XQuery

point important

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

//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 élements author et editor

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

Les nœuds (éléments) sont comparés par leur valeur textuelle:
- < a>< b>5< /b>< /a> eq < a>5< /a> → true()
- < last>Bob< /last> eq < first>Bob< /first> → true()

Utiliser la fonction deep-equal pour comparer la structure et la valeur:
- deep-equal((< a>< b>1< /b>< /a>),(< a>1< /a>)) → false()
- deep-equal((< a>1< /a>),(< a>1< /a>)) → true()

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;
- « 5 is 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

```
for – let – where – order by – return

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.

```
– 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)
```

```
– 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
```

- return

```
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)


for $e in document(''doc.xml'')//employees
return $e/first, $e/last -> erreur
```

- order by : ascending ou descending ,

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

- utilisation de `if then else ``

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

- utilisation de some

```
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.

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.
```

- utilisationd de `every`

```
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


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.
```

- instance of :

```
<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()
```

- distinct-values :

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

TD

ex1

1. le nom de tous les menus des restaurants

```
<results>{
for $m in document("guide.xml")//menu
return <menu nom = "{$m/@nom}"}/>
}
</results>
```

2. Donner le nom et le prix de tous les menus dont le prix est inférieur à 100

```
<results>{
    for $m in document("guide.xml")//menu
    return $m
}
<results>
```

avec `document("guide.xml")` optionnel 3. Donner le nom des restaurants 2 étoiles avec leur nom de menu

```
<results>{
for $r in //restaurant
where $r/@etoile = 2
return <restaurant nom = "{$r/@nom}">{
    for $m in $r/menu
    return <menu nom = "{$m/@nom}"/>
    }</restaurant>
}
</results>
```

4. Donner le nom de chaque restaurant avec son numéro de département

```
<results> {
for $r in //restaurant
    for $v //ville
        where $v/@nom = $r/@ville
    return <restaurant nom = "{$r/@nom}" departement = "{$v/@departement}"
}</results>
```

autre solution

```
<results> {
for $r in //restaurant
    return <restaurant nom = "{$r/@nom}" departement = "//ville[@nom =
$r/@ville]/@departement"/>
}</results>
```

5. Quels sont les restaurants ayant (au moins) un menu dont le prix est égal au tarif de visite du plus beau monument de la ville ? Donner le nom du restaurant et le tarif.

```
<results>{
    for $r in //restaurant
    for $m in $r/menu
        for $v in //ville
        where $v/@nom = $r/@ville and  $m/@prix  =
$v/plusBeauMonument/@tarif
        return
        <result>
            <restaurant nom  = "{$r/@nom}"/>
            <tarif_monument prix = " {$m/@prix}"/>
        </result>
    }
}</results>
```

solution ci-dessus presque ok mais contient des elements `<result>` en double si un restaurant a plus d'un menu qui satisfait la condition

```
<results>{
    for $r in //restaurant,
        $v in //ville,
        $p in distinct-values($r/menu/@prix)
    where $v/@nom = $r/@ville and  $p= $v/plusBeauMonument/@tarif
    return
    <result>
        <restaurant nom  = "{$r/@nom}"/>
        <tarif_monument prix = " {$p}"/>
    </result>
    }
}</results>
```

Remarque :

- ne pas ecrire `distinct-values(//menu)` car c'est un element complexe
- a utiliser quand on a les ensemble de valeur

autre solution avec '' il existe au moins 1 menu qui satisfait la condition sur le tarif" : **sous format** : `some variable in satisfies`

```
<results>{
    for $r in //restaurant,
        $v in //ville,
    where $v/@nom = $r/@ville and  some $m in $r/menu satisfies $m/@prix =
$v/plusBeauMonument/@tarif
    return
    <result>
        <restaurant nom  = "{$r/@nom}"/>
        <tarif_monument prix = " {$m/@prix }"/>
    </result>
```

```
    }
}</results>
```

Pour les XQuery, on peut retourner un arbre mais pas un foret

- **si on veut retourner 2 elements nous avons besoin de mettre un element au dessus**

ex2

R4 : Auto jointure d'un document
Pour chaque auteur, donner la liste des titres de ses livres. Un élément du résultat contient un auteur avec tous les
titres qu'il a écrit. `//author` contient des elements en double`

```
<results>{
    for $l  in distinct-values(//last),
        $f in distinct-values(//author[last = $l]/first)
    return
        <result>
            <author>
                <last>{$l}</last>
                <first>{$f}</first>
            </author>
            <title> {//book[author[last = $l and first $f]]/titre}</title>
        </result>
</results>
}
```

- autre solution

```
<results>{
    for $l  in distinct-values(//last),
        $f in distinct-values(//author[last = $l]/first)
    return
        <result>
            <author>
                <last>{$l}</last>
                <first>{$f}</first>
            </author>
            {
                for $b in //book
                where some $a in $b/author satisfies $a/last = $l and
$a/first =$f
                return $b/title
            }
        </result>
</results>
}
```

Remarque:

- `//book[author/last = $l and author/first = $f]` n'est pas bon car `author/last` est un ensemble qu'on arrivera pas a distinguer

TME

ex1

1. Le nom de la personne dont l'identifiant est “person0”.

```
<results>{
    for $p in //person
    where $p/@id ='person0'
    return $p/name
}</results>
```

2. La valeur initiale (élément initial) des trois premières enchères en cours.

```
<results>{
    for $a in//open_auctions/auction[position() <= 3]
    return
    <result id = "{$a/@id}">
        {$a/initial}
    </result>
  }
</results>
```

3. La valeur de la première et de la dernière augmentation (élément increase de bidder) effectuée sur les trois premières des enchères en cours, selon l'ordre des enchères défini dans les données xml (ne pas trier les enchères chronologiquement).

```
<results>{
    for $a in//open_auctions/auction[position() <= 3]
    return
    <result id= "{$a/@id}">
      <first>{$a/bidder[1]/increase/text()}</first>
      <last>{$a/bidder[last()]/increase/text()}</last>
    </result>
  }
</results>
```

4. Le prix des objets vendus à plus de 480.

```
for $p in //price[text() > 480]
return $p
equivalent
for $p in //price[. > 480]
return $p
```

autre solution

```
for $p in //auction[price > 480]
return $p/price
```

5. Le nom des objets du continent africain

```
for $c in //africa
return $c/item/name
```

6. Le nom des objets du continent africain avec leur prix de vente

```
for $a in //closed_auctions/auction
for $o in //africa/item[@id = $a/itemref/@item]
return
<res>
{$o/name}
{$a/price}
</res>
```

7. Le nombre de personnes qui n'ont pas de page web

```
count(//person[not(homepage)])
```

ex2

resultat faux pour l'instant

```
for $t in distinct-values(//lieutournoi)
for $g in distinct-values(//gain[lieutournoi]/annee)
return <tournoi lieu = "{$t}" annee = "{$g}"/>
```