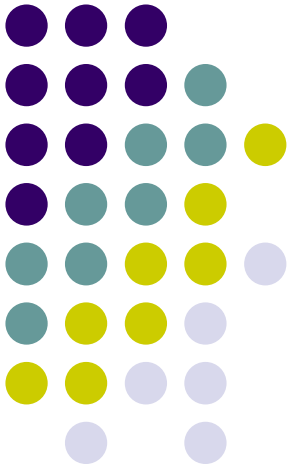
Le Langage XQuery



Pr. Sidi Mohammed Benslimane
École Supérieure en Informatique
08 Mai 1945 – Sidi Bel Abbes –

s.benslimane@esi-sba.dz

Du relationnel à XML

- Modèle Relationnel (années 80)
  - Schéma fixe
  - Mise en œuvre facile
  - Intégration d'objet difficile
- Modèle Objet (années 90)
  - Plus souple et fortement typé
  - Non adapté au structuration faible
  - Mise en œuvre difficile
- Modèle XML (années 2000)
  - Schémas flexibles et irréguliers
  - Données auto-descriptives (Balises et attributs)
  - Modèle de type hypertexte (Support des références)
  - Types de données variés et extensibles (Textes, numériques,..., types utilisateur)

Le stockage d'informations

- La plupart des informations disponibles actuellement sont stockées dans des bases relationnelles.
- Le langage SQL est mature, et bien implanté
- De plus en plus de données semi-structurées sont produites et stockées
- Peut-on adapter SQL aux données semi-structurées ?

Une structure plus complexe

Relationnel : Uniforme et répétitif

- Chaque rangée a une valeur dans chaque colonne.

Par exemple: Tous les comptes en banque ont une structure similaire.

- Problème des valeurs nulles
- Les méta-informations sont stockées à part

XML : très variable !

- Tolérer les éléments absents
- En XML, il est naturel de chercher de manière indépendante du niveau.
  - Exemple : trouver tout ce qui est rouge //*[@couleur ="rouge"]
- Chaque objet XML doit se décrire, les métadonnées sont dans le document,
  - Exemple : trouver tous les éléments qui ont le contenu identique à leur nom: //*[name(.) = string (.)]

En conclusion …

- XML est très différent de SQL, et justifie donc le fait de vouloir créer un langage de requêtes dédié.

- Le langage:
  - Doit respecter le modèle de données XML
  - Doit comprendre les espaces de noms
  - Doit supporter des types de données simples et complexes
  - Doit supporter les quantificateurs existentiels et universels (some et every)
  - Doit supporter les agrégations (sum, count, avg)
  - Doit pouvoir transformer et créer des structures XML
  - Doit pouvoir combiner des informations de plusieurs documents.

Le Langage XQuery: Présentation

- XQuery est une spécification du W3C dont la version 1.0 date de Janvier 2007 et la version 4.0 de Novembre 2020.
- XQuery est un sur-ensemble de Xpath
- XQuery utilise le typage de XML-Schema
- XQuery est un langage de requête XML permettant de:
  - Extraire des informations d'un document XML, ou d'une collection de documents XML.
  - Effectuer des calculs complexes à partir des informations extraites (utiliser toutes les fonctions prédéfinies pour Xpath et définir de nouvelles fonctions)
  - Reconstruire de nouveaux documents ou fragments XML.

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 ou une erreur
- Les formes principales que peuvent prendre une expression Xquery sont :
  - Expressions de chemins (Xpath)
  - Expressions FLWOR (For-Let-Where-Order-Return)
  - Conditions (if-then-return-else-return)
  - Expressions quantifiées (some, every)
  - Expressions de types de données
  - Expressions de listes
  - Fonctions

