Module: XML et les bases de données

Xquery

Houda Chabbi Drissi

houda.chabbi@hefr.ch

Source-Livres:

- Xquery from the experts. ISBN 0-321-18060-7
- An Introduction to XML and Web Technologies. ISBN 0-321-26966-7

Exemple: livres.xml

```
<livre>
   <BD titre='Asterix le gaulois'>
     <scenariste>Gossini</scenariste>
     <dessin>Uderzo</dessin>
     <pub>Dargaux</pub>
     <prix>18.95</prix>
   </BD>
   <BD  annee='2001' titre='Titeuf'>
     <scenariste>Zep</scenariste>
     <dessin>Zep</dessin>
     <pub>Dargaux</pub>
    <prix>10.00</prix>
   </BD>
   <BD annee='1980' titre='Lucky luck'>
    <scenariste>Gossini</scenariste><scenariste>Moris</scenariste>
     <dessin>Moris</dessin>
     <pub>Dargaux</pub>
    <prix>10.00</prix>
   </BD>
</livre>
```

Besoins

- Extraire une information de ce fichier de BD (requête):
  - Titre / Scénariste etc…
- Prendre en compte le typage des éléments

Xpath

- Présenter cette information sous différent format:
  - *HTML / Text / XML*
- Construire une nouvelle information à partir de l'intiale (modification/ ajout):
  - *HTML / Text / XML*
- Faire du tri dans la présentation des éléments sélectionnés.

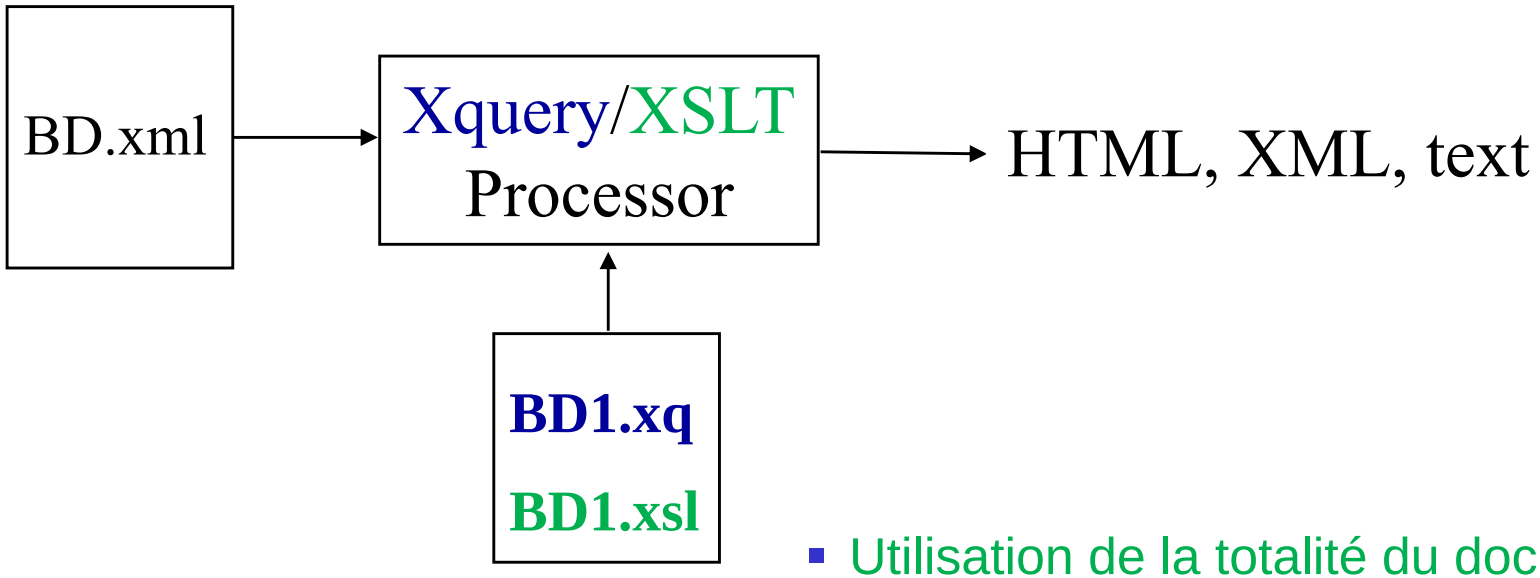
XSL

- Construire des requêtes complexes dans des SGBDs

XQuery

XSLT… Xquery?

- L'objectif attendu est le même (→ transformation) utilisant les éléments communs (Xpath, Xschema)



- Utilisation de parties du document XML
- Besoin d'extraction d'informations
- Pas de comportement par défaut.
  - →Optimisation des requêtes
  - →Fortement typé

XQuery → Déclaratif

- Utilisation de la totalité du document XML→ Règles par défaut parcours tout le document.
- Besoin de publication → XML/HTML
  - →Parcours d'arbre
  - →Pas fortement typé

XSL → Procédural / Règles

Hypothèses des concepteurs

Plan

1. Introduction et historique
2. Modèle de donnée Xquery: XDM
3. Syntaxe XQuery
4. Xquery avancé: Fonctions, jointure
5. Conclusion
6. Exercice

Interrogation de documents XML?

- Comment interroger des documents XML?

- Différentes approches:
  - SQL : XML « mappé » dans une BD relationnel ☹
  - Expressions XPath : extraction de fragments 😐
  - XSL: extraction + transformation (règles) 😐 ☺
  - Langage de requêtes pour XML ☺ ☺

Annexe: Historique des langages de requêtes

- Différentes propositions pour XML:
  - XOQL (Xyleme),
  - XML-QL,
  - XQL,
  - Lore, ...
- Enfin: XQuery : W3C Working Draft 02 May 2003

Annexe: XML et autres techno (inspirations)

- Nécessité d'interagir avec des technologies existantes :
  - XML + SQL ⇨ XQuery
  - XML + UML ⇨ XMI
  - XML + IHM ⇨ XUL

Caractéristiques du langage

- Besoin Langage pour interroger les données XML

- Caractéristiques:
  - Interrogation d'une hiérarchie (arbres XML)
  - Plutôt déclaratif vs XSL plutôt programmation
  - Permettant le retour de structures complexes avec des types différents

Caractéristiques des résultats attendus

- Des **structures complexes** avec des types différents:
  Différents fragments de XML atomiques ou pas

- Des **transformations structurelles**:
  Changer l'ordre d'une hiérarchie

- Impliquant des recherches qui tiennent compte des critères d'ordre:
  Trouver la BD apparaissant avant celle de Zep.

Solution? Un nouveau langage d'interrogation…

XQUERY

- Objectifs du groupe de travail XML Query :
  - trouver un **modèle de données** pour les documents XML,
  - **des opérateurs de requêtes** pour ce modèle de données,
  - et **un langage de requête basé** sur ces opérateurs de requêtes.

- XQuery est un langage XML de requête qui ressemble à SQL et qui est représenté sous la forme d'une expression.

