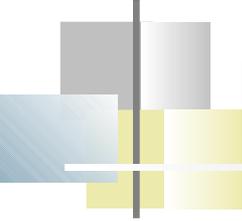


Documents XML

Description - Manipulation

---



# Programme du cours

---

- Introduction
  - Documents structurés, XML, Bases de données
- Documents XML : description et manipulation
  - Description de documents
  - Typage - *Xschema*
  
  - Modélisation -- *XDM*
  - Localisation de composants XML -- *XPath*
  
  - Interrogation de documents XML -- *XQuery*

The logo consists of a vertical line on the left, with two yellow squares above it and two grey squares below it. To the right of this graphic, the word "XQuery" is written in a blue, sans-serif font.

# XQuery

---

<http://www.w3.org/TR/xquery>

<http://www.w3.org/TR/xquery-semantic>

<http://www.w3.org/TR/xpath-functions>

Cours en ligne de Bernd Amman LIP6, Jacques Le Maitre LSIS,  
Georges Gardarin, Prism, Benjamin Nguyen Prism



# 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)
    - À partir de plusieurs propositions (depuis 1994)
  - Véritable langage de programmation, très puissant

# Document XML

<guide>...

  <itinéraire numéro="6">

    <nom>Col d'Izoard</nom>

    <commentaire>Cette montée pédestre, au haut lieu du cyclotourisme, vous évite les lacets de la route et, du fond du vallon du torrent d'Izoard, vous permet d'admirer, loin des foules et de la motorisation, les rochers déchiquetés de la Casse Déserte (cagneules).</commentaire>

    <cotation>1</cotation>

    <départ>

      <lieu>Brunissard</lieu>

      <altitude>1760</altitude>

    </départ>

    <arrivée>

      <lieu>Col d'Izoard</lieu>

      <altitude>2360</altitude>

    </arrivée>

    <temps unité="heure">2</temps>

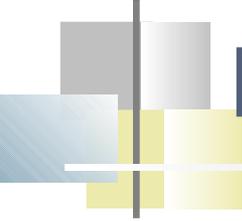
    <montée>De <lieu>Brunissard<lieu> remonter (NW) la route de l'Izoard, <lieu>D 902</lieu>, jusqu'au premier virage de l'entrée des bois ...</montée>

    <descente> Sur <lieu>Brunissard<lieu> par l'itinéraire de montée. Sur Cervières, par le <lieu>refuge d'Izoard<lieu> et la route <lieu>D 902</lieu>, sur le versant Nord du col.</descente>

  </itinéraire>

...

</guide>

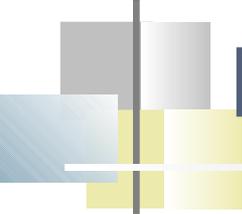


# Exemple de requête

---

*Numéro et nom des itinéraires au départ de Brunissard*

```
<réponse> (1)
  {
    for $i in $guide/guide/itinéraire (2)
    where $i/départ/lieu = 'Brunissard' (3)
    return (4)
      <itinéraire> (5)
        {<numéro>$i/@numéro</numéro>} (6)
        {$i/nom} (7)
      </itinéraire> (8)
    }
  </réponse> (9)
```

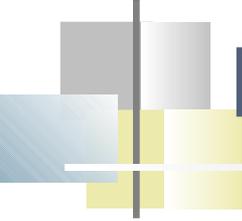


## Exemple de requête

---

*Numéro et nom des itinéraires au départ de Brunissard*

```
<réponse> (1)
  {
    for $i in $guide/guide/itinéraire (2)
    where $i/départ/lieu = 'Brunissard' (3)
    return (4)
      <itinéraire> (5)
        {<numéro>$i/@numéro</numéro>} (6)
        {$i/nom} (7)
      </itinéraire> (8)
    }
  </réponse> (9)
```



# Résultat

---

*Numéro et nom des itinéraires au départ de Brunissard*

```
<réponse>
```

```
...
```

```
<itinéraire>
```

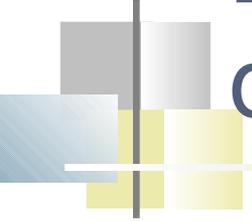
```
  <numéro>6</numéro>
```

```
  <nom>Col d'Izoard</nom>
```

```
</itinéraire>
```

```
...
```

```
</réponse>
```

