XQUERY?

- Est un langage **d'interrogation** de données XML
- N'est pas un dialecte XML
- Est un sur-ensemble de **XPath 2.0**
- Utilise le même **modèle de données (XDM)** que XPath : séquences d'items, un item étant un noeud ou une valeur atomique**.**
- Utilise le typage de XML-Schema

Objectifs

- Comment interroger/manipuler des documents XML ?
  - SQL : stocker XML dans une BDR
  - XPath : extraction de fragments d'arbres
  - XSLT : Extraction + transformation (règles)

- Un vrai langage **de requêtes XML**
  - Le « SQL » de XML

- XQuery 1.0 recommandation de janvier 2007 (langage typé)
  XQuery 3.0 proposed recommendation (2013)

- Véritable langage de programmation, très puissant

Xquery versus SQL

- Semblable à SQL pour les BD sauf que SQL:
  - Travaille sur des bases de données relationnelles: données fortement en relation dans des tables
  - Base ses requêtes sur la recherche de ces relations fortes

- En quoi les données XML sont-elles différentes?
  - Les données relationnelles SQL sont denses
    - Chaque rangée a une valeur dans chaque colonne
    - Problème des valeurs nulles
  - Ce n'est pas le cas de XML qui peut avoir
    - Des éléments vides
    - Des éléments absents
  - C'est un degré de liberté supérieur pour XML
  - C'est pourquoi on appelle les documents XML: semi-structurés

Base de données natives

- Une base de données XML Native (*NXD* en anglais) est une base de données qui s'appuie sur le modèle de données fourni par XML.

- Utilise typiquement des langages de requête XML comme XPath ou XQuery.

- Exemples de logiciels de BD XML:
  - eXist
  - Xindice
  - BaseX. Open Source,
  - DB-XML
  - Oracle Berkeley DB XML. Open Source basé sur Oracle Berkeley DB
  - etc

XQuery/XPath

- XPath
  - XPath permet d'exprimer des requêtes de filtrage sur des arbres
  - Il n'est pas possible
    - De créer des nœuds
    - De construire des arbres/des documents nouveaux
    - De trier des nœuds
    - Il est difficile d'exprimer des jointures
- XQuery
  - XQuery est un langage de requête complet qui permet
    - De créer des nœuds et de construire des arbres nouveaux
    - De définir et d'instancier des variables
    - De définir des fonctions….

Expression XQuery

- Une requête XQuery est une composition d'expressions
- XQuery est sensible à la casse (case-sensitive), les mots clef du langage doivent être écrits en minuscule.
- Des commentaires peuvent être ajoutés en respectant la syntaxe (:ceci est mon commentaire :)
- 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.

Exemple de requête XQuery

- employe.xml

```
<employes>
  <employe>
    <nom>Dupond</nom>
    <prenom>Albert</prenom>
    <date_naissance>23/09/1958</date_naissance>
  </employe>
  <employe>
    <nom>Dupont</nom>
    <prenom>Alphonse</prenom>
    <date_naissance>23/12/1975</date_naissance>
  </employe>
  <employe>
    <nom>Dupont</nom>
    <prenom>Isabelle</prenom>
    <date_naissance>12/03/1967</date_naissance>
  </employe>
...
</employes>
```

Requête XQuery

```
xquery version "1.0" encoding "utf-8";

for $b in doc ("employe.xml")//employe
   where $b/nom = "Dupont"
   return
    <dupont>{
     $b/prenom,
     $b/date_naissance
    }</dupont>
```

Résultat

```
<dupont>
 <prenom>Alphonse</prenom>
 <date_naissance>23/12/1975</date_naissance>
</dupont>
<dupont>
 <prenom>Isabelle</prenom>
 <date_naissance>12/03/1967</date_naissance>
</dupont>
```

Récupérer un document ou une collection

- Une requête prend généralement en entrée un document ou une collection de documents

- Ces entrées sont spécifiées à l'aide des fonctions suivantes:
  - *doc(''nomDocument.xml'')* prend en entrée l'URI d'un document XML et renvoie le nœud racine du document.