Xquery 1.0-3.0

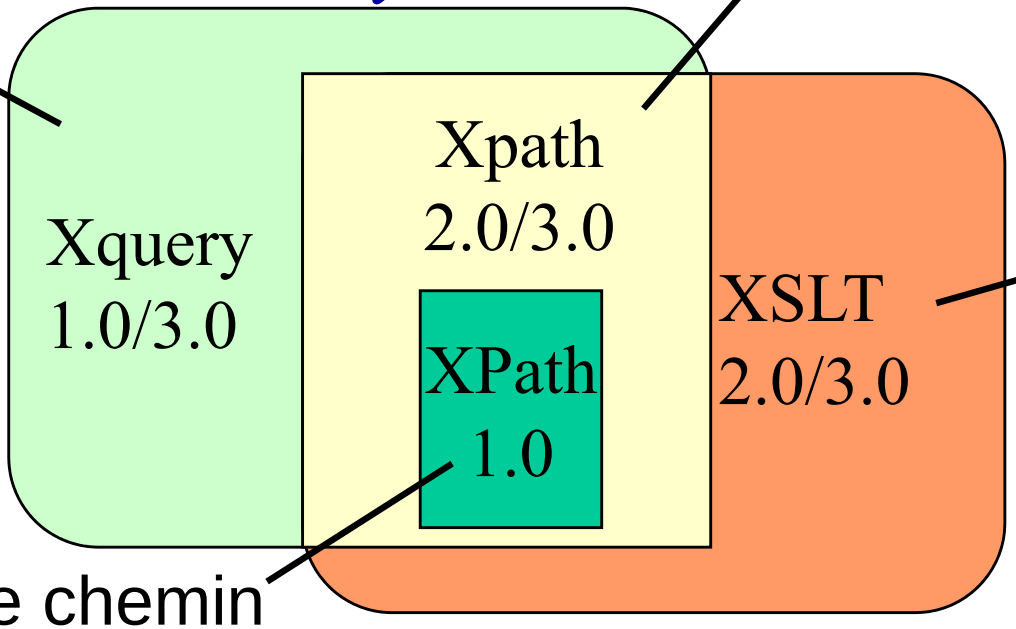
- Standard w3c (2006-2014): Basé XPath 2.0-3.0 considéré comme un sur-langage
  - Expressions XPath: extraction de fragments

- Règles XSLT: extraction + transformation
- XQuery : langage de requêtes pour XML

XQuery, XSLT et XPath

- **Expressions FLWOR**
- **Constructeurs XML**
- **UDF (user defined functions)**

- Expressions conditionnelles/ arirhmétiques / quantifiées
- Fonctions & opérateurs



- Templates
- Stylesheets…

- Expression de chemin
- Expressions de comparaison
- Quelques fonctions

Implémentations XQuery

- Open Source
  - BaseX
  - Saxon (Michael Kay)
- Commercial
  - BEA System (WebLogic Integration)
  - IBM, Microsoft, Oracle (with DB products)
  - ...
- Site: www.w3c.org/xquery

Xquery: caractéristiques

- Projections d'arbres sur des sous-arbres,
- Sélections d'arbres et de sous-arbres à partir de différents documents,
- Utilisation de variables dans les requêtes pour mémoriser un arbre ou pour itérer sur des collections d'arbres,
- Permet la jointure à partir de documents sources multiples,
- Ré-ordonnancement des arbres,
- Imbrication de requêtes,
- Construction/modification des données
- Calculs d'agrégats
- Utilisation possible de fonctions définies par l'utilisateur

Composants du langage XQUERY

- Expression de cheminement (XPath1)
- Littéraux (Entiers, Flottants, Doubles, Chaînes…),
- Variables (Notation *$nom*),
- Opérateurs
- Définition et appel de fonctions
- Expressions FLWR
- Expressions conditionnelles (XPath2)
- Conditions «Some» et «Every» (XPath2)
- Constructeur

Annexe: Main Principles

The design of XQuery satisfies the following rules:

- Closed-form evaluation. XQuery relies on a data model, and each query maps an instance of the model to another instance of the model.
- Composition. XQuery relies on expressions which can be composed to form arbitrarily rich queries.
- Type awareness. XQuery may associate an XSD schema to query interpretation. But XQuery also operates on schema-free documents.
- XPath compatibility. XQuery is an extension of XPath 2.0 (thus, any XPath expression is also an XQuery expression).
- Static analysis. Type inference, rewriting, optimization: the goal is to exploit the declarative nature of XQuery for clever evaluation.

Annexe: Concepts of XQuery

- Declarative/Functional
  - No execution order!
- Document Order
  - all nodes are in "textual order"
- Node Identity
  - all nodes can be uniquely identified
- Atomization
- Effective Boolean Value
- Type system

Plan

1. Introduction et historique
2. Modèle de donnée de Xquery: XDM
3. Syntaxe XQuery
4. Xquery avancé: Fonctions, jointure
5. Conclusion
6. Exercice

Le modèle des données pour Xquery

Basé comme XPATH2: XML Data Model

- Modèle abstrait "data model" pour les données XML (l'équivalent du modèle relationnel dans SGBDR)
- Pas de standard de stockage ou d'accès pour l'instant.
- Une instance de ce « data model » est une *sequence* composée de zéro ou plusieurs *items*. La séquence vide est souvent considérée comme "null value".

XML Data Model

- Les items sont:
  - Des nœuds ou des valeurs atomiques

- Les nœuds sont:

  document | element | attribute | text | namespaces | PI | comment

- Valeurs atomiques sont:
  - Des Instances de tous les types atomiques de XML Schema

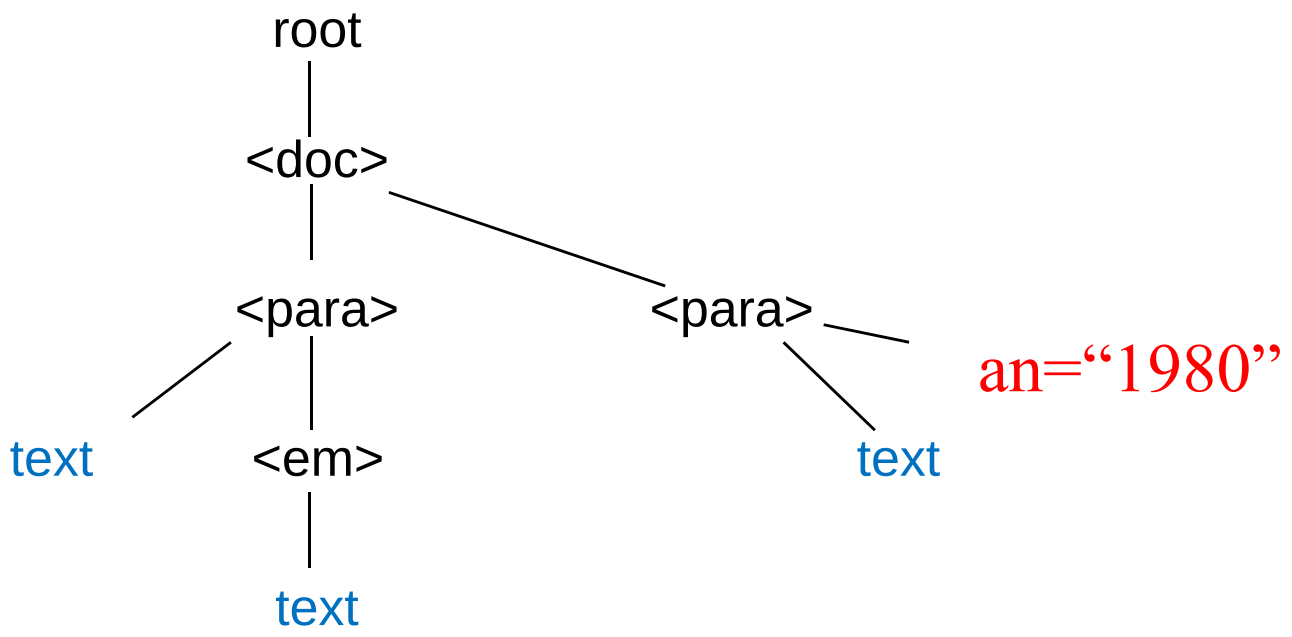
  string, boolean, ID, IDREF, decimal, QName, URI, ...
  - untyped atomic values