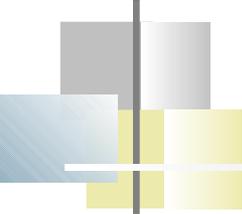


# Le FLWR intuitivement – Forme de base

---

**for**  $i$  **in**  $s$  **where**  $cond$  **return**  $i'$

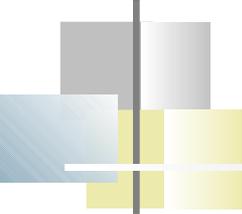
- L'opérateur *for...where...return* itère sur les items  $i$  de la séquence  $s$  et construit une séquence de nouveaux items  $i'$ 
  - Pour chaque  $i$  de  $s$ , si  $cond$  est vérifiée,  $i'$  est construit
- Analogie forte avec SQL



# Expressions XQuery

---

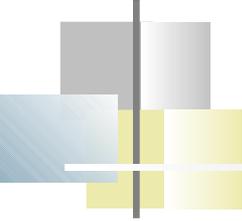
- Une requête XQuery est une composition d'expressions
- Chaque expression a une valeur ou retourne une erreur
- La valeur d'une expression est une instance du modèle XDM
  - une séquence de 0 ou plusieurs items
- Les expressions n'ont pas d'effets de bord (par exemple, de mise à jour)



# Expressions XQuery

---

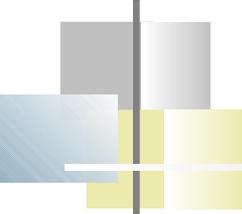
- Une expression est construite à partir :
  - De constantes littérales (chaînes, nombres)
  - De noms de variable
  - D'opérateurs (sur les nœuds, arithmétiques, etc.)
  - De chemins de localisation
  - De fonctions prédéfinies etc.



# Modèle de données

---

- Commun à XPath et XQuery
- Une valeur (une instance du modèle) est une séquence ordonnée d'items
- Un item est soit un nœud, soit une valeur atomique
- Une valeur atomique est une instance d'un type atomique (cf.XSchema)
- Chaque nœud et chaque valeur a un type

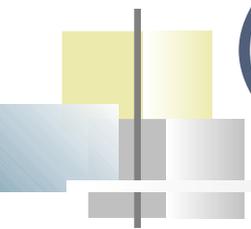


# Modèle de données

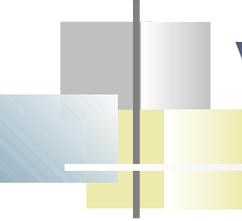
---

- Une séquence peut être vide (séquence vide)  
( )
- Il y a équivalence entre un item et une séquence singleton (une séquence de 1 item)  
 $12 = (12)$
- Les séquences sont plates : elles ne sont pas imbriquées  
 $(12, (3, 5), \langle \text{exemple}/\rangle, \text{"exemple"}) =$   
 $(12, 3, 5, \langle \text{exemple}/\rangle, \text{"exemple"})$
- Les séquences sont triées  
 $(3, 5)$  est différent de  $(5, 3)$

# Expressions simples (requêtes)



---



# Valeurs atomiques

---

- Valeurs atomiques

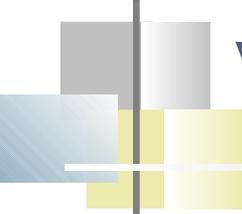
nombres 12 3.14 2.602e-19

chaînes "salut"

- Valeurs construites

fn:true()

xs:date("2008-03-03")



# Variable

---

- Référence à une variable  $v$

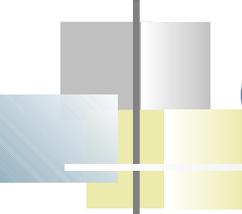
`$v`

Retourne la valeur la plus récemment liée à la variable  $v$

- Déclaration de variables globales

```
declare variable $v := <exp>
```

```
declare variable $doc := doc('docs/xmpbib.xml')
```



# Quelques premiers opérateurs

---

- Opérateurs arithmétiques

`+ - * div (attention / réservé) idiv mod`  
`1+2                    3.4-6.5                    $x mod 2`

- Opérateurs de comparaison pour valeurs atomiques, nœuds et séquences

`5 gt 7.5 → FALSE`

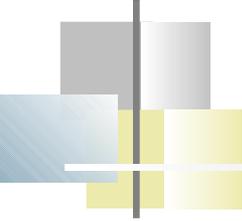
La comparaison est effectuée après la conversion de 5 en `xs:float`

