

## Cours 7 : Programmer avec XQuery

## XSL versus XQuery

---

- Définir un vrai [langage de requêtes](#) pour XML
- Par rapport à XSLT, l'objectif de transformation est le même et certains éléments sont communs (XPath, XML-Schema).
- XQuery : point de vue "[bases de données](#)",
- XSLT : point de vue "[gestion de documents](#)".

## Historique

---

- ✓ [1998](#) : atelier de travail W3C pour XML Query
- ✓ [1999](#) : groupe de travail W3C sur XML Query (actuellement 39 membres, 25 compagnies)
- ✓ [2000](#) : le groupe de travail définit les requêtes, les cas d'utilisation et le modèle de donnée.
- ✓ [2001](#) : le groupe de travail publie un premier *draft*
- ✓ [2007](#) : XQuery 1.0

## XQuery

---

- n'est [pas](#) un dialecte XML
- est un [sur-ensemble](#) de XPath 2.0
- utilise le même [modèle de données](#) que XPath : séquences d'items, un item étant un nœud ou une valeur atomique.
- utilise le [typage](#) de XML-Schema
- permet de [définir des fonctions](#), et utiliser toutes les fonctions prédéfinies pour XPath.

## Un modèle simple

---

- Une **valeur** est une séquence ordonnée de 0 ou plusieurs items.
- Un **item** est un nœud ou une valeur atomique.
- Il y a 7 sortes de nœuds:
  - ▶ Document
  - ▶ Élément
  - ▶ Attribut
  - ▶ Texte
  - ▶ Commentaire
  - ▶ Instruction
  - ▶ Espace de nom

## Exemples de valeurs

---

- 47
- <A/>
- (1, 2, 3)
- (47, <A/>, "Hello")
- ( )
- Un document XML
- Un attribut seul

## Remarques sur les valeurs

---

- Il n'y a pas de distinction entre un item et une séquence de longueur 1.
- Il n'y a pas de séquences imbriquées
- Il n'y a pas de valeur nulle
- Une séquence peut être vide
- Une séquence peut contenir des données hétérogènes
- Toutes les séquences sont ordonnées

## Règles générales pour XQuery

---

- XQuery est un langage sensible à la casse. Les mots clés sont en minuscules.
- Chaque expression a une valeur, et **pas d'effet de bord**.
- Les expressions sont **composables**.
- Les expressions peuvent générer des erreurs. Les commentaires sont possibles (**: un commentaire :**)

## Qu'est ce qu'une requête XQuery?

---

Une requête est une expression qui

- Lit une séquence de fragments XML ou de valeurs atomiques.
- Retourne une séquence de fragments XML ou de valeurs atomiques.

## Qu'est ce qu'une requête XQuery?

---

Les formes principales que peuvent prendre une expression XQuery sont :

- Expressions de chemins
- Constructeurs
- Expressions FLWOR
- Expressions de listes
- Conditions
- Expressions quantifiées
- Expressions de types de données
- Fonctions

## Contexte d'évaluation

---

Les expressions sont évaluées relativement à un contexte:

- Espace de nom
- Variables
- Fonctions
- Date et heure
- Item contexte (nœud courant)
- Position (dans la séquence en train d'être traitée)
- Taille de cette séquence

## Les expressions de chemin

---

Vous connaissez déjà XQuery !

Une expression de chemin XPath est une requête XQuery :

Un chemin retourne un ensemble ordonné de nœuds d'un document.

## Exemple

---

TP6, question 2 : extraire le sous-arbre des clubs.

```
xquery version "1.0";
doc("championnat.xml")//clubs
```

## Autre exemple

---

TP6, question 3 : construire un tableau HTML à partir des clubs.

```
xquery version "1.0";
<html>
  <body>
    <h1>Les clubs de Ligue 1 -- saison 2010-2011</h1>
    <table>
      <thead><tr><th>ville</th><th>club</th></tr></thead>
      <tbody>
        { for $c in doc("championnat.xml")//clubs/club return
          <tr>
            <td>{$c/ville/text()}</td>
            <td>{$c/nom/text()}</td>
          </tr>
        }
      </tbody>
    </table>
  </body>
</html>
```

## Constructeur

---

- Dans l'exemple précédent, les noms des éléments sont connus, on peut donc écrire les balises telles quelles.
- On peut définir des [éléments ou attributs](#) dont le nom et le contenu sont [calculés](#) (comme en XSLT).

```
element
{fn:node-name($e)}
{$e/@*, 2 * fn:data($e)}
```

si \$e a pour valeur

```
<length units="inches">5</length>,
alors le résultat de cette expression est l'élément :
<length units="inches">10</length>.
```

## Autres exemples

---

```
element book {
  attribute isbn {"isbn-0060229357" },
  element title { "Harold and the Purple Crayon"},
  element author {
    element first { "Crockett" },
    element last { "Johnson" }
  }
}
```

équivalent à

```
<book isbn="isbn-0060229357">
  <title>Harold and the Purple Crayon</title>
  <author>
    <first>Crockett</first>
    <last>Johnson</last>
  </author>
</book>
```

## Expression FLWOR

Une expression FLWOR lie des variables, applique des prédicats, et construit un nouveau résultat.



