

Module MLBDA
Master Informatique
Spécialité DAC

COURS 7 – XQUERY

Plan

Concepts des langages de requêtes XML

- Motivations
- Caractéristiques

XQuery

Exemples de requêtes

Manipulation de données XML

Document XML

- Structure flexible pas toujours connue à l'avance
- Peut varier selon les documents

Applications XML

- Formats de données standards
 - MathML, SVG, GeoML, RDF, ...
- Echange de données
 - Tables, XLS, CSV → documents XML → Tables, XSL, CSV
- Transformation de données
- Stockage, intégration et **interrogation de données**
 - Bases de données XML
 - XQuery

Interroger des données XML

Bases de données XML

- Collection de documents XML
- Importation et exportation de documents
- Interrogation des collections : Xpath / XQuery
- Optimisation, indexation, ...

Interroger des documents XML

- XPath : sélectionner des fragments XML dans un document
- *XPath est insuffisant pour interroger des collections*
 - Interroger plusieurs documents à la fois
 - Combiner des fragments
 - Créer des nouveaux documents / fragments

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

Extraction / filtrage de fragments dans des documents

- Navigation dans un arbre → XPath

Combiner des ensembles de fragments :

- Logiques : and, or, négation, quantification
- Ensembles / listes : union, différence, concaténation, tri
- Algébriques : jointure, projection

Construire des nouveaux fragments :

- Créer une nouvelle structure d'arbre
- Imbrication: ajouter des niveaux

Types et fonctions :

- Agrégation : somme, moyenne,
- 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 bib.dtd

