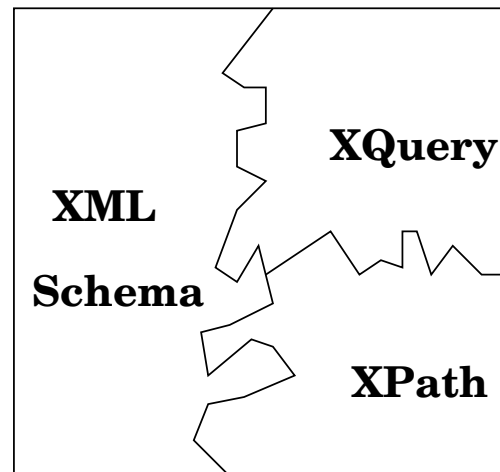
Langages de requêtes pour XML

Un petit historique :

- Langages de requêtes pour données semi-structurées : POQL de l'INRIA (1996), UnQL de Penn. Univ. (1996), Lorel de Stanford Univ. (1997)
- 1998 : Workshop **Query Languages'98 (QL'98)**
- Langages de requêtes pour XML : XOQL (Xyleme), XML-QL, XQL, Lore, ...
- XQuery : W3C Working Draft 02 May 2003

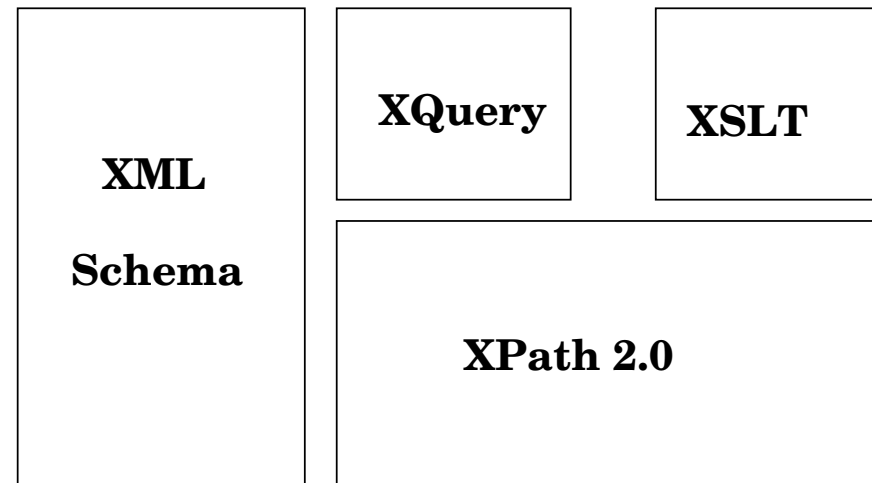
La place de XQuery dans le monde XML

Modèles et Langages:



(Don Chamberlin)

Groupes de travail:



Besoin d'un modèle de données

On a besoin d'un modèle de données pour définir le sens (la sémantique) des expressions XQuery:

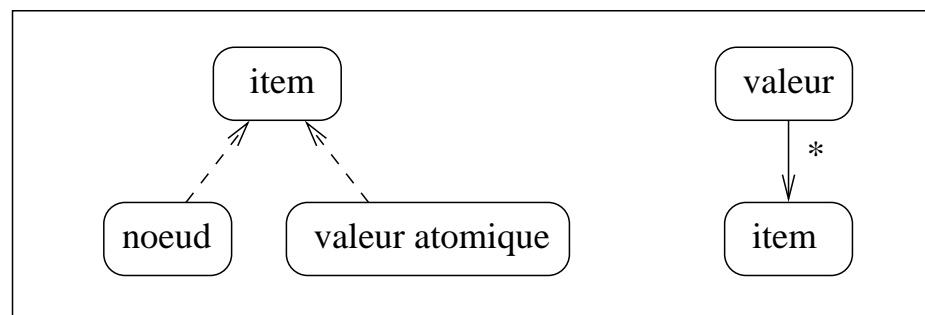
Requête:

```
document("bib.xml")/bib/book[1]/(editor union author)
```

La requête retourne une *séquence d'éléments* de type *(editeur | auteur)* qui sont des enfants du premier élément de type *book* dans le document *bib.xml*.