- Opérateurs booléens

`$x=2 and $y=4`

`$x=2 or $y=4`

`fn:not(1 and 1) → FALSE`



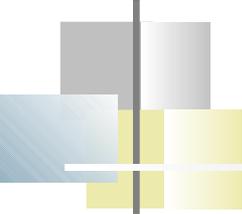
# Premiers opérateurs

---

- Séquences d'entiers  $exp_1$  to  $exp_2$ 
  - $()$  si la valeur de  $exp_1$  ou de  $exp_2$  égale  $()$
  - si les valeurs de  $exp_1$  et  $exp_2$  sont convertibles en deux entiers  $n_1$  et  $n_2$ 
    - $()$  si  $n_1 > n_2$
    - séquence des entiers de  $n_1$  à  $n_2$  si  $n_2 \geq n_1$

3 to 7      ->      3, 4, 5, 6, 7

13 to 7     ->       $()$



# Premiers opérateurs

---

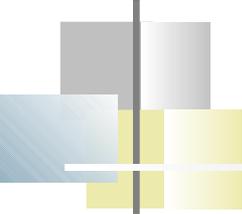
- Opérations sur les séquences

- concaténation  $exp_1, exp_2$

séquence constituée des items de la séquence valeur de  $exp_1$  suivis des items de la séquence valeur de  $exp_2$

$1+2, 4-2, 3*2 \rightarrow 3, 2, 6$

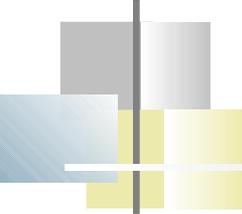
$((9, 1), 5, (8, 2)) \rightarrow (9, 1, 5, 8, 2)$



# Expressions complexes

---

- Expressions de chemin (XPath 2.0)
- Constructeurs de nœuds
- Opérateurs
- Expressions FLWR (`for-let-where-return`)
- Expressions conditionnelles - tests (`if-then-else`)



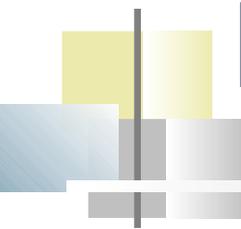
# Expressions complexes

---

## ■ Fonctions

- `nom (exp1, ..., expn)`
- **Racines** : `collection ("url") doc ("url")`
  - `fn:doc` retourne le nœud racine du document dont l'URL est donné en argument  

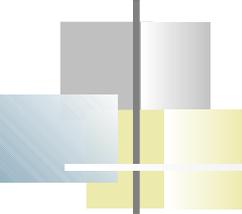
```
declare variable $bib := fc:doc(xmpbib.xml)
```
  - `fn:collection` retourne la séquence de nœuds contenus dans la ressource dont l'URL est donné en argument
- Fonctions prédéfinies
  - XQuery 1.0 and XPath 2.0 functions and operators
- Fonctions définies par les utilisateurs



# Expressions de chemin

---

Valeur : une séquence de  
nœuds

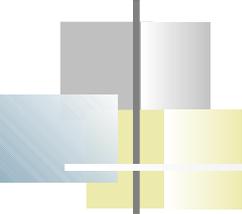


# Un cas d'usage : sa DTD

---

```
<!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 )>
```

```
<bib>
  <book year="1999">
    <title>The Economics of Technology and Content for Digital TV</title>
    <editor>
      <last>Gerbarg</last><first>Darcy</first>
      <affiliation>CITI</affiliation>
    </editor>
    <publisher>Kluwer Academic Publishers</publisher>
    <price>129.95</price>
  </book>
  <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 the Unix environment</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </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>
</bib>
```



# Expressions de chemin

---

## *Requête*

```
doc("biblio.xml")//author
```

```
$bib//author
```

## *Réponse*

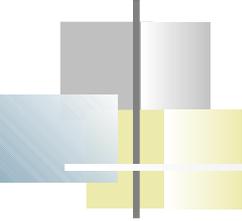
```
<author><last>Stevens</last><first>W.</first></author>
```

```
<author><last>Stevens</last><first>W.</first></author>
```

```
<author><last>Abiteboul</last><first>Serge</first></author>
```

```
<author><last>Buneman</last><first>Peter</first></author>
```

```
<author><last>Suciu</last><first>Dan</first></author>
```



# Expressions de chemin

---

```
doc("biblio.xml")/bib/book/author
```

```
doc("biblio.xml")/bib/book[1]/publisher
```

```
doc("biblio.xml")//book/(author,publisher)
```

```
doc("biblio.xml")//book[position() lt last()]
```

les nœuds restants sont triés selon l'ordre du document

# Dans une expression de chemin

---

```
doc("biblio.xml")//book[2]/(publisher union title)
```

```
doc("biblio.xml")//book[2]/(title union publisher)
```

```
<title>TCP/IP Illustrated</title>
```

```
<publisher>Addison-Wesley</publisher>
```

les nœuds restants sont triés selon l'ordre du document

# Dans une expression de chemin

```
doc("biblio.xml")//book[2]/(publisher union title)
doc("biblio.xml")//book[2]/(title union publisher)
doc("biblio.xml")//book[2]/(publisher,title)
```

```
<title>TCP/IP Illustrated</title>
```

```
<publisher>Addison-Wesley</publisher>
```