- Typed (I.e. schema validated) et untyped (I.e. pas de validation XSD) pour les nœuds comme pour les valeurs

Exemple du Data Model (1)

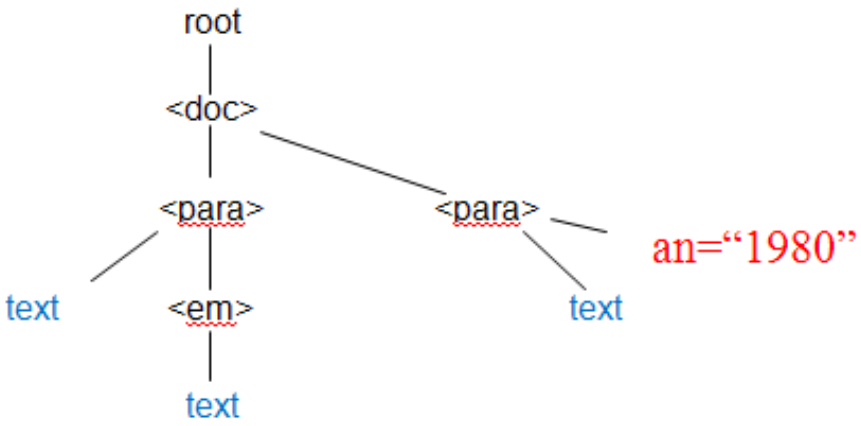
```
<doc>
  <para>blabla1 <em>blabla2</em></para>
  <para an="1980">blabla4.</para>
</doc>
```

A pour représentation:



Exemple du Data Model (2)

```
<doc>
  <para>blabla1 <em>blabla2</em></para>
  <para an="1980">blabla4.</para>
</doc>
```



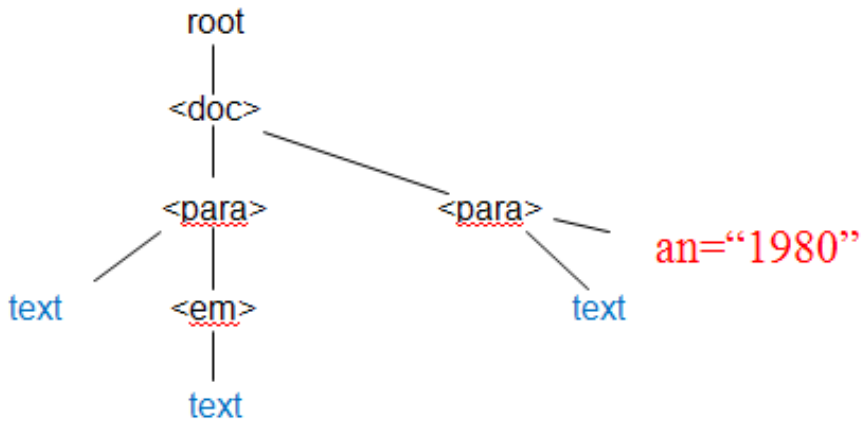
Untyped XML dans XDM:

- 4 noeuds “elements”, 1 noeud attribut, 3 noeuds texte.
  - Name(doc element) = {-}:doc
- Si pas de XSD
  - type(doc element) = xdt:untyped
  - type(para element) = xdt:untyped
  - type(an attribute) = xdt:untypedAtomic
  - …
  - typed-value(em element) = (“blabla2”, xdt:untypedAtomic)
  - typed-value(an attribute) = (“1980”, xdt:untypedAtomic)

Exemple du Data Model (3)

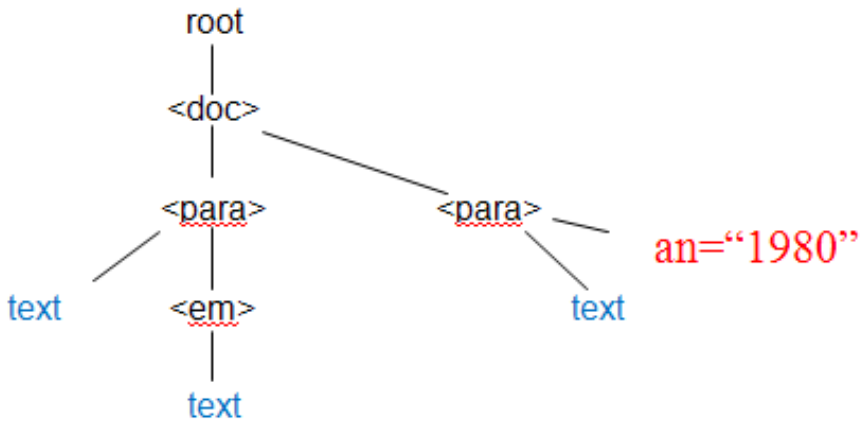
Soit la XSD associée au document:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:complexType name="para-type" mixed="true">
      <xs:sequence>
            <xs:element ref="em" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="an" type="xs:integer"/>
  </xs:complexType>
  <xs:element name="para" type="para-type"/>
  <xs:element name="em" type="xs:string"/>
  <xs:complexType name="doc-type">
      <xs:sequence>
            <xs:element ref="para" maxOccurs="unbounded"/>
      </xs:sequence>
  </xs:complexType>
  <xs:element name="doc" type="doc-type"/>
</xs:schema>
```



Exemple du Data Model (4)

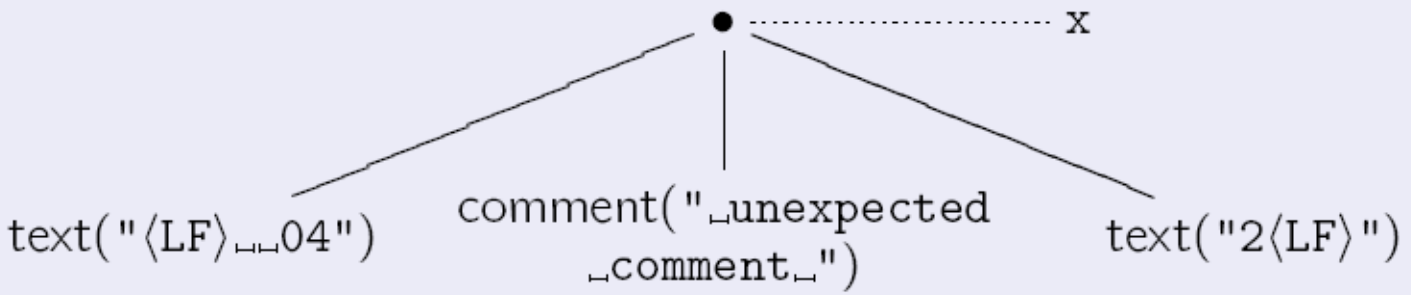
```
<doc>
<para>blabla1 <em>blabla2</em></para>
  <para an="1980">blabla4.</para>
</doc>
```



- Après validation avec XSD:
  - type(doc element) = {uri}:doc-type
  - type(para element) = {uri}:para-type
  - type(an attribute) = xs:integer
  - …
  - typed-value(em element) = (“blabla2”, xs:string)
  - typed-value(an attribute) = (“1980”, xs:integer)

* Data model: untyped vs. typed

Untyped view . . .