Le modèle de données de XQuery

- Une **valeur** est une *séquence ordonnée d'items*.
- Un **item** est un **noeud** ou une **valeur atomique**.
- Chaque noeud et chaque valeur a un **type**.



Valeurs/Séquences

- Pas de distinction entre un item et une séquence de longueur 1 : $47 = (47)$
- Une séquence peut contenir des valeurs hétérogènes :
 $(1, \text{' 'toto' '}, <toto/>)$
- Pas de séquences imbriqués :
 $(1, (2, 6), \text{"toto"}, <toto/>) =$
 $(1, 2, 6, \text{' 'toto' '}, <toto/>)$
- Une séquence peut être vide : $()$
- Les séquences sont triées : $(1, 2) \neq (2, 1)$.

Expressions XQuery

- Une requête XQuery est une **composition d'expressions**.
- Chaque expression a une valeur ou retourne une erreur.
- Les expressions n'ont pas d'effets de bord (par exemple, pas de mise-à-jour).

Expressions (requêtes) simples :

- Valeurs atomiques: `46`, `''Salut''`
- Valeurs construites: `true()`, `date(''2002-10-23'')`

Expressions Complexes

- Expressions de chemins (XPath 2.0) : *FILM//ACTEUR*
- Expressions FLWR (for-let-where-return)
- Tests (if-then-return-else-return)
- Fonctions
 - Racines : *input*, *collection("url")*, *doc("url")*
 - Fonctions prédéfinies : **XQuery 1.0 and XPath 2.0 Functions and Operators**
 - Fonctions utilisateurs

Opérations

- Opérateurs arithmétiques: $1+2$, $3 \cdot 4-6 \cdot 5$, $\$y \bmod 2$
- Opérations sur les séquences :
 - concaténation : $1, A$
 - union, intersection, différence de *séquences de noeuds*
- Opérateurs de comparaison pour valeurs atomiques, noeuds et séquences
- Opérations booléennes: $\$x=2 \text{ and } \$y=4 \text{ or not}(\$z)$

DTD Example: *bib.dtd*

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

Document Exemple: *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 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>
```

Expressions de chemins

Chaque étape est une expression XQuery (XPath 2.0):

- `doc("bib.xml")/bib/book/author`
- `doc("bib.xml")/bib//book[1]/publisher`
- `doc("bib.xml")//book/(author, publisher):
résultat?`
- `doc("bib.xml")/(descendant::author,
descendant::publisher): résultat?`
- `doc("bib.xml")//book/(@title union publisher)`
- `doc("bib.xml")//book[position() lt last()]`

Construction de noeuds XML

Le nom est connu, le contenu est calculé par une expression *expr*

Requête:

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

Résultat:

```
<?xml version="1.0"?>
<auteurs>
  <la>Rigaux</la>
  <la>Scholl</la>
  <la>Voisard</la>
</auteurs>
```

Construction de noeuds XML

Le nom et le contenu sont calculés:

- *element* { *expr-nom* } { *expr-contenu* }
- *attribute* { *expr-nom* } { *expr-contenu* }

Requête:

```

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

```

Résultat:

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

Différence de séquences de noeuds

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>
```

Concaténation de séquences

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>Amann</la><fi>B.</fi></author>  
  <author><la>Rigaux</la><fi>P.</fi></author>  
</livre>
```

On a changé l'ordre des noeuds (\neq union)

Transformation noeud → valeur

Requête:

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

Résultat:

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

(xf:string est une fonction prédéfinie).

Comparaison de valeurs atomiques

Requête:

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

Résultat:

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

Requête:

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

Résultat:

```
ERROR
```

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

Comparaison de séquences

Comparaison de séquences : $s1 = s2$ s'il existe au moins un élément dans $s1$ qui est égal à un élément dans $s2$.

Requête:

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

Résultat:

2

Comparaisons de noeuds

Comparaison de l'identité de deux noeuds :

$n1 \text{ is } n2 \Leftrightarrow n1 \text{ est identique à } n2.$

Requête:

```
document("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>
```

Comparaisons par la position

Comparaison de la position de deux noeuds: $n1 \ll n2$ ($n1 \ll n2$)
 $\Leftrightarrow n1$ apparaît avant (après) $n2$ dans le document.

Requête:

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

Résultat:

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

Affectation de variables: **for** et **let**

La clause *for* **\$var** *in* **exp** affecte la variable **\$var** successivement avec chaque item dans la séquence retournée par **exp**.

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>Amann</la><fi>B.</fi></author>
  <author><la>Rigaux</la><fi>P.</fi></author>
</livre>
```

Sélection: where

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:

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

Résultat:

```
<?xml version="1.0"?>
<livre title="Data on the Web"/>
```

Tests: if-then-else

Requête:

```
<livres>
  { for $b in document("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:

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

Quantification

some $\$var$ in $expr1$ satisfies $expr2 \Leftrightarrow$ *il existe au moins un* nœud retourné par l'expression $expr1$ qui satisfait l'expression $expr2$.

every $\$var$ in $expr$ satisfies $expr \Leftrightarrow$ *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 $b/publisher="Morgan Kaufmann Publishers"
return string($a/la)
```

Résultat:

Scholl, Voisard, Abiteboul, Buneman, Suciu

Trier avec `sort by`

Expr1 `sort by` **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, $b/@year } </livre>
  sort by(@year) }
</livres>
```

Résultat:

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

Jointure

Fichier d'adresses:

```
<addresses>
  <person>
    <name>Amann</name>
    <country>France</country>
    <institution>CNAM</institution>
  </person>
  <person>
    <name>Scholl</name>
    <country>France</country>
    <institution>CNAM</institution>
  </person>
  <person>
    <name>Voisard</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="Amann" institut="CNAM" />
  <auteur nom="Rigaux" />
</livre>,
<livre titre="Spatial Databases" >
  <auteur nom="Rigaux" />
  <auteur nom="Scholl" institut="CNAM" />
  <auteur nom="Voisard" 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 noeud,
- la construction, la comparaison et la transformation de valeurs,
- l'agrégation des valeurs d'une séquence.

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 { xf:avg($l/price) }}
```

Résultat :

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

Exemple: index-of

Requête:

```
<books> {  
  let $b1 := document("bib.xml")//book  
  for $b in $b1  
  return <book> {  
    $b/@title,  
    attribute no {index-of($b1, $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>
```

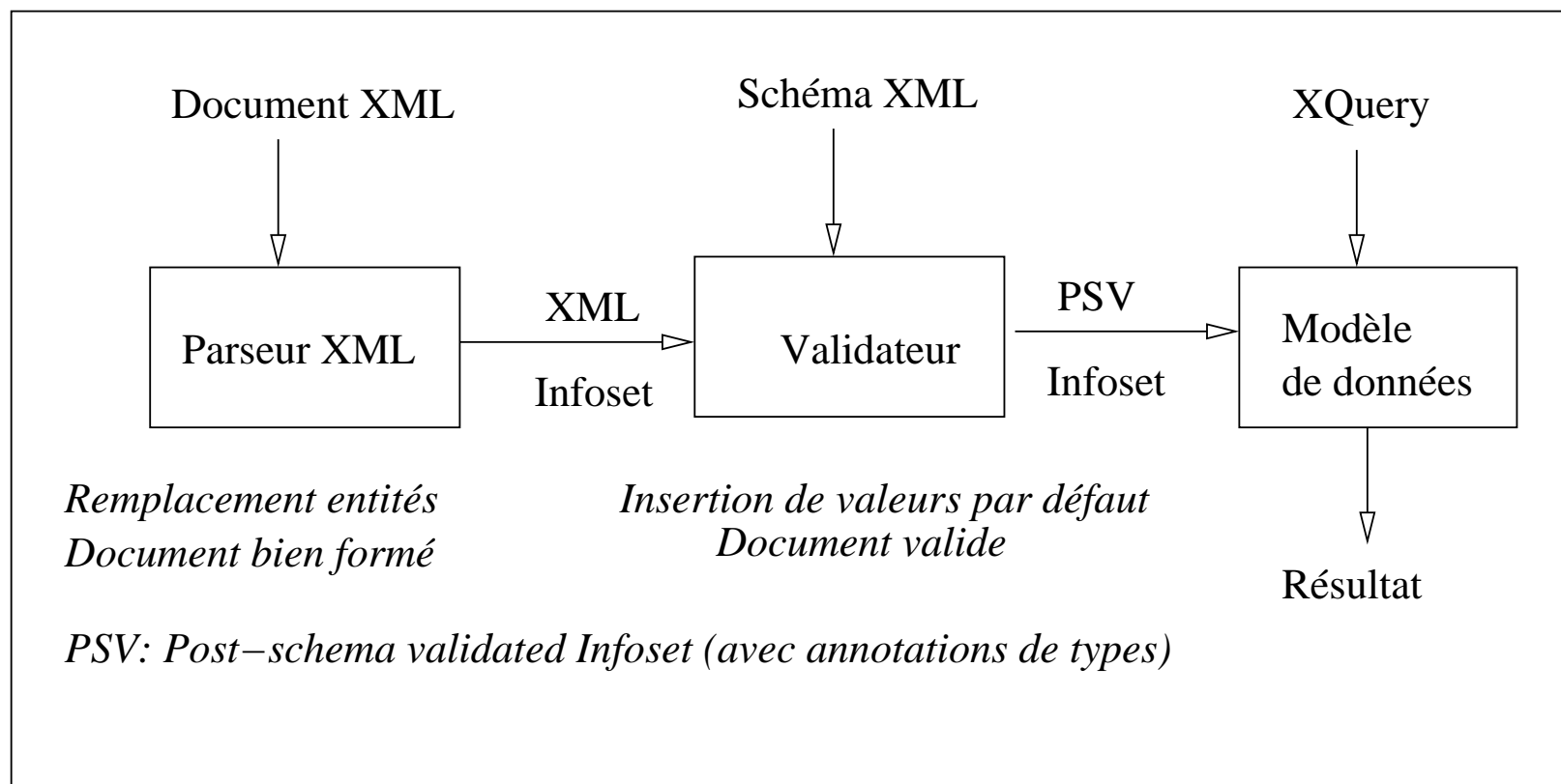
Définition de Fonctions

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`.

Modèle abstrait d'un traitement XML



Modèle abstrait d'un traitement XML

Les éléments et les attributs sont annotés par un type (XMLSchema).

Chaque noeud a un type :

- type XML Schema : *element a { () }*
- type inconnu (“anySimpleType”)

Exemples de valeurs et de leur type

- $47 \Rightarrow \text{xsd:int}$
- $(1, 2, 3) \Rightarrow \text{xsd:int}, \text{xsd:int}, \text{xsd:int}$
- $(47, \text{'toto'}) \Rightarrow \text{xsd:int}, \text{xsd:string}$
- $\langle a \rangle \Rightarrow \text{element } a \{ () \}$
- $\langle a \rangle \text{toto} \langle /a \rangle \Rightarrow \text{element } a \{ \text{text} \}$
- $() \Rightarrow ()$
- $\text{ERROR} \Rightarrow \text{ERROR}$

Plus d'informations...

- Site du W3C sur XQuery: <http://www.w3.org/XML/Query>
- P. Wadler, XQuery: a typed functional language for querying XML
- J. Siméon et P. Wadler, The Essence of XML
- Galax: db.bell-labs.com/galax/
- GMD IPSI XQuery