FOR and LET clauses generate a list of tuples of bound variables, preserving input order.

WHERE clause applies a predicate, eliminating some of the tuples

ORDER BY clause imposes an order on the surviving tuples

RETURN clause is executed for each surviving tuple, generating an ordered list of outputs

## Expression FLWOR

- **for**  $\$x$  in Expr la variable  $\$x$  prend **successivement chacune des valeurs** des items de la séquence retournée par Expr

- **let**  $\$x :=$  Expr la variable  $\$x$  prend **la valeur de la séquence** retournée par Expr.

- On peut définir une clause **for** ou **let** avec plusieurs variables :  
**for**  $\$i$  in (1, 2),  $\$j$  in (3, 4) **return** ( $\$i$  ,  $\$j$ )  
a pour résultat la séquence (1 3 1 4 2 3 2 4)

- **where** Expr on ne garde que les valuations des variables qui satisfont l'expression

## Expression FLWOR

- **order by** permet de **trier** le résultat. Plus précisément, on trie les tuples de valeurs associés aux variables.  
**for**  $\$e$  in  $\$employees$   
**order by**  $\$e/salary$  descending,  $\$e/name$  ascending  
**return**  $\$e/name$

- **return** Expr l'expression est évaluée **pour chaque tuple** de valeurs associé aux variables, et **les résultats** de ces évaluations **sont concaténés** pour former la séquence résultat (comme si on avait l'opérateur virgule).

## Exemple

```
for $s in (<one/>, <two/>, <three/>) return  
<out>{$s}</out>
```

a pour résultat

```
<out><one/></out><out><two/></out><out><three/></out>
```

## Exemple

---

```
let $s := (<one/>, <two/>, <three/>)  
return <out>{$s}</out>
```

a pour résultat

```
<out><one/><two/><three/></out>
```

## Exemple

---

```
xquery version "1.0";  
declare default element namespace  
    "http://www.lifl.fr/~yroos/art";  
let $art := doc("./essais/artistes.xml")  
let $cat := doc("./essais/catalogueMusee.xml")  
return  
    <art>  
        <artistes>{  
            for $a in $art//artiste  
            let $o := $cat//oeuvre[auteur/nom = $a/nom]  
            return  
                <artiste>  
                    {$a/*}  
                    <nb-oeuvres>{count($o)}</nb-oeuvres>  
        }</artistes>  
        {$art/art/mouvements}  
    </art>
```

## Exemple (suite)

---

on ne conserve que les artistes qui ont au moins 2 oeuvres

```
<artistes>{  
    for $a in $art//artiste  
    let $o := $cat//oeuvre[auteur/nom = $a/nom]  
    where count($o) >= 2  
    return  
        <artiste>  
            {$a/*}  
            <nb-oeuvres>{count($o)}</nb-oeuvres>  
}</artistes>
```

## Jointures entre documents

---

```
for $p in doc("www.irs.gov/taxpayers.xml")//person  
for $n in doc("neighbors.xml")//neighbor[ssn = $p/ssn]  
return  
    <person>  
        <ssn> { $p/ssn/text() } </ssn>  
        { $n/name }  
        <income> { $p/income/text() } </income>  
    </person>
```

## Autre exemple

---

```
for $d in doc("depts.xml")//deptno
  let $e := doc("emps.xml")//employee[deptno = $d]
  where count($e) >= 10
  order by avg($e/salary) descending
  return
  <big-dept>
    { $d,
      <headcount>{count($e)}</headcount>,
      <avgsal>{avg($e/salary)}</avgsal>
    }
  </big-dept>
```

Résultat : Retourne la liste des départements avec plus de 10 employés, classés par salaire

## Opérations sur des listes

---

- XPath gère des listes de valeurs : (7,9, <onze/>)
- XQuery dispose d'opérateurs pour gérer les listes
  - Concaténation
  - Opérations ensemblistes (union, intersection, différence)
  - Fonctions (**remove**, **index-of**, **count**, **avg**, **min**, **max** etc...)
- On peut gérer les listes [avec une sémantique ensembliste](#), dans ce cas les duplicata sont retirés, les nœuds sont fusionnés basés sur leur identité, et l'ordre est préservé.

## Exemple

---

```
for $p in distinct-values(doc("bib.xml")//publisher)
let $a :=
  avg(doc("bib.xml")//book[publisher = $p]/price)
return
  <publisher>
    <name>{ $p/text() }</name>
    <avgprice>{ $a }</avgprice>
  </publisher>
```

Résultat : Liste chaque éditeur et le prix moyen de leurs livres

## Autres expressions

---

- Unordered** {expr}
- If** (expr1) **then** expr2 **else** expr3
- Some** var in expr1 **satisfies** expr2
- Every** var in expr1 **satisfies** expr2

## unordered

---

```
unordered {
  for $p in fn:doc("parts.xml")/parts/part[color = "Red"],
    $s in fn:doc("suppliers.xml")/suppliers/supplier
  where $p/suppno = $s/suppno
  return
    <ps>
      { $p/partno, $s/suppno }
    </ps>
}
```