```
<x>
04<!-- unexpected comment -->2
</x>
```

Typed view . . .

- XQuery can work with the typed view, if the input XML document has been **validated** against an XML Schema description.

Source https://svn.uni-konstanz.de/dbis/xml_w09/material/XMLDB-08.pdf

Impact du Data Model

- (8, xs:integer) n'est pas la même chose que (8, myxs:MaTailleType) dans le Xquery

- Les implementations peuvent stockés:
  - ✓ La string value (obtenu via fn:string) et extrait dynamiquement le type.
  - ✓ Le typed value (obtenu via fn:data) et extrait la valeur lexicale suivant le type à chaque fois qu'il est nécessaire.
  - ✓ Les deux

Quand il n'y a pas de validation les deux types de valeurs sont les même.

* Data model: untyped vs. typed

```
<x>
04<!-- unexpected comment -->2
</x>
```

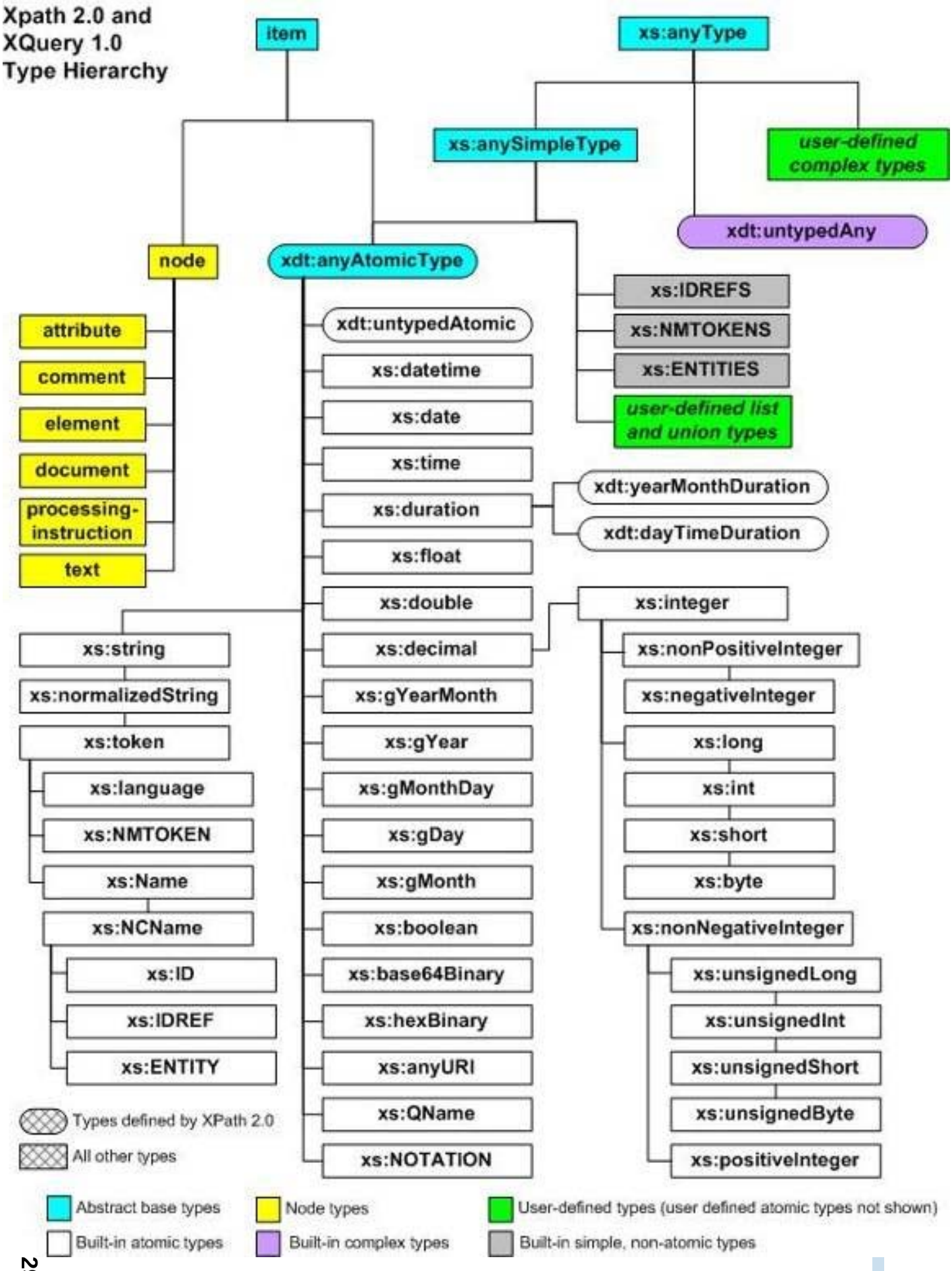
Node properties of unvalidated element x

node-name	x
parent	()
children	(t_1, c, t_2)
attributes	$\emptyset$
string-value	"⟨LF⟩␣␣042⟨LF⟩"
typed-value	"⟨LF⟩␣␣042⟨LF⟩"
type-name	untypedAtomic

Node properties of **validated** element x

node-name	x
parent	()
children	(t_1, c, t_2)
attributes	$\emptyset$
string-value	"042"
typed-value	42
type-name	integer

Source https://svn.uni-konstanz.de/dbis/xml_w09/material/XMLDB-08.pdf



www.cafeconleche.org/.../xquery/XQuery.html.

Les types supportés

- Les *atomic types* du XML Schema
  - String: "125.0" ou '125.0'
  - Integer: 125
  - Decimal: 125.0
  - Double: 125.e2

- On peut construire les valeurs avec les fonctions dédiées:
  - fn:true(), fn:date("2002-5-20")

Xquery: Input

- Fichier texte en XML,

Fonction doc("livresHouda.xml")

- Fragment XML extrait à partir du web via un URI

- Une collection de documents XML associés à un URI

Fonction collection("NosLivres")

- Données dans une base de données native/enabled XML

fn:doc("livresHouda.xml") *fn="http://www.w3.org/2005/04/xpath-functions"*

Xquery output: un item ou une séquence

- Une séquence est une collection d'items *(valeurs atomiques ou nœud)* qui:
  - peut être vide
  - est ordonnée :

    (1,5) ≠ (5,1)
- Une séquence de longueur 1 correspond à un item :

  12 ⇔ (12)
- Une séquence peut contenir des valeurs hétérogènes :

  (1, "titi", <grosminet/>)
- Pas de séquences imbriquées

Un item: détail

- Un item est *une valeur atomique ou un nœud.*
- Les *nœuds ont un id* les valeurs pas.
- Les éléments et attributs ont *un type annotations*, qui est soit inféré par une XSD ou inconnu si pas de schéma associé.
- Les *nœuds ont un ordre relatif à leur document*. Les attributs pas.

Les commentaires en XQuery