<!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 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 Unix</itle>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  <book year="1994">
    <title>Database theory</title>
    <author><last>Abiteboul</last><first>Serge</first></author>
    <author><last>Hull</last><first>Rick</first></author>
    <author><last>Vianu</last><first>Victor</first></author>
  </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 TV</title>
    <editor>
      <last>Gerbarg</last><first>Darcy</first>
      <affiliation>CITI</affiliation>
    </editor>
    <publisher>Kluwer Academic Publishers</publisher>
    <price>129.95</price>
  </book>
</bib>
```

Qu'est-ce qu'on veut faire avec des requêtes ?(1)

Sélection et extraction :

- tous les couples (\$titre, \$auteur) des ouvrages publiés par Eyrolles depuis 2000
- l'arbre XML de la base est « mis à plat » : séquence de (\$titre, \$auteur)

Reconstruction d'arbres XML :

- générer un arbre respectant la DTD originale
- réorganiser la structure par rapport à une DTD nouvelle : livres par auteur
 <!ELEMENT author (book+)>
 <!ELEMENT book (title, author+,publisher?, price?)>

Combiner plusieurs sources de données :

- joindre la base des **livres** et celle des **adresses** pour avoir les livres et les adresses des auteurs et éditeurs.

Trier les résultats :

- titre des livres par ordre alphabétique

Qu'est-ce qu'on veut faire avec des requêtes ?(2)

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)

Filtres d'extraction ensemblistes :

- extraire pour chaque livre tous les enfants **sauf** le prix

Accès approximatif par le contenu (« plein texte »):

- retrouver les sections ou les chapitres traitant de bases de données (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

XQuery est un **langage fonctionnelle** sur des **séquences** :

- valeur XQuery :
 - **séquence** de fragments/noeuds XML et de valeurs (entiers, chaîne des caractères,)
- expression XQuery :
 - **composition de fonctions sur des séquences**
 - retourne une valeur ou une erreur
 - pas d'effets de bord
 - fonctions utilisateurs (Turing Complet)

Expressions XQuery

expressions simples

expressions de chemins

comparaisons

construction d'éléments

expressions FLWOR

conditions

quantificateurs

types de données

Modèle de données Xquery (XDM) : items et valeurs

Items:

- Types XML Schema : entier, string, booléen, date, XML
- valeurs atomiques : 32, "coucou", true(), false(), integer("12"), date ("01/01/2017")
- fragments / nœuds XML:
<livre><auteur>Ullman</auteur></livre>

Opérateurs items :

- logiques : **and**, **or**, **not**
- arithmétiques : **+**, **-**, *****, **div**, **mod**
- comparaison de valeurs : **eq**, **lt**, **gt**
- comparaison de position : **is**, **<<**, **>>**

Fonctions : contains(), substring(), abs()...

Valeurs :

◦ **séquence d'items**

- (10, 1, 2, 3, 4)
- (3,false(),true())
- (1, "toto", <toto/>, <a>1)
- Séquence vide: ()

Opérateurs séquences:

- concaténation : **,**
- union: **union**
- différence : **except**
- intersection : **intersect**
- comparaison : **=**, **<**, **>**, **!=**, **<=**, **>=**

Fonctions : count(), last(), first()...

Expressions XQuery

Expressions simples :

- items et variables
- « 32 » est une expression qui retourne la valeur 32
- « 32 » → 32
- \$a, \$book, \$x : variables affectés à une valeur (séquence)

Expression sur les séquences :

- séquence d'expressions qui retourne une valeur (séquence)

→ **requête XQuery**

L'opérateur de concaténation « , » évalue chacune des expressions et la remplace par la séquence résultat dans l'ordre.

Résultat : une séquence d'items

- « (10, (1, 2), 3, 4) » → (10, 1, 2, 3, 4)
- « (1+2, 2=3, 3>2) » → (3, false(), true())
- « (<a>1+2) » → (<a>1+2)
- « 47 » → (47)

Autres opérateurs:

- « (10, 1, 2, 3, 4) **except** (1+1) » → (10, 1, 3, 4)
- « (10, (1,2), ()) **union** (3,(4)) » → (10,1,2,3,4)
- « (1+2, 2+3, 3*2) **intersect** (4) » → ()
- « (1+2, 2=3, 3>2) **intersect** (true()) » → (true())

Expressions de chemin

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

- séquences des éléments de type `author` avec le `nom` 'martin'

XPath sert à extraire des séquences de fragments / nœuds XML à partir de documents ou d'autres fragments.

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

Le contexte d'une expression Xpath est:

- Un document XML: `document("bib.xml")//book/author`
- Une collection de documents XML : `collection("archive")//book/author`
- Une variable: `$alice/last`
- Défini par défaut : `//book/author`

Expressions de chemin et opérateurs de séquences

Expressions XPath 1.0: retourne une séquence

Expression XPath 1.1: XPath 1.0 + opérateurs séquences

`//book, //author` → séquences des éléments `book` suivi des éléments `auteur`

`//book union //author` → séquences des éléments `book` et `auteur` dans l'ordre du document

`//* except //*[*]` → séquence de tous les nœuds sans enfants (feuilles)

`//book/(* except (author, editor))` → séquence des enfants de `book` sauf les éléments `author` et `editor`

Comparaisons d'items

Les opérateurs **eq**, **ne**, **lt**, **le**, **gt**, **ge** permettent de comparer des **item** :

- `$book1/title eq 'Data on the Web'` → `true()` ssi `$book1` a exactement un élément fils `title` dont la valeur est la chaîne "Data on the Web"
- `$book1/price gt 100` → `true()` ssi il y a un seul prix et sa valeur est > 100
- `1 eq 1` → `true()`, `1 eq 3` → `false()`, `1 eq (2)` → `false()`, `(1+1) eq (2)` → `true()`,
- `(1,2) eq (3)` → **erreur**

Comparaisons de noeuds XML

Les nœuds (éléments) sont comparés par leur **valeur textuelle**:

- `<a>5 eq <a>5` → `true()`
- `<last>Bob</last> eq <first>Bob</first>` → `true()`

Utiliser la fonction `deep-equal` pour comparer **la structure et la valeur**:

- `deep-equal((<a>1),(<a>1))` → `false()`
- `deep-equal((<a>1),(<a>1))` → `true()`

Comparaisons de nœuds XML

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

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

is renvoie true() si s'il s'agit du même nœud à gauche et à droite;

- « `<a>5 is <a>5` » → **false()**
- « `//book[year= "2002"] is //book[title= "Data on the Web"]` » → **true()** si Data on the Web est le seul livre de 2002

<< (resp. **>>**) renvoie true() si le nœud de gauche précède (resp. succède) le nœud de droite dans l'ordre du document:

- « `//produits[ref="123"] << //produits[ref="456"]` » → **true()** si le produit 123 est avant le produit 456 dans le document.

Comparaisons de valeurs (séquences)

Les opérateurs $=$, \neq , $<$, $>$, \leq , \geq permettent de comparer des séquences d'items (valeurs).

$S \text{ op } S'$ retourne `true()` s'il existe un item i dans S et un item i' dans S' tels que $i \text{ op } i'$ est vrai.

Exemples:

- $(1,2,3)=(1,2,3)$ \rightarrow `true()`
- $(1,2,3)>(1,2,3)$ \rightarrow `true()`
- $\text{not}((1,2,3)<(1,2,3))$ \rightarrow `false()`
- $\text{not}((1,2,3)>(1,2,3))$ \rightarrow `false()`
- $(1,2,3)=(1+1,4,5)$ \rightarrow `true()`
- $(1,2,3)=(1,3+7,5)$ \rightarrow `true()`
- $(1,2,3)=(3+7,5)$ \rightarrow `false()`

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.

Requête:

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



Crée un nouvel élément **book**, avec un **titre** et un **auteur**, un **nom** et un **prenom**.

Résultat:

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

Construction d'éléments

Insertion de résultats d'expressions XQuery avec { <exp> }

\$b :