Expressions de chemins

books.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
    <book category="COOKING">
        <title lang="en"> Everyday italian</title>
        <author>Glada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
    </book>
    <book category="CHILDREN">
        <title lang="en"> Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>
    <book category="WEB">
        <title lang="en"> Learning XML</title>
        <author>Erik T.Ray</author>
        <year>2003</year>
        <price>39.95</price>
    </book>
</bookstore>
```

Requête

- *Liste de tous les auteurs*

doc("books.xml")//book/author

```xml
<author>Glada De Laurentiis</author>
<author>J K. Rowling</author>
<author>Erik T.Ray</author>
```

- *Liste de tous les auteurs et prix*

doc("books.xml")//book/(author, price)

```xml
<author>Glada De Laurentiis</author>
<price>30.00</price>
<author>J K. Rowling</author>
<price>29.99</price>
<author>Erik T.Ray</author>
<price>39.95</price>
```

Expressions: Accès via un prédicat

books.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
    <book category="COOKING">
        <title lang="en"> Everyday italian</title>
        <author>Glada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
    </book>
    <book category="CHILDREN">
        <title lang="en"> Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>
    <book category="WEB">
        <title lang="en"> Learning XML</title>
        <author>Erik T.Ray</author>
        <year>2003</year>
        <price>39.95</price>
    </book>
</bookstore>
```

Requête

- *livres apparus avant 2005*

doc("books.xml")//book[year<"2005"]

```xml
<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T.Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>
```

Construction de noeuds: Production de XML

bib.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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>Morgzan Kaufmann</publisher>
        <price>35.00</price>
    </book>
</bib>
```

Cas 1: nom connu, contenu construit par une expression

Exemple: **noms des auteurs du 2ème livre**

<auteurs>

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

</auteurs>

Résultat :

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

Construction de noeuds: Production de XML

bib.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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>Morgzan Kaufmann</publisher>
        <price>35.00</price>
    </book>
</bib>
```

Cas 2: le nom et le contenu sont calculés

– Constructeurs d'élément et d'attribut

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

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

Exemple:

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

Résultat :

```
<title publisher="Oreilly"/>
```

Expressions FLWOR

FLWOR = "For Let Where Order Return"
Rappelle l'idée du select-from-where-order-by de SQL

Voici un survol:

1. For : itération sur une liste de fragments XML
2. Let : association du résultat d'une expression à une variable
3. Where : condition de sélection
4. Order : tri des résultats
5. Return : expression à retourner

Expressions FLWOR: La boucle For

Syntaxe:

For $variable **in** exp_recherche **Return** exp_resultat

Associe à chaque $variable une valeur (fragment XML) trouvée pour exp_recherche

```xml
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
    <book category="COOKING">
        <title lang="en"> Everyday italian</title>
        <author>Glada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
    </book>
    <book category="CHILDREN">
        <title lang="en"> Harry Potter</title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>
    <book category="WEB">
        <title lang="en"> Learning XML</title>
        <author>Erik T.Ray</author>
        <year>2003</year>
        <price>39.95</price>
    </book>
</bookstore>
```

books.xml

```
for $t in doc("books.xml")//book/author/text()
return
<Auteur>
{
  $t
}
</Auteur>
```

```
<Auteur>Glada De Laurentiis</Auteur>
<Auteur>J K. Rowling</Auteur>
<Auteur>Erik T.Ray</Auteur>
```

Expressions FLWOR: La boucle For

employes.xml

```
<?xml version="1.0" >
<employes>
   <employe>
      <nom>Salem</nom>
      <prenom>Ail</prenom>
      <date_naissance>23/09/1958
      </date_naissance>
   </employe>
   <employe>
      <nom>Ammar</nom>
      <prenom>Fatima</prenom>
      <date_naissance>23/12/1975
      </date_naissance>
   </employe>
</employes>
```

La requête FLWOR suivante :

```
for $b in doc("employes.xml")//employe
return
<Infos>
{
$b/prenom,
$b/date_naissance
}
</Infos>
```

va renvoyer le résultat suivant :

```
<Infos>
  <prenom>Ail</prenom>
  <date_naissance>23/09/1958</date_naissance>
</Infos>
<Infos>
  <prenom>Fatima</prenom>
  <date_naissance>23/12/1975</date_naissance>
</Infos>
```