  - *collection(''nom'')* permet de récupérer une forêt de documents XML nommée nom et mémorisée dans un SGBD-XML et renvoie une séquence composées des nœuds racines des documents de la collection

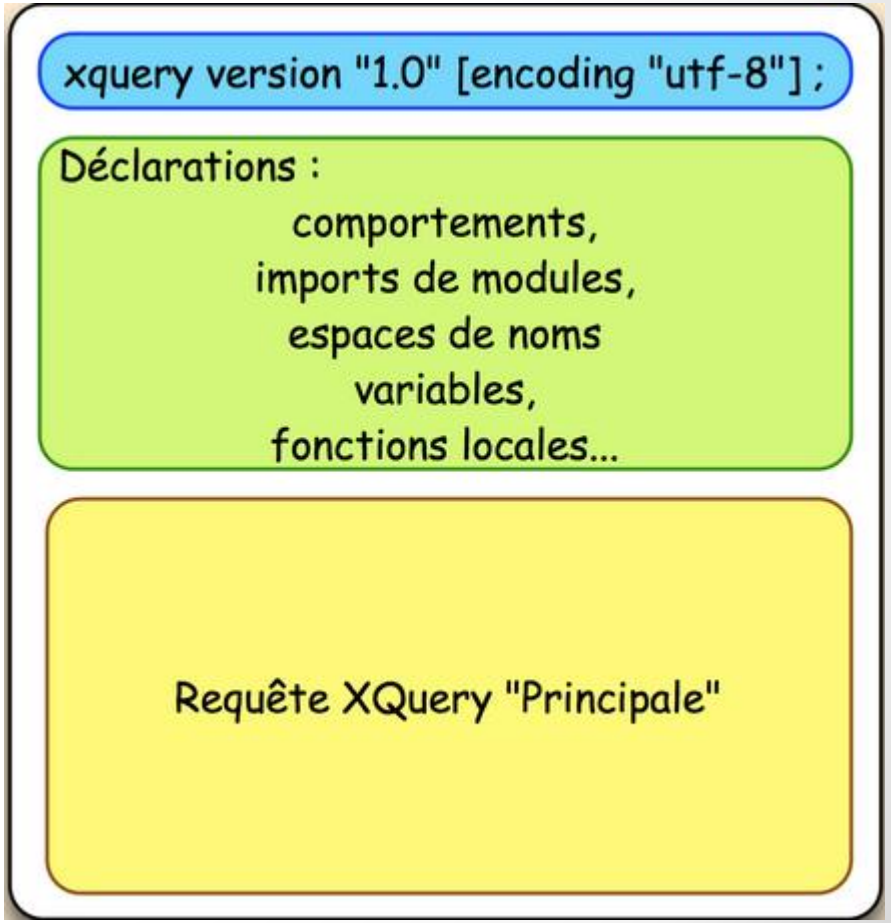
```
collection("shakespeare")//TITLE
```

L'expression est évaluée pour chaque document de la collection, dans la séquence issue de la collection (''shakespeare'')

Forme d'une requête XQuery

Composée de trois parties :

- Une ligne d'entête commencée par "xquery" et contenant la version et, éventuellement l'encodage du document ;
- Un ensemble de déclarations :
  - déclarations de variables,
  - déclarations de fonctions utilisateur, détermination des espaces de nom et de leur utilisation,
  - etc
- L'expression XQuery à exécuter.
- La ligne d'entête et les déclarations sont toutes terminées par ";"



Forme d'une requête XQuery

- Définition:
  - Une requête XQuery est une composition d'expressions
  - Chaque expression a une valeur ou retourne une erreur.

- **Forme 1: Expressions simples:**
  - Valeurs atomiques: 46, "salut " ,
  - Variables, opérateurs

- **Forme 2: Expressions complexes**
  - Expressions de chemins (XPath): FILM//ACTEUR
  - Expressions FLWR: For-Let-Where-Return
  - Tests: if-then-return-else-return
  - Fonctions:
    - Racines (collection ("url"), doc(" url "))
    - prédéfinies (celles définies par XPath),
    - à définir (fonctions utilisateurs)

Expressions de chemin

- Toutes les expressions XPath sont des expressions de Xquery.