```
(: Ceci est un commentaire XQuery :)
```

Plan

Xquery : syntax

- Structure

  Un **prolog** + une **expression**

- Sensible à la casse. Les mots clés sont en général en minuscule.

- N'a pas une syntaxe XML ☹,

- Langage déclaratif (à la SQL)

Xquery: prolog

- Rôle:
  - Donner le contexte dans lequel se fera la compilation et l'évaluation de l'expression.
- Contient des:
  - Définitions des namespace
  - Imports de schema
  - Définitions de fonctions
  - Import des libraires de fonctions
  - …

```
xquery version "1.0 "
    (:permet d'indiquer au processeur la version :)

declare boundary-space strip
    (:elimine les blancs en trop sinon preserve:)

declare namespace b:= "www.eif.ch"
    (:les namespaces que l'on va utiliser plus loin:)

import schema at URI
    (:permet d'importer le XSD indiqué:)

declare variable $x := doc("livres.xml")
    (:declare une variable x contenant le document:)
```

Exemple prolog: (xmlspy altova /using_math.xq)

```
(: Copyright (c) 2004 Altova GmbH, http://www.altova.com :)

(: Test for imported functions and variables :)

(: Purpose: demonstrating library import, direct element constructors, computed
   content, for loops, sqrt  calculation etc.:)

xquery version "1.0" encoding "UTF-8";

declare boundary-space preserve;

declare default element namespace "http://www.altova.com";

import module namespace math="http://www.xmlspy.com/xquery/math" at
   "math.xq"  ;

<demo_using_math>

    ....

</demo_using_math>
```

Les namespaces implicitement implémentés

xml = "http://www.w3.org/XML/1998/namespace"

(:seul a ne pas pouvoir être redéfini:)

xs = "http://www.w3.org/2001/XMLSchema"

xsi = "http://www.w3.org/2001/XMLSchema-instance"

fn = "http://www.w3.org/2005/04/xpath-functions"

xdt = "http://www.w3.org/2005/04/xpath-datatypes"

L'expression de XQuery

XQuery Expr :=Constants | Variable | FunctionCalls |

PathExpr | ComparisonExpr | ArithmeticExpr |

LogicExpr | FLWRExpr | ConditionalExpr |

QuantifiedExpr | TypeSwitchExpr |

InstanceofExpr | CastExpr | UnionExpr |

IntersectExceptExpr | ConstructorExpr |

ValidateExpr

Permet l'imbrication d'expressions

Les constantes

XQuery Expr :=Constants | Variable | FunctionCalls |
PathExpr | ComparisonExpr | ArithmeticExpr |
LogicExpr | FLWRExpr | ConditionalExpr |
QuantifiedExpr |TypeSwitchExpr |
InstanceofExpr | CastExpr | UnionExpr |
IntersectExceptExpr | ConstructorExpr |
ValidateExpr

- Des valeurs directes:
  - Chaine entre double ou simple quote

"toto" ou ´toto´

  - Nombre 1

- Valeurs avec un type spécifique:

xs:date("2011-08-30")

Les variables

XQuery Expr :=Constants | Variable | FunctionCalls | PathExpr | ComparisonExpr | ArithmeticExpr | LogicExpr | FLWRExpr | ConditionalExpr | QuantifiedExpr |TypeSwitchExpr | InstanceofExpr | CastExpr | UnionExpr | IntersectExceptExpr | ConstructorExpr | ValidateExpr

- Identifiés par un nom précédé par $
- Peuvent être définis dans:
  - Le prolog
  - Fonctions
  - Expressions FLWOR

Appel de fonctions

XQuery Expr :=Constants | Variable | FunctionCalls |
PathExpr | ComparisonExpr | ArithmeticExpr |
LogicExpr | FLWRExpr | ConditionalExpr |
QuantifiedExpr |TypeSwitchExpr |
InstanceofExpr | CastExpr | UnionExpr |
IntersectExceptExpr | ConstructorExpr |
ValidateExpr

Nomfonction(list_arguments)

Un argument est n'importe quelle expression simple:

- Variable, Xpath, …

Exemple de fonctions prédéfinies

- Exemple de fonctions qui travaillent sur les séquences:

Function	**Example**
count	count((0,4,2)) → 3
max	max((0,4,2)) → 4
subsequence	subsequence((1,3,5,7),2,3) → (3,5,7)
empty	empty((0,4,2)) → false()
exists	exists((0,4,2)) → true()
distinct-values	distinct-values((4,4,2,4)) → (4,2)
to	(1 to 10)[. mod 2 eq 1] → (1,3,5,7,9)

L'expression de XQuery

XQuery Expr :=Constants | Variable | FunctionCalls |
PathExpr | ComparisonExpr | ArithmeticExpr |
LogicExpr | FLWRExpr | ConditionalExpr |
QuantifiedExpr | TypeSwitchExpr |
InstanceofExpr | CastExpr | UnionExpr |
IntersectExceptExpr | ConstructorExpr |
ValidateExpr

Expressions de chemin

XQuery Expr :=Constants | Variable | FunctionCalls | **PathExpr** | ComparisonExpr | ArithmeticExpr | LogicExpr | FLWRExpr | ConditionalExpr | QuantifiedExpr |TypeSwitchExpr | InstanceofExpr | CastExpr | UnionExpr | IntersectExceptExpr | ConstructorExpr | ValidateExpr

Basée sur la syntaxe XPath 2

- *Titres des BD dans le document livres.xml:*

```
<livre>
   <BD titre='Asterix le gaulois'>
     <scenariste>Gossini</scenariste>
     <dessin>Uderzo</dessin>
     <pub>Dargaux</pub>
     <prix>18.95</prix>
   </BD>
   <BD  annee='2001' titre='Titeuf'>
     <scenariste>Zep</scenariste>
     <dessin>Zep</dessin>
     <pub>Dargaux</pub>
    <prix>10.00</prix>
   </BD>
   <BD annee='1980' titre='Lucky luck'>
    <scenariste>Gossini</scenariste><scenariste>Moris</scenariste>
     <dessin>Moris</dessin>
     <pub>Dargaux</pub>
    <prix>10.00</prix>
   </BD>
</livre>
```

Expressions de chemin

Basée sur syntaxe XPath 2

- *Titres des BD dans le document livres.xml:*

```
document("livres.xml")//BD/@titre
```

L'expression de XQuery

XQuery Expr :=Constants | Variable | FunctionCalls |
PathExpr | ComparisonExpr | ArithmeticExpr |
LogicExpr | FLWRExpr | ConditionalExpr |
QuantifiedExpr | TypeSwitchExpr |
InstanceofExpr | CastExpr | UnionExpr |
IntersectExceptExpr | ConstructorExpr |
ValidateExpr

Arithmetic Expressions

XQuery Expr :=Constants | Variable | FunctionCalls | PathExpr | ComparisonExpr | **ArithmeticExpr** | LogicExpr | FLWRExpr | ConditionalExpr | QuantifiedExpr |TypeSwitchExpr | InstanceofExpr | CastExpr | UnionExpr | IntersectExceptExpr | ConstructorExpr | ValidateExpr