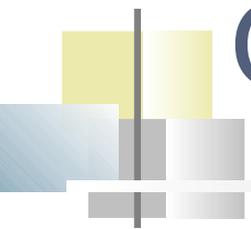
```
for $b in doc("biblio.xml")//book
```

```
return
```

```
($b/publisher,$b/title)
```

```
<publisher>Addison-Wesley</publisher>
```

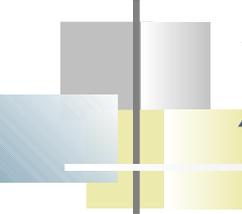
```
<title>TCP/IP Illustrated</title>
```



# Constructeurs de nœuds

---

# Constructions de nœuds

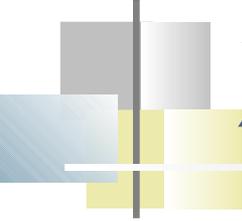


## XML

---

- Un constructeur produit un nouveau nœud
  - Il existe des constructeurs pour chaque sorte de nœud
    - élément et attribut
- Un résultat peut contenir des fragments prédéfinis et des expressions à évaluer
  - parties connues (constructeurs directs)
  - parties calculées par expressions (constructeurs calculés)
- A l'exécution
  - les parties connues sont copiées
  - les parties calculées sont évaluées

# Constructions de nœuds



## XML

---

Le nom du nœud est connu, son contenu est calculé par une expression

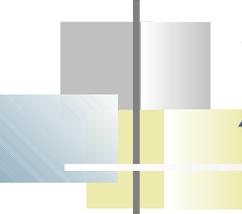
*Requête:*

```
<auteurs>
    { doc("biblio.xml") //book[4]/author/last  }
</auteurs>
```

*Résultat:*

```
<?xml version="1.0"?>
<auteurs>
  <last>Abiteboul</last>
  <last>Buneman</last>
  <last>Suciu</last>
</auteurs>
```

# Constructions de nœuds



## XML

---

- Le nom du nœud et son contenu sont calculés par des expressions

```
element {expression_nom}{expression_contenu}
```

```
attribute{expression_nom}{expression_contenu}
```

### *Requête:*

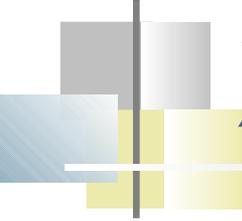
```
<auteurs>
```

```
    {doc("biblio.xml")//book[4]/author/last }
```

```
</auteurs>
```

```
element auteurs {doc("biblio.xml")//book[4]/author/last}
```

# Constructions de nœuds



## XML

---

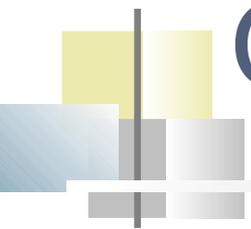
### *Requête:*

element

```
{ doc("biblio.xml")//book[1]/fn:name(*[1]) }
{
  attribute
  { doc("biblio.xml")//book[1]//fn:name(*[3]) }
  {doc("biblio.xml")//book[1]/*[3]}
}
```

### *Résultat:*

```
<title publisher="Kluwer Academic Publishers"/>
```



# Opérateurs (suite)

---

# Opérateurs

## Séquences de nœuds

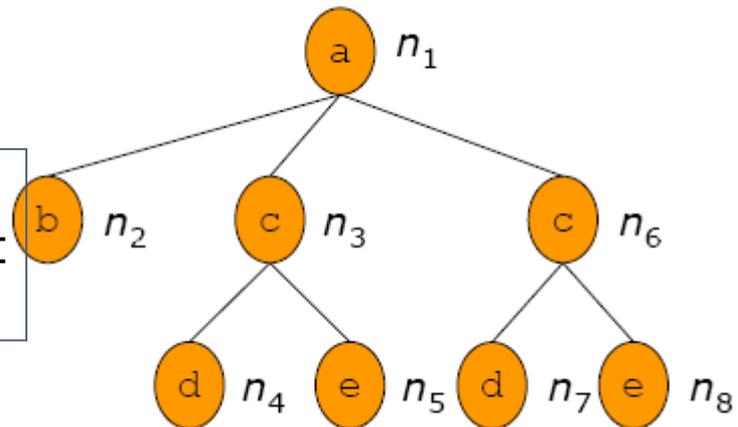
- Union intersection différence
- Si le nœud contexte est la racine du fragment