## if then else

---

```
for $h in doc("library.xml")//holding
return
  <holding>
    { $h/title, if ($h/@type = "Journal")
      then $h/editor
      else $h/author }
  </holding>
```

## some

---

```
for $b in doc("bib.xml")//book
  where some $p in $b//paragraph satisfies
    (contains($p,"sailing") and
     contains($p,"windsurfing"))
  return $b/title
```

## every

---

```
for $b in doc("bib.xml")//book
  where every $p in $b//paragraph satisfies
    contains($p,"sailing")
  return $b/title
```



## Structure d'une requête

---

✓ Le Prologue contient:

- Déclarations de Namespace
- Importations de schémas
- Importation de Modules
- Définitions de fonctions
- Déclarations de variable globales et externes

✓ Le Corps contient: Une expression qui définit le résultat de la requête

## Les espaces de nom

---

✓ En XQuery, tous les noms sont des noms qualifiés.

✓ Les déclarations sont faites dans le prologue.

```
declare namespace acme = "http://www.acme.com/names"
```

## Importation de schéma

---

✓ Un espace de nom est défini par un schéma

✓ On lie l'espace de nom au préfixe et on importe le schéma par l'instruction suivante

```
import schema namespace acme =  
"http://www.acme.com/names"  
at  
"http://www.acme.com/schemas/names.xsd"
```

## Espaces de nom prédéfinis

---

✓ `http://www.w3.org/2001/XMLSchema` associé au préfixe **xs:**

✓ `http://www.w3.org/2005/xpath-functions`, contenant toutes les fonctions XPATH, associé au préfixe **fn:**

✓ il existe aussi le préfixe **local:** pour les fonctions définies par l'utilisateur

## Définition de fonctions

---

- ✓ À faire apparaître dans le prologue
- ✓ Les fonctions ne peuvent pas être surchargées.
- ✓ Une grande partie de XML n'est pas typé. XQuery tente de forcer le type à celui attendu par un cast automatique.

Exemple: `abs($x)` attend un argument numérique

- Si `$x` est un nombre, retourner sa valeur absolue
- Si `$x` n'a pas de type, le transformer en nombre
- Si `$x` est un nœud, extraire sa valeur puis la traiter comme ci-dessus

Il y a donc incompatibilité avec la surcharge !

## Exemple

---

```
define function local:depth($n as node()) as
xs:integer
{
  (: A node with no children has depth 1 :)
  (: Else, add 1 to max depth of children :)
  if (empty($n/*)) then 1
  else max(for $c in $n/* return depth($c)) + 1
} ;
```

## Exemple

---

```
(: determine si une l'année d'une date est bissextile :)
declare function fd:bissextile($dt as xs:date) as xs:boolean {
  let $i := fd:annee($dt)
  return $i mod 4 = 0 and ($i mod 100 != 0 or $i mod 400 =0)
};
```

```
(: ajoute des jours à une date :)
declare function fd:ajoute-jour($dt as xs:date , $j as
xs:integer)
  as xs:date{
  if ($j = 0) then $dt
  else fd:ajoute-jour(fd:ajoute-un-jour($dt) , $j - 1)
} ;
```

## Modules

---

Une requête peut utiliser plusieurs modules

- ✓ Module principal
  - Contient la requête
  - Est exécutable
- ✓ Module bibliothèque
  - Définit les fonctions et variables
  - Déclare son espace de noms
  - Peut être importé
  - Exporte ses variables et ses fonctions dans son espace de noms
- ✓ Module imports

## Exemple de module bibliothèque

---

```
xquery version "1.0" ;

(: Un module doit toujours déclarer un namespace :)
module namespace fa = "http://fil.univ-lille1.fr/miage-fa-fc/library" ;

(: le namespace par défaut est celui exporté par le schéma des données de base :)
declare default element namespace "http://fil.univ-lille1.fr/miage-fa-fc" ;

import module namespace fd = "http://fil.univ-lille1.fr/miage-fa-fc/dates"
at "http://www.fil.univ-lille1.fr/FORMATIONS/MIAGE-FC-FA/schemas/dates.xq" ;

(: Le fichier qui contient toutes les données :)
declare variable $fa:miage :=
  doc("http://www.fil.univ-lille1.fr/FORMATIONS/MIAGE-FC-FA/schemas/miage-fa-
fc.xml") ;

(: retourne le début d'une semaine de séminaire :)
declare function fa:debut-seminaire($semaine as xs:integer)
as xs:date{
  let $d := $fa:miage//trimestre[@num="T0"]/du
  return
    if ($semaine = 1) then $d
    else fd:ajoute-jour($d , 7)
};

...
```

## Exemple d'importation de module

---

```
xquery version "1.0" ;
import module
  namespace b = "http://fil.univ-lille1.fr/miage-fa-fc/library"
  at "http://www.fil.univ-lille1.fr/FORMATIONS/MIAGE-FC-FA/
schemas/miage-fa-fc.xq" ;

b:edt("M1")
```