```
1 + 4

5 div 6

1 - (4 * 8.5)

<a>42</a> + 1

() * 42

(1,2) - (2,3)

<a>baz </a> + 1

$a div 5

$b mod 10

-55.5
```

Arithmetic Expression - Evaluation

- Apply the following rules:
  - *atomize* all operands.
  - if either operand is (), => ()
  - if an operand is untyped, cast to *xs:double* (if unable, => *error*)
  - if the operand types differ but can be *promoted* to common type, do so (e.g.: *xs:integer* can be promoted to *xs:double*)
  - if operator is consistent w/ types, apply it; result is either atomic value or *error*
  - if type is not consistent, throw type exception

Arithmetic Expressions

XQuery Expr :=Constants | Variable | FunctionCalls | PathExpr | ComparisonExpr | **ArithmeticExpr** | LogicExpr | FLWRExpr | ConditionalExpr | QuantifiedExpr |TypeSwitchExpr | InstanceofExpr | CastExpr | UnionExpr | IntersectExceptExpr | ConstructorExpr | ValidateExpr

5	1 + 4
0.833…	5 div 6
-33	1 - (4 * 8.5)
43	<a>42 + 1
()	() * 42
Type error	(1,2) - (2,3)
Type error	<a>baz + 1
	$a div 5
	$b mod 10
	-55.5

L'expression de XQuery

XQuery Expr :=Constants | Variable | FunctionCalls |
PathExpr | ComparisonExpr | ArithmeticExpr |
LogicExpr | FLWRExpr | ConditionalExpr |
QuantifiedExpr | TypeSwitchExpr |
InstanceofExpr | CastExpr | UnionExpr |
IntersectExceptExpr | ConstructorExpr |
ValidateExpr

Les opérateurs de comparaisons

Catégorie	Signification	Symbole
Valeur	Compare des valeurs atomiques	eq, ne, lt, le, gt, ge
Général	quantification + comparaison de valeurs	=, !=, <=, <, >, >=
Nœud	Pour tester l'identité d'un nœud	is, isnot
Ordre	Pour tester les positions relatives d'un nœud ./. à un autre (ordre du document)	<<, >>

Comparaison de valeurs (atomiques)

Si un des opérandes est:

eq, ne, lt, le, gt, ge

- Un nœud → est transformé en une valeur atomique (sinon erreur) via fn:data(node) qui extrait la valeur typée du nœud
- Un Untyped → est traité comme une chaine.

```
xs:decimal(//livre/BD[3]/prix) gt 100.0
```

On fait le cast si pas de XSD pour définir le type de prix

```
//livre/BD[3]/scenariste eq "moritz"
```

Erreur: retour de plusieurs noeuds

Comparaison générale (séquences)

=, !=, <=, <, >, >=

Compare des *séquences de valeurs atomiques* entre elles:

- True **si** au moins **une** valeur de l'opérande gauche = à **une** valeur de l'opérande droit!

//livre/BD[3]/scenariste = "moritz"

True: si l'un des scénaristes est moritz

Comparaisons de valeurs vs général

(atomique vs sequence)

	Résultats avec =	
<a>42 eq "42"	true	true
<a>42 eq 42	error	true
<a>42 eq 42.0	error	true
<a>42 eq "42.0"	false	false
<a>42 eq 42	true	true
<a>42 eq 42	false	false
<a>toto eq 42	error	error
ns:gYear("2000") eq ns:integer(2000)	error	error
ns:float("2000") eq ns:integer(2000)	true	true

Atomique → Un Untyped est traité comme une chaine

Sequences → cast to double les untyped

Comparaisons de valeurs vs général

(atomique vs sequence)

Différences

```
() eq 42                                   ()
() = 42                                    false
```

Pas possible avec **eq**

```
(<a>42</a>, <b>43</b>) = 42.0    true
(<a>42</a>, <b>43</b>) = "42"    true
(1,2) = (2,3)                    true
```

Exemples comparaison atomique

eq, ne, lt, le, gt, ge

- Requête

```
document("livres.xml")//BD[dessin eq "Zep"]
```

- Résultat

```
<BD  annee='2001' titre='Titeuf'>
    <scenariste>Zep</scenariste>
    <dessin>Zep</dessin>
    <pub>Dargaux</pub>
    <prix>10.00</prix>
</BD>
```

```
<livre>
  <BD titre='Asterix le gaulois'>
    <scenariste>Gossini</scenariste>
    <dessin>Uderzo</dessin>
    <pub>Dargaux</pub>
    <prix>18.95</prix>
  </BD>
  <BD  annee='2001' titre='Titeuf'>
    <scenariste>Zep</scenariste>
    <dessin>Zep</dessin>
    <pub>Dargaux</pub>
    <prix>10.00</prix>
  </BD>
  <BD annee='1980' titre='Lucky luck'>
    <scenariste>Gossini</scenariste>
    <scenariste>Moris</scenariste>
    <dessin>Moris</dessin>
    <pub>Dargaux</pub>
    <prix>10.00</prix>
  </BD>
</livre>
```

- Requête

  document("livres.xml")//BD[dessin eq "Zep"]

- Fonctionnement:

  comparaison n'est possible que si les deux protagonistes sont de type atomique.

- Mauvaise requête:

  document("livres.xml")//BD[scenariste eq "Zep"]

  Renverra une erreur car **scenariste** renvoie une *séquence de valeurs*.

Exemple comparaison générale

=, !=, <=, <, >, >=

- Requête

```
Count(document("livres.xml")//BD[scenariste =
   ("Zep" , "Moris")] )
```

- Fonctionnement:

  Compte le nombre d'éléments **BD** dont les fils **scenariste** contiennent au moins un des 2 noms *Zep* ou *Moris*.

- Resultat: **2**

Exemple comparaison de nœuds

is, isnot

- Requête

document("livres.xml")//BD[scenariste[2] is scenariste[last()]]

- Fonctionnement: Extraction des nœuds **BD** qui ont exactement 2 fils **scenariste**

- Résultat:

```
<BD annee='1980' titre='Lucky luck'>
    <scenariste>Gossini</scenariste>
    <scenariste>Moris</scenariste>
    <dessin>Moris</dessin>
    <pub>Dargaux</pub>
   <prix>10.00</prix>
</BD>
```

Exemple comparaison par la position

<<, >>

- **n1 << n2** signifie que **n1** apparaît avant **n2** dans le document.
- **n1 >> n2** signifie que **n1** apparaît après **n2** dans le document.

- Requête