$c/e$  union  $c/*$   $\Rightarrow n_4, n_5, n_7, n_8$

$c/d$  intersect  $c/*$   $\Rightarrow n_4, n_7$

$c/*$  except  $c/d$   $\Rightarrow n_5, n_8$

Doublons éliminés  
Nœuds rangés dans l'ordre du document  
(sauf contre ordre dans le prologue)



# Différence de séquences de nœuds dans un constructeur

---

## Requête:

```
<livre>
```

Tous les sous-éléments sauf les auteurs:

```
{doc("biblio.xml")//book[2]/(* except author) }
```

```
</livre>
```

## Résultat:

```
<livre>
```

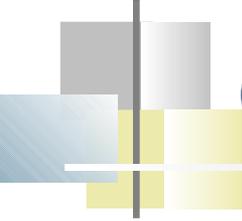
Tous les sous-éléments sauf les auteurs:

```
<title>TCP/IP Illustrated</title>
```

```
<publisher>Addison-Wesley</publisher>
```

```
<price>65.95</price>
```

```
</livre>
```



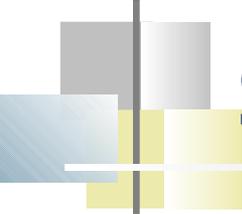
# Dans une expression de chemin

---

```
doc("biblio.xml")//book[2]/(title union publisher)
```

```
<title>TCP/IP Illustrated</title>
```

```
<publisher>Addison-Wesley</publisher>
```



# Concaténation de séquences de noeuds

---

## Requête:

```
<livre>
```

Le prix suivi du titre:

```
{doc("biblio.xml")//book[2]/price,  
  doc("biblio.xml")//book[2]/ title) }
```

```
</livre>
```

## Résultat:

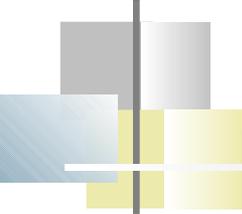
```
<livre>
```

Le prix suivi du titre:

```
<price> 65.95</price>
```

```
<title>TCP/IP Illustrated</title>
```

```
</livre>
```



# Transformation nœud -> valeur

---

Requête:

```
"Les auteurs du quatrième livre sont",  
doc("biblio.xml")//book[4]/author/fn:string(last)
```

Résultat:

```
" Les auteurs du quatrième livre sont", "Abiteboul",  
"Buneman", "Suciu"
```

# Comparaison de valeurs atomiques

eq  
ne  
lt  
le  
gt  
ge

*Requête:*

```
doc("biblio.xml")//book/author[last eq "Suciu"]
```

*Résultat:*

```
<author><last>Suciu</la><first>Dan</first></author>
```

*Requête:*

```
doc("biblio.xml")//book[author/last eq "Suciu"]
```

*Résultat:*

**ERROR**

L'expression `author/last` dans le prédicat ne retourne pas une valeur

atomique mais une séquence de valeurs

# Comparaison de séquences

=  
!=  
<=  
<  
>=  
>

$exp_1 = exp_2$

s'il existe au moins un item dans la valeur de  $exp_1$  qui est égal à un item de la valeur de  $exp_2$

$5 \neq 7 \rightarrow \text{FALSE}$

$2 > (5, 10, 6) \rightarrow \text{FALSE}$

$(1, 2) = (7, 2, 10, 7) \rightarrow \text{TRUE}$

# Comparaison de séquences

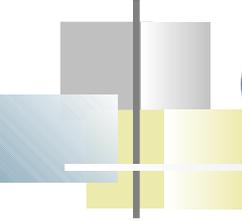
=  
!=  
<=  
<  
>=  
>

*Requête:*

```
Count (doc ("biblio.xml")  
        //book[author/last = ("Suciu", "Stevens")]
```

*Résultat:*

3



# Comparaison de nœuds

---

$n_1$  is  $n_2$

$n_1$  est identique à  $n_2$

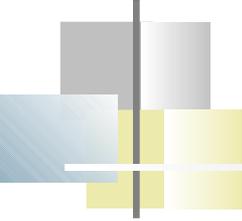
## *Requête:*

```
doc("biblio.xml")//book[author[3] is author[last()]]
```