Expressions FLWOR: La boucle For

Exemple de Requête

```
for $x in (1 to 5)
return
 <test>
 {
  $x
 }
</test>
```

Résultat

<test>1</test>
<test>2</test>
<test>3</test>
<test>4</test>
<test>5</test>

Expressions FLWOR: Affectation Let

Syntaxe: **LET** $variable **:=** expression-Xpath

Permet d'associer à une $variable une liste de noeuds résultants de l'évaluation d'une expression-Xpath

Exemple :

let $R := doc("bib.xml")//book
return(count($R))

Résultat: 2

Exemple

let $x:=4
return($x*25)

Résultat: 100

```
<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>
```

Différence entre le FOR et le LET

- La clause **For** $var **in** exp donne à $var une valeur à chaque itération retournée par exp

- La clause **Let** $var **:=** exp range dans $var la séquence “entière” retournée par exp

Différence entre le FOR et le LET

- **For:**
  - **for $x in doc("employes.xml")//employe** génère un lien de chaque employé vers $x pour chaque élément dans l'entreprise
- **Let:**
  - **let $x := doc("employes.xml")//employe** génère un seul lien $x représentant ici l'ensemble des employés de l'entreprise

Expression FLWOR : difference entre F et L

```
for $i in (2,3)
return <res> {2 * $i} </res>
```

```
<res>4</res>
<res>6</res>
```

```
for $i in (2,3)
return <res> {2 * 1} </res>
```

```
<res>2</res>
<res>2</res>
```

```
let $i := (2,3)
return <res> {2 * 1} </res>
```

```
<res>2</res>
```

```
let $i := (2,3,4)
return <res> {count($i)} </res>
```

```
<res>3</res>
```

Expressions: La boucle For Let

bib.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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>Morgzan Kaufmann</publisher>
        <price>35.00</price>
    </book>
</bib>
```

Exemple de requête :

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

Résultat :

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

Expressions FLWOR: La clause Where exp

- **Syntaxe:** **Where** expression-Xpath

Cette clause est facultative.

Elle permet de filtrer le résultat par rapport au résultat booléen de l'expression expression-Xpath

Expressions: La clause Where exp

books.xml

```
<bookstore>
    <book category="COOKING">
            <title lang="en">Everyday Italian
            </title>
            <author>Giada De Laurentiis
            </author>
            <year>2005</year>
            <price>30.00</price>
    </book>
    <book category="CHILDREN">
            <title lang="en">Harry Potter
            </title>
            <author>J K. Rowling</author>
            <year>2005</year>
            <price>29.99</price>
    </book>
    <book category="WEB">
            <title lang="en">Learning XML</title>
            <author>Erik T. Ray</author>
            <year>2003</year>
            <price>39.95</price>
    </book>
</bookstore>
```

Requête :

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
return $x/title
```

Résultat :

```
<title lang="en">Learning XML</title>
```

Expressions FLWOR: La clause Order By

Syntaxe:

expr1 **order by** expr2 (ascending | descending)

La clause Order By est facultative.

Permet de trier les éléments de la séquence retournée par l'expression expr1 selon les valeurs retournées par expr2

Expressions: La clause Order By

bib.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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>Morgzan Kaufmann</publisher>
        <price>35.00</price>
    </book>
</bib>
```

Requête:

```
<livres>
{ for $b in doc("bib.xml")//book
order by $b/price descending
return  $b/publisher
}
</livres>
```

Résultat

```xml
<livres>
  <publisher>Morgzan Kaufmann</publisher>
  <publisher>Oreilly</publisher>
</livres>
```

Expressions: La clause Group By