```
<book isbn="isbn-1234567890">  
  <titre>100 ans de solitude</titre>  
</book>
```

Requête:

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




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 {		<book isbn="isbn-1234567890">
attribute isbn {"isbn-1234567890" },		<titre>100 ans de solitude</titre>
element titre {"100 ans de solitude"},		<auteur>
element auteur {		<prenom>Gabriel</prenom>
element first {"Gabriel"},		<nom>Garcia Marquez</nom>
element last {"Garcia Marquez"		</auteur>
}		</book>
}		
}		

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 séquence d'éléments **book** à la variable \$b
- **where** filtre les couples (\$p,\$b) où \$b contient plus que 100 éléments (livres)
- **return** construit pour chaque couple la valeur résultat.

Résultat final : séquence d'éléments **publisher**

for et let : Affectation de variables

-for \$var in <exp>

- Itération sur les items de la séquence générée par <exp> et, à chaque itération, affectation à la variable \$var de l'item courant (boucle)

-let \$var:= <exp>

- Affecte à la variable \$var de la séquence générée 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 : séquence avec un élément
out:

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

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

Résultat : séquence de **trois** éléments
out:

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

where : applications de filtres

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 la sequence des titres des livres dont Ullman est auteur

return : construction du résultat

La clause **return** est évaluée une fois pour chaque itération et le résultat est une séquence d'items.

L'ordre est déterminé par l'ordre du documents XML et les clauses **for** et **let**.

```
for $e in document("doc.xml")//employees  
return ($e/first, $e/last)
```

→ séquence qui alterne les éléments **first** et **last** dans l'ordre du document

```
for $e in document("doc.xml")//employees  
return $e/first, $e/last
```

→ le résultat est une séquence d'éléments **first** suivi du résultat de l'expression **\$e/last**

→ **Erreur** : la variable **\$e** n'est pas liée

order by et return : tri

La clause **order by** permet de réordonner les n-uplets dans l'ordre croissant (**ascending**) et décroissant (**descending**).

```
for $e in  
document("doc.xml")//employees  
order by $e/salary descending  
return $e/name
```

→ séquence avec un élément **name** pour chaque employé

```
for $b in document('bib.xml')//book  
order by $b/price, b/title  
return $b
```

→ séquence d'éléments **book** triés par le prix et le titre.

if-then-else : construction conditionnelle

Produire des résultats « conditionnelles »:

```
<books>
{ for $x in //book
  return
    <book>{ $x/title } est {
      if ($x/@year > 1999)
        then "récent"
        else "ancien" }
    </book> }
</books>
```

```
<books>
  <book>
    <title>TCP/IP Illustrated</title> est ancien</book>
  <book>
    <title>Advanced Programming in the Unix environment</title> est
ancien</book>
  <book>
    <title>Data on the Web</title> est récent</book>
  <book>
    <title>Database theory</title> est ancien</book>
  <book>
    <title>The Economics of Technology and Content for Digital
TV</title> est ancien</book>
</books>
```

some et exists : quantificateurs existentiels et universelles

some \$x in <expr1> satisfies
<expr2> → true() s'il existe AU MOINS UN nœud dans la séquence renvoyée par <expr1> qui satisfait <expr2>

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

→ true() si au moins un livre a un attribut dont la valeur est supérieure à 2003.

every \$x in <expr1> satisfies
<expr2> → true() si TOUS les nœuds dans la séquence renvoyée par <expr1> satisfont <expr2>

every \$b in
 document("bib.xml")//book
satisfies \$b/@year

→ true() si tous les livres ont un attribut year.

Exemple

On suppose qu'un élément **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='LIP6'  
return $b/title
```

→ titre des livres dont tous les éditeurs sont affiliés au LIP6.