## *Résultat:*

```
<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>
```

# Comparaison de nœuds par la position



$n_1 \ll n_2$

$n_1$  apparaît avant  $n_2$  dans le document (`true`)

$n_1 \gg n_2$

$n_1$  apparaît après  $n_2$  dans le document

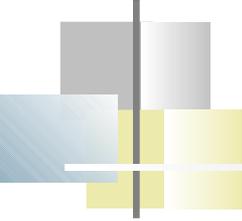
**Requête:**

```
<livre>
  {doc("biblio.xml")}
  //book[author[last="Abiteboul"] << author[last="Suciu"]]
  //title}
</livre>
```

**Résultat:**

```
<title>data on the web</title>
```

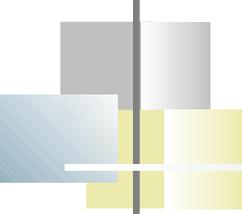




# FLWR

---

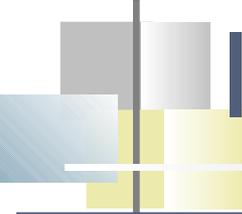
(`clause` `for` | `clause` `let`)  
(`clause` `where`)?  
(`clause` `order by`)?  
(`clause` `return`)



# For – let - return

---

- La clause `for $var in exp`  
affecte la variable `$var` successivement avec chaque item de la séquence retournée par `exp` (itérer sur une séquence)
- La clause `let $var := exp`  
affecte la variable `$var` avec la séquence “complète” retournée par `exp` (liaisons)
- La clause `return exp`  
permet de construire le résultat



# Illustrations simples

---

## Requête

```
for $i in (1, 2)
```

```
Return
```

```
<regle> {$i} </regle>
```

## Résultat

```
<regle> 1 </regle>
```

```
<regle> 2 </regle>
```

## Requête

```
for $i in (1, 2), $j in (5, 6)
```

```
return
```

```
<regle>
```

```
  {$i} fois {$j} vaut {$i * $j}
```

```
</regle>
```

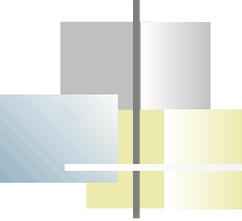
## Résultat

```
<regle> 1 fois 5 vaut 5</regle>
```

```
<regle> 1 fois 6 vaut 6</regle>
```

```
<regle> 2 fois 5 vaut 10</regle>
```

```
<regle> 2 fois 6 vaut 12</regle>
```



# Illustrations simples

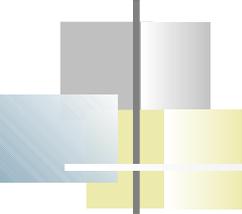
---

```
For $i in (1 to 3) {--FLWR incomplet--}
```

```
Let $j := (1 to $i)
```

Produit successivement :

- $i=1, j=1$
- $i=2, j=(1, 2)$
- $i=3, j=(1, 2, 3)$



# For – let – return

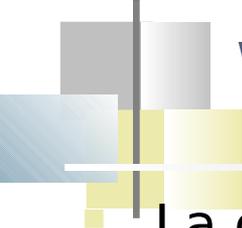
---

## Requête:

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

## Résultat:

```
<livre nb_auteurs="3">
  <author><last>Abiteboul</last><first>Serge</first></author>
  <author><last>Buneman</last><first>Peter</first></author>
  <author><last>Suciu</last><first>Dan</first></author>
</livre>
```



# Where

---

La clause *where* **exp**

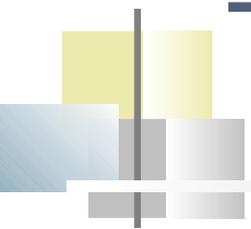
permet de filtrer un résultat par rapport au résultat booléen de l'expression *exp*

## *Requête:*

```
<livre>
  { for $a in doc("biblio.xml")//book
    where $a/author[1]/last eq "Abiteboul"
    return $a/title }
</livre>
```

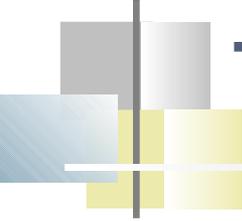
## *Résultat:*

```
<livre>
  <title>Data on the Web</title>
</livre>
```



# Tests

---

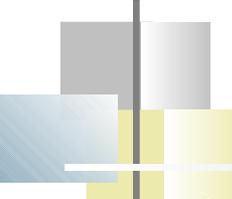


# Tests

---

```
if exp1 then exp2 else exp3
```

- teste la valeur booléenne de  $exp_1$
- Retourne la valeur de  $exp_2$  si elle est vraie
- Retourne la valeur de  $exp_3$  sinon