- Utilisent les sélecteurs XPath

Selector	Selected nodes
/	Document root
//	Any sub-path
*	Any element
name	Element of tag name
@*	Any attribute
@name	Attribute of name name
text()	Any text node
processing-instruction('name')	Processing instruction of given name
comment()	Any comment node
node()	Any node
id('value')	Element of id value

Construction de noeuds

- Un constructeur produit un nouveau nœud
  - Il existe des constructeurs pour chaque sorte de nœud: élément et attribut

- A l'exécution
  - les parties connues sont recopiées: **Constructeurs directs**
  - les parties calculées sont évaluées: **Constructeurs calculés**

Construction de noeuds

- **Cas 1 : le nom du nœud est connu**, son contenu est calculé par une expression
- Attention: les accolades sont obligatoires sinon l'expression est prise pour du texte
- **Requête:** auteurs du 2ème livre

bib.xml

```
<bib>
  <book title="Comprendre XSLT">
    <author> <la>Amann</la><fi>B.</fi>
    </author>
    <author><la>Rigaux</la><fi>P.</fi>
    </author>
    <publisher>OReilly</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>
</bib>
```

```
<auteurs>
{ doc("bib.xml")//book[2]/author/la }
(:les accolades sont
obligatoires sinon l'expression est prise pour du texte:)
</auteurs>
```

Résultat :

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

Construction de noeuds

- **Cas2 :** le nom du nœud et son contenu **sont calculés** par une expression

- Constructeurs d'élément et d'attribut
  - element nom *{* expr-contenu *}* *ou* element *{*expr*}{* expr-contenu *}*
  - attribute *{* expr-nom *} {* expr-contenu *}*

- **Requête:**

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

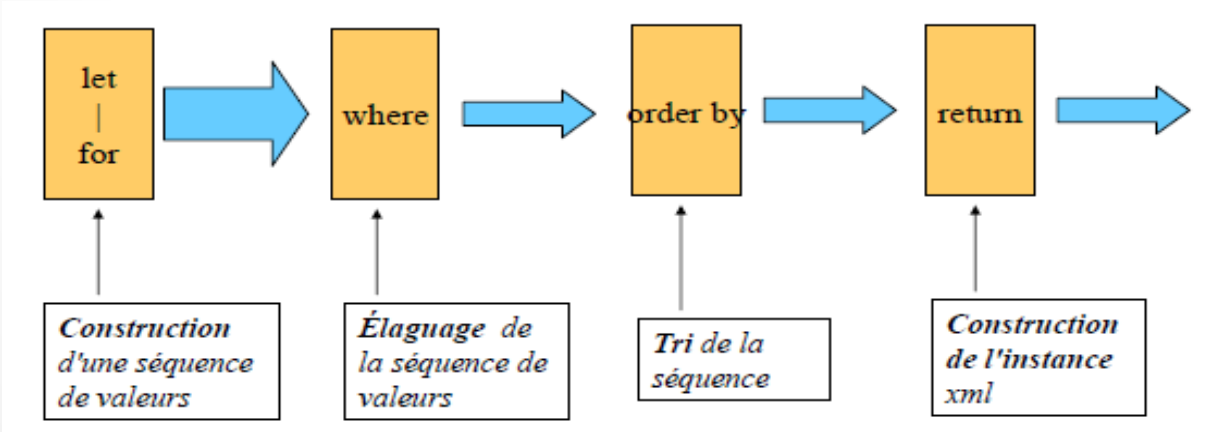
Même Résultat :

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

Expression FLWOR

- **FOR … LET … WHERE …ORDER BY … RETURN**

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



La forme minimale comporte un "for" ou un "let" et un "return

Expression FLWOR

Itération For: *for* **$var** *in* **exp**

- $*var* est une variable: nom précédé de $
- ***for*** instancie une variable $*var* en lui faisant prendre (successivement) les valeurs des items d'une séquence en entrée
- **Requête**:

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

- **Résultat:**

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

- La clause **return exp** permet de construire le résultat

Expression FLWOR

Affectation Let: let $variable := expression Xpath

Permet d'associer à une variable une liste de noeuds résultant de l'évaluation d'une expression Xpath

- **Requête1:**

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

Résultat

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

- **Requête 2:**

```
xquery version "1.0" encoding "utf-8";

let $b := doc("bib.xml")//book[2]
 let $p:=count($b/author)
return $p
```

Résultat: 3

Expression FLWOR

La Clause 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:** déterminer tous les titres des livres publiés en 2001

```
xquery version "1.0" encoding "utf-8";
 <title>{
 for $b in doc("bib.xml")//book
where $b/@year="2001"
return $b/@title
}
</title>
```

- **Résultat:**

```
<title title="Spatial Databases"/>
```

NB: Pas de let, dans cette requête. On n'est pas obligé à avoir les 2, for et let. Ici pas de order by non plus.

Expression FLWOR

Order By: *exp1 order by exp2 (ascending | descending)*

Permet de réordonner (tri) les n-uplets dans l'ordre croissant(ascending) et décroissant (descending).

- **Requête:**

```
xquery version "1.0" encoding "utf-8";
 <livres>{
 for $b in doc("bib.xml")//book
order by $b/@title descending
return <livre>{$b/@title,$b/@year}</livre>
}
</livres>
```

- **Résultat:**

```
<livres>
  <livre title="Spatial Databases" year="2001"/>
  <livre title="Comprendre XSLT"/>
</livres>
```

Exercices

- Films.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<FILM annee="1992">
  <TITRE>Reservoir dogs</TITRE>
  <GENRE>Policier</GENRE>
  <PAYS>USA</PAYS>
  <MES idref="37"/>
  <ROLES>
    <ROLE>
      <PRENOM>Quentin</PRENOM>
      <NOM>Tarantino</NOM>
      <INTITULE>Mr. Brown </INTITULE>
    </ROLE>
    <ROLE>
      <PRENOM>Harvey</PRENOM>
      <NOM>Keitel</NOM>
      <INTITULE>Mr. White/Larry</INTITULE>
    </ROLE>
  </ROLES>
</FILM>
```

- Afficher tous les titres des films

```
xquery version "1.0" encoding "utf-8";
for $b in doc("Films.xml")//FILM
return <FILM> {$b/TITRE}</FILM>
```

- Quel rôle joue Harvey Keitel dans le film Reservoir Dogs?

```
for $b in doc("Films.xml")//FILM
where $b/TITRE="Reservoir dogs"
return
    for $a in $b/ROLES/ROLE
    where $a/PRENOM="Harvey" and $a/NOM="Keitel"
    return $a/INTITULE
```

ou

```
for $b in doc("Films.xml")//FILM,  $a in $b/ROLES/ROLE
where $b/TITRE="Reservoir dogs"
        and  $a/PRENOM="Harvey"
        and $a/NOM="Keitel"
    return $a/INTITULE
```

Tests: if-then-else

- **Syntaxe**

  if <exp1> then <exp2> else <exp3>

- teste la valeur booléenne de exp1
- Retourne la valeur de exp2 si elle est vraie
- Retourne la valeur de exp3 sinon

Tests: if-then-else

- **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>
```

- **Requête:**

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

Résultat:

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

Quantificateurs

SOME ... IN ... SATISFIES
EVERY ... IN ... SATISFIES

- **SOME $x in expr1 SATISFIES expr2** signifie qu'il existe AU MOINS un noeud renvoyé par **expr1** qui satisfait **expr2.**

  Requête: some $b in doc(″bib.xml″)//book
  satisfies $b/@year >2003

  Résultat: Renvoie **true** si au moins un livre a un attribut dont la valeur est supérieure à 2003.

- **EVERY $x in expr1 SATISFIES expr2** signifie que TOUS les noeuds renvoyés par **expr1** satisfont **expr2**.