```xml
<order num="00299432" date="2015-09-15" cust="0221A">
  <item dept="WMN" num="557" quantity="1" color="navy"/>
  <item dept="ACC" num="563" quantity="1"/>
  <item dept="ACC" num="443" quantity="2"/>
  <item dept="MEN" num="784" quantity="1" color="white"/>
  <item dept="MEN" num="784" quantity="1" color="gray"/>
  <item dept="WMN" num="557" quantity="1" color="black"/>
</order>
```

order.xml

```
for $item in doc("order.xml")//item
let $d := $item/@dept
group by $d
order by $d
return
<department code="{$d}">
{
for $i in $item
order by $i/@num
return $i
}
</department>
```

Résultat

```xml
<department code="ACC">
  <item dept="ACC" num="443" quantity="2"/>
  <item dept="ACC" num="563" quantity="1"/>
</department>
<department code="MEN">
  <item dept="MEN" num="784" quantity="1" color="white"/>
  <item dept="MEN" num="784" quantity="1" color="gray"/>
</department>
<department code="WMN">
  <item dept="WMN" num="557" quantity="1" color="navy"/>
  <item dept="WMN" num="557" quantity="1" color="black"/>
</department>
```

Expressions FLWOR: Sélection par satisfies

✓***some*** *$var* ***in*** *expr1* ***satisfies*** *expr2*

Il existe au moins un nœud retourné par l'expression *expr1 qui satisfait l'expression expr2*

✓***every*** *$var* ***in*** *expr1* ***satisfies*** *expr2*

Tous les noeuds retournés par l'expression *expr1 satisfont l'expression expr2*

Expressions: Sélection par satisfies

Catalog.xml

```
<?xml version="1.0"?>
<catalog>
    <product dept="WMN">
        <number>557</number>
        <name language="en">Linen Shirt</name>
    </product>
    <product dept="ACC">
        <number>563</number>
        <name language="en">Ten-Gallon Hat</name>
    </product>
    <product dept="ACC">
        <number>443</number>
        <name language="en">Golf Umbrella</name>
    </product>
    <product dept="MEN">
        <number>784</number>
        <name language="en">Rugby Shirt</name>
    </product>
</catalog>
```

Requête :

```
for $p in doc("catalog.xml")//product
where some $d in $p/@dept
satisfies $d="ACC"
return $p/name
```

Résultat :

```
<name language="en">Ten-Gallon Hat</name>
<name language="en">Golf Umbrella</name>
```

Expressions de Test: If- then-else

XQuery prend en charge l'instruction if-then-else conditionnelle suivante :

***if* (<expression1>)**

***then* <expression2>**

***else* <expression3>**

Suivant la valeur booléenne effective de expression1, expression2, ou expression3 est évaluée.

Exemple: If- then-else

books.xml

```
<bookstore>
    <book category="COOKING">
        <title lang="en">Everyday Italian
        </title>
        <author>Giada De Laurentiis
        </author>
        <year>2005</year>
        <price>30.00</price>
    </book>
    <book category="CHILDREN">
        <title lang="en">Harry Potter
        </title>
        <author>J K. Rowling</author>
        <year>2005</year>
        <price>29.99</price>
    </book>
    <book category="WEB">
        <title lang="en">Learning XML</title>
        <author>Erik T. Ray</author>
        <year>2003</year>
        <price>39.95</price>
    </book>
</bookstore>
```

Requête :

```
for $x in doc("books.xml")//book
return if ($x/@category="CHILDREN")
then
<Enfant>{($x/title)}</Enfant>
else
<Adulte>{($x/title)}</Adulte>
```

Résultat :

```
<Adulte>
  <title lang="en">Everyday italian</title>
</Adulte>
<Enfant>
  <title lang="en">Harry Potter</title>
</Enfant>
<Adulte>
  <title lang="en">Learning XML</title>
</Adulte>
```

Fonctions prédéfinies

En plus de la fonction **doc**
XQUERY possède des fonctions analogues à celles de SQL:
distinct-values: Élimination des doubles
min,
max,
count,
sum,
avg

XQUERY hérite des fonctions XPATH concernant les chaînes de caractères :
concat, string-length, starts-with, end-with, substring, upper-case, lower-case.

Exemple: Fonctions prédéfinies

books.xml