# Tests

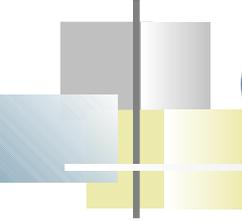
---

## Requête:

```
<livres>
  { for $b in doc("biblio.xml")//book
    return
      if ($b/@year > 1995)
        then <livre récent='vrai'> {$b/title} </livre>
        else <livre> {$b/title} </livre> }
</livres>
```

## Résultat:

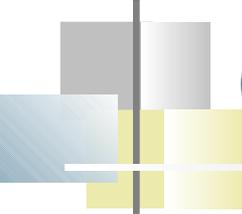
```
<livres>
  <livre récent="vrai">The Economics of Technology ...</livre>
  <livre>TCP/IP Illustrated</livre>
  <livre>Advanced Programming in the Unix environment</livre>
  <livre récent="vrai" >Data on the Web</livre>
</livres>
```



# Quantifications

---

- `some $var in exp1 satisfies exp2`  
il existe au moins un item retourné par l'expression `exp1` qui satisfait l'expression `exp2`
- `every $var in exp1 satisfies exp2`  
tous les items retournés par l'expression `exp1` satisfont l'expression `exp2`



# Quantifications

---

## *Requête*

```
every $b in doc("biblio.xml")//book  
satisfies $b/@year > 2000
```

## *Réponse*

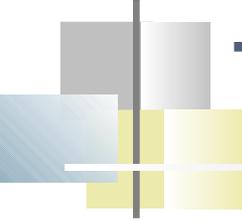
False

## *Requête*

```
some $b in doc("biblio.xml")//book  
satisfies $b/price < 50
```

## *Réponse*

true

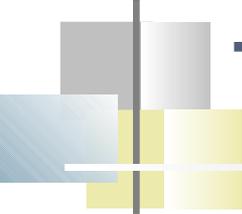


# Tri

---

- `exp1 order by exp2 (ascending | descending)`

trier les éléments de la séquence retournée par l'expression `exp1` par les valeurs retournées par l'expression `exp2`



# Tri

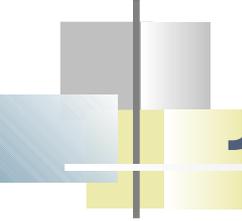
---

## *Requête*

```
<livres>
  { for $b in doc("biblio.xml")//book
    order by ($b/@year)
    return $b/title
  }
</livres>
```

## *Réponse*

```
<livres>
  <title>Advanced Programming in the Unix environment</title>
  <title>TCP/IP Illustrated </title>
  <title>The Economics of Technology and ... TV</title>
  <title>Data on the Web</title>
</livres>
```

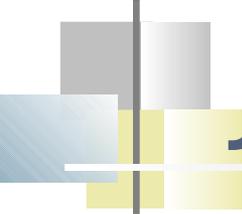


# Jointures

---

## Un nouveau document XML : un fichier d'adresses

```
<adresses>
  <personne>
    <nom>Abiteboul</nom>
    <pays>France</pays>
    <institution>INRIA</institution>
  </personne>
  <personne>
    <nom>Suciu</nom>
    <pays>USA</pays>
    <institution>Université de Washington</institution>
  </personne>
  <personne>
    <nom>Buneman</nom>
    <pays>Ecosse</pays>
    <institution> LFCS, School of Informatics </institution>
  </personne>
</adresses>
```

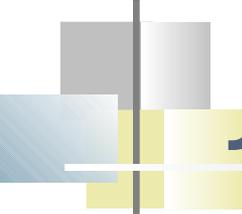


# Jointure : le résultat

---

## Résultat:

```
<livre titre="The Economics of Technology ..."/>
<livre titre="TCP/IP Illustrated">
  <auteur nom="Stevens"/>
</livre>
<livre titre="Advanced Programming in the Unix environment" >
  <auteur nom="Stevens"/>
</livre>,
<livre titre="Data on the Web">
  <auteur nom="Abiteboul" institution="INRIA" />
  <auteur nom="Buneman" institution=" LFCS, School of Informatics " />
  <auteur nom="Suciu" institution="Université de Washington" />
</livre>
```