  Requête: Every $b in doc(″bib.xml″)//book satisfies $b/@year

  Résultat: Renvoie **true** si tous les livres ont un attribut **year.**

Quantificateurs

- 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>
```

- Requête

```
xquery version "1.0" encoding "utf-8";

for $b in doc("bib.xml")//book
where every $p in $b//publisher satisfies
contains($p,"Morgan Kaufmann Publishers")
return $b/price
```

- Résultat

```
<price>35.00</price>
<price>39.95</price>
```

Expressions avec compteur

- **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>
```

Requête

```
xquery version "1.0";
for $p at $i in doc("bib.xml")//book/author/la
return <auteur numero="{$i}">{data($p)}</auteur>
```

Résultat

```
<auteur numero="1">Amann</auteur>
<auteur numero="2">Rigaux</auteur>
<auteur numero="3">Rigaux</auteur>
<auteur numero="4">Scholl</auteur>
<auteur numero="5">Voisard</auteur>
```

La fonction data() appliquée à un élément retourne son contenu.

Jointures

Utilisation de plusieurs documents XML

- fournisseur.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<fournisseur>
  <Row>
    <F>f1</F>
    <Nom>Barnibus</Nom>
    <Remise>0,05</Remise>
    <Ville>Paris</Ville>
  </Row>
</fournisseur>
```

fourniture.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<maFourniture>
  <Row>
    <F>f1</F>
    <code>p1</code>
    <Qte>1</Qte>
  </Row>
  <Row>
    <F>f2</F>
    <code>p2</code>
    <Qte>1</Qte>
  </Row>
</maFourniture>
```

produit.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<produit>
  <Row>
    <code>p1</code>
    <Nom_p>cassis</Nom_p>
    <Couleur>rouge</Couleur>
    <Origine>Dijon</Origine>
  </Row>
</produit>
```

Jointures

Requête: code des produits fournis par chaque fournisseur

```
xquery version "1.0" encoding "utf-8";

for $fournisseur in doc("fournisseur.xml")/fournisseur/Row,
$maFourniture in doc("fourniture.xml")/maFourniture/Row
where
$maFourniture/F = $fournisseur/F
return
<prod>{$maFourniture/code}</prod>
```

Résultat:

```
<prod>
   <code>p1</code>
</prod>
```

Jointures

Requête: fournisseur fournissant des produits de couleur rouge

```
xquery version "1.0" encoding "utf-8";

for $fournisseur in doc("fournisseur.xml")/fournisseur/Row,
$maFourniture in doc("fourniture.xml")/maFourniture/Row,
$produit in doc("produit.xml")/produit/Row
where
$maFourniture/F = $fournisseur/F and
$maFourniture/code = $produit/code and
$produit/Couleur = 'rouge'
return
<prod>{$fournisseur/Nom/text()}</prod>
```

Résultat:

```
<prod>Barnibus</prod>
```

Jointures

Requête: Pour chaque fournisseur, donner Pour chaque couleur, la liste des produits (on veut tous les fournisseurs)

```
for $fournisseur in doc("fournisseur.xml")/fournisseur/Row
return <fournisseur>{$fournisseur/Nom,
   for $couleur in distinct-values(doc("produit.xml")/produit/Row/Couleur)
   return <prod couleur="{$couleur}">
   { for $produit in doc("produit.xml")/produit/Row,
        $maFourniture in doc("fourniture.xml")/maFourniture/Row
   where
      $maFourniture/F = $fournisseur/F and
      $maFourniture/code = $produit/code and
      (:sélection de la couleur :)
      $produit/Couleur = $couleur
   return $produit/Nom_p} </prod>
   }
</fournisseur>
```

Résultat:

```
<fournisseur>
   <Nom>Barnibus</Nom>
   <prod couleur="rouge">
      <Nom_p>cassis</Nom_p>
   </prod>
</fournisseur>
```

Fonctions

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="37.475"/>
```