```
document("livres2.xml")//BD[scenariste[./text() ="Gossini"] <<
   dessin[./text() ="Moris"]]/@titre
```

- Fonctionnement:Extraction de l'attribut **titre** des éléments **BD où** la valeur "Gossini" apparaît avant la valeur *"Moris".*

- Résultat: titre = "Lucky Luck"

L'expression de XQuery

XQuery Expr :=Constants | Variable | FunctionCalls |
PathExpr | ComparisonExpr | ArithmeticExpr |
LogicExpr | FLWRExpr | ConditionalExpr |
QuantifiedExpr | TypeSwitchExpr |
InstanceofExpr | CastExpr | UnionExpr |
IntersectExceptExpr | ConstructorExpr |
ValidateExpr

Expressions logiques

expr1 **and** expr2

expr1 **or** expr2 **fn:not()** comme fonction

retourne **true, false**: Logique binaire (≠ SQL)

Comportement:

1. Calcule pour chaque opérande sa *Boolean Effective Value (BEV)*:
   - ✓ Si (), "", NaN (Not A Number), 0, alors **false**
   - ✓ Si l'opérande est de type **xs:boolean**, le **retourner**;
   - ✓ Si l'opérande est une séquence avec 1 premier item nœud alors **true**
   - ✓ Sinon lever une **erreur**
2. Puis on utilise la logique booléenne sur les BEV obtenues

false and error → non-déterministe: *soit* false *soit* error
true or error → non-déterministe: *soit* true *soit* error

Exemple: Expressions logiques sur BaseX

Calcule pour chaque opérande sa *Boolean Effective Value (BEV)*:

- ✓ Si (), "", NaN (Not A Number), 0, alors **false**

( ) and 2 eq 2 → false
( ) or 2 eq 2 → true

0 or 2 eq 2 → true
0 and 2 eq 2 → false
10 or 2 eq 2 → true
10 and 2 eq 2 → true

- ✓ Si l'opérande est de type **xs:boolean**, le **retourner**;
- ✓ Si l'opérande est une séquence avec 1 premier item nœud alors **true**

(<e/>,1) and 2 eq 2 → true
(1,<e/>,1) and 2 eq 2 → error

- ✓ Sinon lever une **erreur**

L'expression de XQuery

XQuery Expr :=Constants | Variable | FunctionCalls |
PathExpr | ComparisonExpr | ArithmeticExpr |
LogicExpr | FLWRExpr | ConditionalExpr |
QuantifiedExpr | TypeSwitchExpr |
InstanceofExpr | CastExpr | UnionExpr |
IntersectExceptExpr | ConstructorExpr |
ValidateExpr

Constructeurs d'éléments

- Permettent de construire des structures XML:
  - constructeur d'élément, de commentaires, PI…
- Deux familles:
  - constructeurs **directs**: syntaxe XML

* Mon texte préféré *

  - constructeurs **calculés**

Constructeurs calculés

- ✓element {expr-nom} {expr-contenu}
- ✓attribute {expr-nom} {expr-contenu}

Avec:

- ***expr-nom*** : chemin pour calculer le nom de l'élément ou de l'attribut
- ***expr-contenu :*** contenu du nouvel élément ou attribut (à évaluer)

Constructeurs: exemple 1

- Requête:

```
<livre>
  { document("livres.xml")//BD[2]/@annee }
  { document("livres.xml")//BD[2]/pub }
 </livre>
```

- Résultat

```
<livre annee="2001">
   <pub>Dargaux</pub>
</livre>
```

```
<livre>
   <BD titre='Asterix le gaulois'>
      <scenariste>Gossini</scenariste>
      <dessin>Uderzo</dessin>
      <pub>Dargaux</pub>
      <prix>18.95</prix>
   </BD>
   <BD  annee='2001' titre='Titeuf'>
      <scenariste>Zep</scenariste>
      <dessin>Zep</dessin>
      <pub>Dargaux</pub>
     <prix>10.00</prix>
   </BD>
  <BD annee='1980' titre='Lucky luck'>
      <scenariste>Gossini</scenariste>
      <scenariste>Moris</scenariste>
      <dessin>Moris</dessin>
      <pub>Dargaux</pub>
     <prix>10.00</prix>
   </BD>
</livre>
```

- Requête:

```
<livre>
 { document("livres.xml")//BD[2]/@annee }
 { document("livres.xml")//BD[2]/pub }
</livre>
```

- Fonctionnement
  - Construction d'un élément **livre** pour encadrer la partie calculée de la requête.
  - Partie calculée (entre accolades) renvoie l'attribut **année** pour livre et les éléments **pub** du deuxième élément BD parcouru dans le document **livres.xml**

Constructeurs: exemple 2

- Requête

```
element
{document('livres.xml')//BD[1]/name(@*[1]) }
{attribute
    {document('livres.xml')//BD[1]/name(*[3]) }
    { document('livres.xml')//BD[1]/*[3] }
}
```

- Résultat

```
<titre pub='Dargaux'/>
```

```
<livre>
  <BD titre='Asterix le gaulois'>
  <scenariste>Gossini</scenariste>
    <dessin>Uderzo</dessin>
    <pub>Dargaux</pub>
    <prix>18.95</prix>
  </BD>
  <BD  annee='2001' titre='Titeuf'>
    <scenariste>Zep</scenariste>
    <dessin>Zep</dessin>
    <pub>Dargaux</pub>
   <prix>10.00</prix>
  </BD>
 <BD annee='1980' titre='Lucky luck'>
    <scenariste>Gossini</scenariste>
    <scenariste>Moris</scenariste>
    <dessin>Moris</dessin>
    <pub>Dargaux</pub>
   <prix>10.00</prix>
  </BD>
</livre>
```

- Requête

```
element {document('livres.xml')//BD[1]/name(@*[1]) }
  {attribute
     {document('livres.xml')//BD[1]/name(*[3]) }
     { document('livres.xml')//BD[1]/*[3] }
  }
```

- Fonctionnement
  - Création d'un élément **vide** dont le nom est le premier attribut**: titre** du premier BD
  - Cet élément contient un attribut dont le nom est le troisième élément fils du premier BD qui est **pub** et la valeur est la valeur de ce troisième élément.

Constructeurs: exemple 3

- Requête

  element BandeDessinée
  {attribute publication **{** document('livres.xml')//BD[1]/*[3] **}**
  }

- Résultat

  <BandeDessinée publication="Dargaux"/>

- Equivalent à

<BandeDessinée>
{attribute publication **{** document('livres.xml')//BD[1]/*[3] **} }**
</BandeDessinée>

```
<livre>
   <BD titre='Asterix le gaulois'>
   <scenariste>Gossini</scenariste>
    <dessin>Uderzo</dessin>
    <pub>Dargaux</pub>
    <prix>18.95</prix>
  </BD>
  ...
</livre>
```

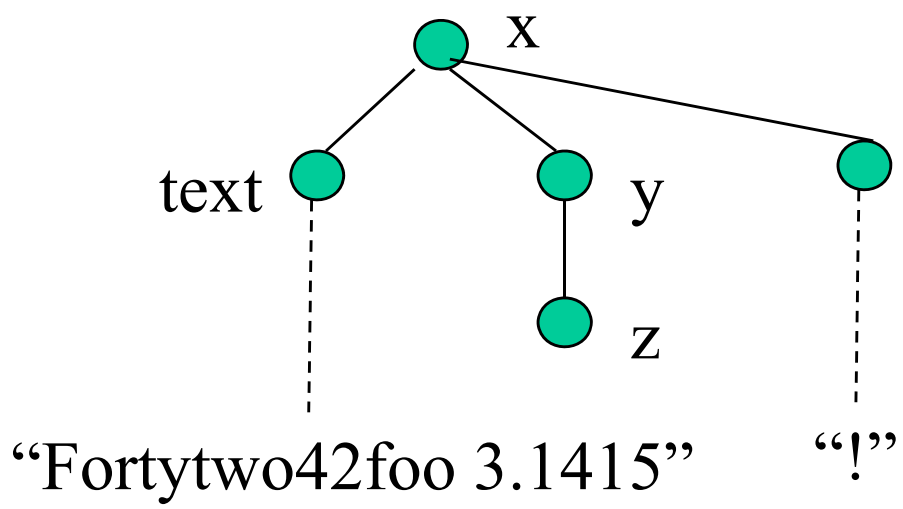
- Requête