```
<bookstore>
    <book category="COOKING">
            <title lang="en">Everyday Italian
            </title>
            <author>Giada De Laurentiis
            </author>
            <year>2005</year>
            <price>30.00</price>
    </book>
    <book category="CHILDREN">
            <title lang="en">Harry Potter
            </title>
            <author>J K. Rowling</author>
            <year>2005</year>
            <price>29.99</price>
    </book>
    <book category="WEB">
            <title lang="en">Learning XML</title>
            <author>Erik T. Ray</author>
            <year>2003</year>
            <price>39.95</price>
    </book>
</bookstore>
```

Requête :

```
let $b := doc("books.xml")//book
let $avg := avg( $b//price )
return $b[price > $avg]
```

Résultat :

```
<book category="WEB">
  <title lang="en">Learning XML</title>
  <author>Erik T.Ray</author>
  <year>2003</year>
  <price>39.95</price>
</book>
```

Les livres qui sont plus chers que la moyenne des prix.

Déclaration de Fonction

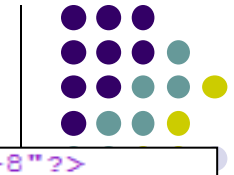
Syntaxe

declare function nom signature {code} ;

nom : soit dans un espace de noms déclaré soit préfixé par “local” (espace de noms des fonctions par défaut) ;
signature : (liste des paramètres, éventuellement typés)
code : n'importe quelle expression XQuery.

```
declare function local:discountPrice(
$price as xs:decimal?,
$discount as xs:decimal?) as xs:decimal?
{
let $ actualDiscount:= ($price * $discount) div 100
return ($price - $actualDiscount)
};
let $prod := doc("price.xml")//prod[1]
return local:discountPrice($prod/price, $prod/discount)
```

Jointure: Utilisation de plusieurs documents



bib.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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>Morgzan Kaufmann</publisher>
        <price>35.00</price>
    </book>
</bib>
```

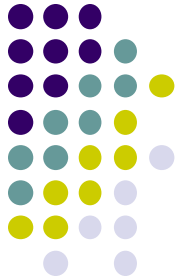
```
for $b in doc("bib.xml")//book
return element livre
  {
   attribute titre {$b/@title},
   for $a in $b/author
   return element auteur
     {
      attribute nom {$a/la},
      for $p in doc("adr.xml")//person
      where $a/la = $p/name
      return attribute institut {$p/institution}
     }
  }
```

adr.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<adresses>
    <person>
        <name>Amann</name>
        <country>France</country>
        <institution>CNAM</institution>
    </person>
    <person>
        <name>Scholl</name>
        <country>France</country>
        <institution>INRIA</institution>
    </person>
    <person>
        <name>Voisard</name>
        <country>Germany</country>
        <institution>FU Berlin</institution>
    </person>
    <person>
        <name>Rigaux</name>
        <country>France</country>
        <institution>CNRS</institution>
    </person>
</adresses>
```

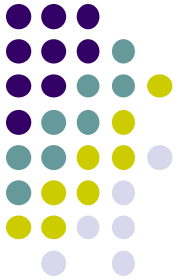
```xml
<livre titre="Comprendre XSLT">
  <auteur nom="Amann" institut="CNAM"/>
  <auteur nom="Rigaux" institut="CNRS"/>
</livre>
<livre titre="Spatial Databases">
  <auteur nom="Rigaux" institut="CNRS"/>
  <auteur nom="Scholl" institut="INRIA"/>
  <auteur nom="Voisard" institut="FU Berlin"/>
</livre>
```

XQuery Vs XSLT



- XQuery et XSLT sont deux langages dessinés pour interroger et manipuler des documents XML
- Ces deux langages utilisent le même modèle de données et possèdent le même ensemble de fonctions
- XSLT, plus pour transformer
- XQuery, plus pour interroger que pour transformer

Deux façons de faire, même résultat



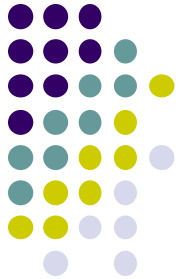
Requête XQuery