Types

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

<item> instance of <type> → true() si la valeur du premier opérande est du type du deuxième opérande:

3 instance of xs:integer → true()
<x>3</x> instance of element() → true()

typeswitch .. cause ..default ..: branchement en fonction du type

```
typeswitch($e)
case $e as element(book) return $e/editor
case $e as element(author) return $e/last
default return ()
```

cast : « forcer » 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")//book
  where $b/publisher = "Addison-Wesley"
  and $b/@year > 1991
  return <livre annee= "{$b/@year}">
    {$b/title}
  </livre>
}
</bib>
```

Résultat :

```
<bib>
  <livre annee= "1994">
    <title> TCP/IP Illustrated </title>
  </livre>
  <livre annee= "1992">
    <title>Advanced Programming in Unix</title>
  </livre>
</bib>
```

Exemple (2)

Liste de toutes les paires (**title**, **last**),
chaque paire étant contenue dans
un élément **result**.

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

```
<results>  
  <result>  
    <title>TCP/IP Illustrated</title>  
    <last>Stevens</last>  
  </result>  
  <result>  
    <title>Advanced Programming in Unix </title>  
    <last>Stevens</last>  
  </result>  
  <result>  
    <title>Data on the Web</title>  
    <last>Abiteboul</last>  
  </result>  
  <result>  
    <title>Data on the Web</title>  
    <last>Buneman</last>  
  </result>  
  ...  
</results>
```

Exemple (3)

Une entrée par livre avec le titre et tous les noms d'auteurs:

```
<results> {  
  for $b in //book  
  return  
    <result>  
      { $b/title}  
      { $b/author/last }  
    </result>  
}  
</results>
```

```
<results>  
  <result>  
    <title>TCP/IP Illustrated</title>  
    <last>Stevens</last>  
  </result>  
  <result>  
    <title>Advanced Programming in Unix </title>  
    <last>Stevens</last>  
  </result>  
  <result>  
    <title>Data on the Web</title>  
    <last>Abiteboul</last>  
    <last>Buneman</last>  
    <last>Suciu</last>  
  ....  
  </result>
```

distinct-values() : élimination de doublons par valeur

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="claudes" a="50" v="Lyon"/>
</c>
```

`distinct-values(//p/@a)` retourne la sequence de valeurs de l'attribute @a sans doublons: (60, 30, 50)

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

Réponse

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

```
<resultat>  
  <age a="60">  
    <p n="paul" a="60" v="Paris"/>  
    <p n="jean" a="60" v="Nice"/>  
  </age>  
  <age a="30">  
    <p n="martin" a="30" v="Paris"/>  
  </age>  
  <age a="50">  
    <p n="claudio" a="50" v="Lyon"/>  
  </age>  
</resultat>
```

Combien d'items <proche> renvoie la requête ?

```
<?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="claudio" a="50" v="Lyon"/>
```

```
</c>
```

```
for $p1 in //p,
```

```
$p2 in //p
```

```
where $p2/@v=$p1/@v
```

```
return
```

```
<proche>
```

```
<p1> {$p1/@n} </p1>
```

```
<p2> {$p2/@n} </p2>
```

```
</proche>
```


Réponse

```
for $p1 in //p, $p2 in //p
where $p2/@v=$p1/@v
return
```

```
<proche>
  <p1> {$p1/@n} </p1>
  <p2> {$p2/@n} </p2>
</proche>
```

4+2=6 items

```
<proche>
  <p1 n="paul"/>
  <p2 n="paul"/>
</proche>,
<proche>
  <p1 n="paul"/>
  <p2 n="martin"/>
</proche>,
<proche>
  <p1 n="paul"/>
  <p2 n="paul"/>
</proche>, ...
```

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

```
<?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="claire" a="50" v="Lyon"/>
```

```
</c>
```

```
for $p in //p
where every $x in //p[@v=$p/@v]
    satisfies $x/@a <= $p/@a
return $p
```

Réponse

```
for $p in //p
where every $x in //p[@v=$p/@v]
    satisfies $x/@a <= $p/@a
return $p
```

<resultat>

<p n="paul" a="60" v="Paris"/>

<p n="jean" a="60" v="Nice"/>

<p n="claudes" a="50" v="Lyon"/>

</resultat>

La requête renvoie la personne la plus âgée de chaque ville.

Jointure et élimination de doublons

Pour chaque auteur, la liste de ses livres.

```
<results>
{
  for $v in distinct-values(//author),
    $a in /descendant::author[.=$v][1]
  return
    <author>
      { $a/last,
        for $b in //book where $b/author = $a
        return $b/title
      }
    </author>
}
</results>
```

```
<results>
  <author>
    <last>Stevens</last>
    <title>TCP/IP Illustrated</title>
    <title>Advanced Programming in Unix</title>
  </author>
  <author>
    <last>Abiteboul</last>
    <title>Data on the Web</title>
    <title>Database theory</title>
  </author>
  <author>
    <last>Buneman</last>
    <title>Data on the Web</title>
  </author>
  ...
</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

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>