Fonctions à définir

Déclaration: declare function fun_name ($arg1 as type1, $arg2 as type2, ...) as type_retour { corps de la fonction };

où

- fun_name est soit dans un espace de noms déclaré soit préfixé par “local” (espace de noms des fonctions pas défaut)
- Les types peuvent être :
  - xs:*type*, où *type* est un type défini dans la norme XML Schema, par exemple xs:string, xs:boolean, xs:number, xs:integer
  - element(), attribute() - le nom peut être spécifié entre les parenthèses
  - Le type peut être suivi de *(séquence qui peut être vide),+ (séquence non vide) ou ? (éventuellement séquencevide) afin d'exprimer le fait de pouvoir gérer une séquence d'éléments au lieu d'un seul élément
- **Exemple:**

```
declare function local:NombreAuteurs( $b as xs:string+ )
as xs:integer?
{
let $b := doc("bib.xml")//element(book)

return (count ($b/author))
};

 let $a:= doc("bib.xml")//book
    return <auteur>{local:NombreAuteurs($a/author) } </auteur>
```

Résultat

```
<auteur>8</auteur>
```

Opérateurs

- Les mêmes que XPath2.0
- **Opérateurs arithmétiques:**

Opérateurs arithmétiques	Définition
+ , -	Addition, soustraction
Div	Division
Mod Exemple: $y mod 2	Reste de la division entière (modulo)

- **Opérateurs booléens:**

Opérateurs booléens	Définition
and Exemple: $x=2 and $y=4	« ET » logique
or Exemple: $x=2 or $y=4	« ou » logique
Not() Exemple: fn:not(1 and 1) →FALSE	Négation logique

Opérateurs

- **Opérateurs de comparaison:**
  - **Comparaison de valeurs atomiques:** eq, ne, lt,le, gt et ge
    - 5 gt 7.5 →FALSE (gt signifie plus grand)

      La comparaison est effectuée après la conversion de 5 en xs:float
  - **Comparaison de position des nœuds avec << et >>** :
    - n1<<n2 (n1>>n2) ←→ n1 apparaît avant (après) n2 dans le document
  - **Comparaison de nœuds avec is:**
    - Comparaison de l'identité des nœuds, pas de leurs valeurs
    - ***n1 is n2*** si ***n1*** est identique à ***n2***

Opérateurs

- Opérations sur **les séquences (1/2)**
  - **Trois opérateurs:**
    - Union (*union*),
    - intersection (*intersect*),
    - différence (*except*)

bib.xml

```
<bib>
    <book title="Comprendre XSLT">
        <author> <la>Amann</la><fi>B.</fi>
        </author>
        <author><la>Rigaux</la><fi>P.</fi>
        </author>
        <publisher>OReilly</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>
</bib>
```

Exemple Requête

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

Résultat

```
<livre>
   Tous les sous éléments sauf auteurs
   <publisher>OReilly</publisher>
   <price>28.95</price>
</livre>
```

Opérateurs

- Opérations sur **les séquences (2/2)**
  - **Concaténation:** exp1,exp2

Séquence constituée des items de la séquence *valeur*(*exp*1) suivie des items de la séquence *valeur*(*exp*2)

(1,2,3)→ 1 2 3

1+2, 4-2, 3*2 -> 3,2,6

Exemple requête

```
<livre>
 { doc("bib.xml")//book[1]/(price,author) }
 </livre>
```

Résultat

```
<livre>
  <price>28.95</price>
  <author>
    <la>Amann</la>
    <fi>B.</fi>
  </author>
  <author>
    <la>Rigaux</la>
    <fi>P.</fi>
  </author>
</livre>
```

Règles générales pour XQuery

- XQuery est un langage sensible à la casse. **Les mots clés (for, let, where, return, if,…)** sont en minuscules.
- Chaque expression a une valeur, et pas d'effet de bord.
- Tous les axes (ancestor, ancestor-or-self, following, following-sibling, preceding et preceding-sibling) ne sont pas supportés
- Les expressions peuvent générer des erreurs.
- Les commentaires sont possibles (: un commentaire :)