```
<ul type="square">{
                    for $product in doc("catalog.xml")/catalog/product[@dept = 'ACC']
                    order by $product/name
                    return <li>{data($product/name)}</li>
                   }
</ul>
```

Équivalent XSLT

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:template match="/">
            <ul type="square">
                <xsl:for-each select="catalog/product[@dept = 'ACC']">
                    <xsl:sort select="name"/>
                    <li><xsl:value-of select="name"/></li>
                </xsl:for-each>
            </ul>
        </xsl:template>
</xsl:stylesheet>
```

S’ils sont si semblables pourquoi deux langages ?



- XSLT est un langage de transformation optimisé pour reformater un document XML complet
  - Généralement, les documents sont entièrement chargés en mémoire et traités en une ou plusieurs passes
- XQuery est un langage optimisé pour sélectionner des fragments de données, possiblement dispersés dans plusieurs documents sauvegardés dans une base de données
  - Les documents sont généralement brisés en plus petits morceaux et indexés pour en faciliter l’accès
  - Les requêtes peuvent retrouver des fragments du document XML sans avoir à le charger entièrement en mémoire

XQuery vs SQL

- XQuery exploite plusieurs idées du SQL
  - Sélection de données, jointures, fonctions, tris, etc.
  - Certains concepteurs du SQL ont participé à la création de XQuery.

- Cependant, le XQuery ne replacera pas le SQL
  - SQL est le langage d'interrogation pour les données hautement structurées.
  - XQuery est le langage d'interrogation pour les données moins structurées.
  - IL est possible de faire des modifications de données avec SQL.
  - XQuery 1.0 ne propose que d'exploiter les données. Ce point a été corrigé avec le XQuery Update Facility (XQUF).

Références bibliographiques

Titre :	BASES DE DONNEES XQuery pour interroger des donnees XML
Titre original :	ELements du langage et mise en oeuver cours et exercices corriges
Type de document :	texte imprimé
Auteurs :	JACQUES LE MAITRE, Auteur ; EMMANAUL BRUNO, Auteur
Editeur :	ellipses
Année de publication :	2013
ISBN/ISSN/EAN :	978-2-7298-8348-5
Langues :	Français (*fre*)

Titre :	XML cours et exercices : modélisation.schéma et DTD.design patterns.XSLT.DOM .relaxNG.XPath.SOAP.XQuery.XSL-FO.SVG.exist/2e EDITION
Type de document :	texte imprimé
Auteurs :	ELEXANDRE BRILLANT, Auteur
Editeur :	EYROLLES
Année de publication :	2010
ISBN/ISSN/EAN :	978-2-212-12691-4
Langues :	Français (*fre*)

Titre :	MODELISATION XML
Type de document :	texte imprimé
Auteurs :	ANTOINE LONJON, Auteur ; JEAN-JACQUES THOMASSON, Auteur
Editeur :	EYROLLES
Année de publication :	2006
ISBN/ISSN/EAN :	978-2-212-11521-5
Langues :	Français (*fre*)

Implémentation & liens

- BaseX, (Open Source Database for XML Documents) http://basex.org/
- Galax, implémentation en Ocaml : http://www.galaxquery.org/
- Qizx/open, interpréteur open source écrit en Java :http://www.qizx.com/
- Saxon, processeur open source XQuery et XSLT http://www.saxonica.com/
- eXist, (Open Source Database for XML Documents) http://exist.sourceforge.net/
- MonetDB? (Open Source Database System) http://monetdb.cwi.nl/projects/monetdb/XQuery/index.html