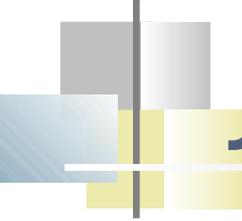


# Jointure : une requête

---

## Requête:

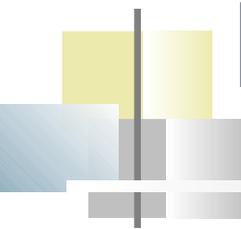
```
for $b in doc("biblio.xml")//book
return
  <livre titre='{$b/title}'>
    {for $a in $b/author
      return
        <auteur nom='{$a/last}'>
          {for $p in doc("addr.xml")//personne
            where $a/last = $p/nom
              return
                attribute institution {$p/institution}}
          }
    }
</livre>
```



# Jointure : une requête

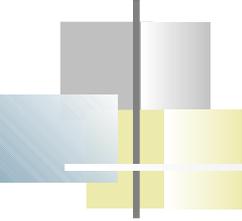
---

```
for $b in doc("biblio.xml")//book
  return
  element livre
    {(attribute titre {$b/title},
      for $a in $b/author
        return element auteur
          {
            (attribute nom {$a/last},
              for $p in doc("addr.xml")//personne
                where $a/last = $p/nom
                  return attribute institution {$p/institution})
          }
    )
  }
})}
```



# Fonctions

---



# Fonction avg

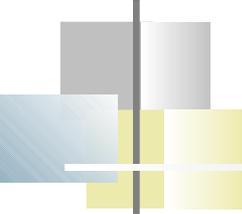
---

## Requête:

```
for $p in distinct-values(doc("biblio.xml")//publisher)
let $l := doc("biblio.xml")//book[publisher = $p]
return element publisher
      {attribute name {fn:string($p)},
       attribute avg_price { fn:avg($l/price) } }
```

## Résultat :

```
<publisher name="Kluwer Academic Publishers" avg_price="129.95"/>,
<publisher name="Addison-Wesley" avg_price="65,95"/>,
<publisher name="Morgan Kaufmann Publishers" avg_price="39.95"/>
```



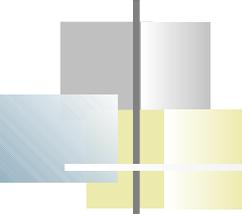
# Définition de fonctions

---

- XQuery permet à l'utilisateur de définir ses propres fonctions

```
declare function NombreAuteurs($b as element(book)) as
xs:integer
{
return count($b/author)
}
```

- Le résultat est de type xs:integer



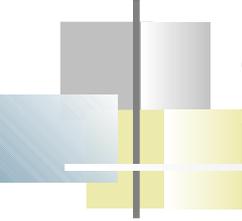
# Remarques

---

- XQuery est un langage sensible à la casse
  - Les mots clés sont en minuscules
- Tous les axes ne sont pas supportés

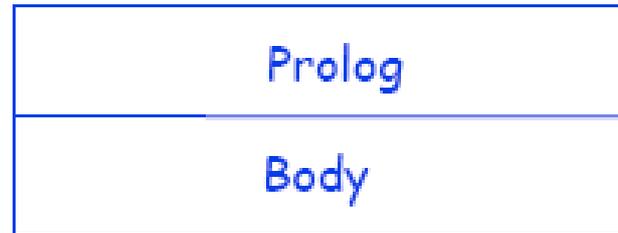
child  
descendant  
attribute  
self  
descendant-or-self  
parent

ancestor  
ancestor-or-self  
preceding  
preceding-sibling  
following  
following-sibling  
namespace

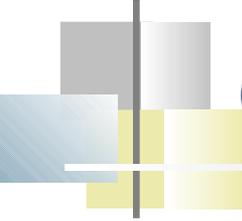


# Structure d'une requête

---



- Le Prologue contient en particulier:
  - Déclarations de Namespace
  - Importations de schémas
  - Définitions de fonctions
  - Déclarations de variable globales
- Le Corps contient:
  - Une expression qui définit le résultat de la requête



# Evaluation d'une expression

---

- Elle est évaluée dans
  - un contexte statique - analyse de l'expression - prologue
    - La requête est analysée et traduite en un arbre d'opérateurs
    - Un type statique est associé à chaque expression (s'appuie sur les types des opérandes - Règles d'inférence de type basées sur XML Schema)
    - Déclenche une erreur si les opérandes ne correspondent pas aux opérateurs
  - un contexte dynamique - évaluation de l'expression (focus)
    - Elle n'est lancée que si aucune erreur n'a été détectée
    - Elle consiste à évaluer l'arbre d'opérateurs
    - Un type dynamique est associé à chaque valeur calculée, qui peut être plus spécifique que le type statique de l'expression à partir de laquelle cette valeur est calculée