```
element BandeDessinée
  {attribute
     publication
     { document('livres.xml')//BD[1]/*[3] }
  }
```

- Fonctionnement
  - Création d'un élément dont le nom est **BandeDessinée.**
  - Cet élément ne contient qu'un attribut dont le nom est **publication** et la valeur est la valeur du troisième élément.

Que donne ceci?

```
<x>Fortytwo{40 + 2}{ "foo", 3.1415, <y><z/></y>, ("", "!")[2] }</x>
```




```
<x>Fortytwo42foo 3.1415<y> <z/></y>!</x>
```

L'expression de XQuery

XQuery Expr :=Constants | Variable | FunctionCalls |
PathExpr | ComparisonExpr | ArithmeticExpr |
LogicExpr | FLWRExpr | ConditionalExpr |
QuantifiedExpr | TypeSwitchExpr |
InstanceofExpr | CastExpr | UnionExpr |
IntersectExceptExpr | ConstructorExpr |
ValidateExpr

Expressions FLW(O)R: syntax

- Exemple

```
for $b in document("livres.xml")//BD    /* similaire au FROM dans SQL */
let $a := $b/scenariste                 /* ☹  SQL */
where $b[@annee > 2000]                 /* similaire au WHERE dans SQL */
return                                  /* similaire au SELECT dans SQL */
   <BD nb_scen="{count($a)}"> { $a } </BD>
```

Expressions FLWR: explication

```
FOR $<var>  in <forest>,
      $<var> ...                    // itération
LET $<var> := <subtree>             // assignation-affectation
WHERE <condition>                   // élagage - filtre
RETURN <result>                     // construction
```

Affectation de variables: for et Let

- **for $var in expr**
  - Affecte successivement **$var** avec chaque item de la séquence retournée par **expr**.

- **let $var := expr**
  - Affecte **$var** avec la séquence entière retournée par **expr**.

Exemple 1: for

```
for $i in (1 to 3) return $i +1
```

Résultat

(2, 3, 4)

Fonctionnement

Variable i prendra ses valeurs successives dans la séquence (1, 2, 3)

Exemple 2: for

```
for $i in (1, 2), $j in (11, 12)
return <res> {$i} plus {$j} = {$i + $j} </res>
```

Résultat

<res>1 plus 11 = 12</res>,
<res>1 plus 12 = 13</res>,
<res>2 plus 11 = 13</res>,
<res>2 plus 12 = 14</res>

Produit cartésien

Fonctionnement

Première boucle: variable i prend ses valeurs successives dans la séquence (1, 2). Boucle imbriquée: variable j prend ses valeurs successives dans (11,12)

Exemple 3: for avec at

- at permet de garder la position de l'élément:

```
for $x at $i in document("livres.xml")//BD
where $i mod 2 = 0
return $x
```

Résultat

Retourne les éléments BD de position paire.

```
<BD  annee='2001' titre='Titeuf'>
    <scenariste>Zep</scenariste>
    <dessin>Zep</dessin>
    <pub>Dargaux</pub>
   <prix>10.00</prix>
 </BD>
```

```
<livre>
   <BD titre='Asterix le gaulois'>
     <scenariste>Gossini</scenariste>
     <dessin>Uderzo</dessin>
     <pub>Dargaux</pub>
     <prix>18.95</prix>
   </BD>
   <BD  annee='2001' titre='Titeuf'>
     <scenariste>Zep</scenariste>
     <dessin>Zep</dessin>
     <pub>Dargaux</pub>
    <prix>10.00</prix>
   </BD>
 <BD annee='1980' titre='Lucky luck'>
     <scenariste>Gossini</scenariste>
     <scenariste>Moris</scenariste>
     <dessin>Moris</dessin>
     <pub>Dargaux</pub>
    <prix>10.00</prix>
   </BD>
</livre>
```

Exemple 3: for avec at

- at permet de garder la position de l'élément:

```
for $x at $i in document("livres.xml")//BD
where $i mod 2 = 0
return $x
```

$x	$i	where	return
BD1	1	false	
BD2	2	true	BD2
BD3	3	false	

Résultat

Retourne les éléments BD de position paire.

Fonctionnement

$i prend les valeurs entre 1.. nombre d'items dans la séquence.

Exemple 1: let

```
for $i in (1, 2)
let $j := (1, $i)
return $j
```

Résultat

(1,1,1,2)

Fonctionnement

Boucle for: variable $i prend ses valeurs successives dans la séquence (1, 2). Pour chaque affectation de $i le let affecte à $j la séquence (1, $i)

Autres exemples

```
for $b in document("livres.xml")//BD[3]
    let $a := $b/scenariste
    return <BD nb_scen="{count($a)}">
                { $a }
           </BD>
```

Résultat

```
<BD nb_scen="2">
     <scenariste>Gossini</scenariste>
     <scenariste>Moris</scenariste>
 </BD>
```

```
<livre>
   <BD titre='Asterix le gaulois'>
     <scenariste>Gossini</scenariste>
     <dessin>Uderzo</dessin>
     <pub>Dargaux</pub>
     <prix>18.95</prix>
   </BD>
   <BD  annee='2001' titre='Titeuf'>
     <scenariste>Zep</scenariste>
     <dessin>Zep</dessin>
     <pub>Dargaux</pub>
     <prix>10.00</prix>
   </BD>
  <BD annee='1980' titre='Lucky luck'>
     <scenariste>Gossini</scenariste>
     <scenariste>Moris</scenariste>
     <dessin>Moris</dessin>
     <pub>Dargaux</pub>
     <prix>10.00</prix>
   </BD>
</livre>
```

```
for $b in document("livres.xml")//BD
let $a := $b/scenariste
return <BD nb_scen="{count($a)}"> { $a } </BD>
```

Fonctionnement

- la variable **$b** sera affectée successivement avec le contenu des 3ieme nœuds **BD** du document livres.xml (clause **for**) (ici une fois!)
- d'affecter **$a** avec l'ensemble des nœuds **scénariste** contenus dans chaque itération de **$b** (clause **let**).

Sélection: where (1)

- **where expr:** Filtre le résultat

- Exemple

```
<BD>
{   for $a in document("livres.xml")//BD
    where $a/dessin eq 'Zep'
    return  $a/@titre   }
</BD>
```

- Résultat

```
<BD titre="Titeuf"/>
```

```
<livre>
  <BD titre='Asterix le gaulois'>
    <scenariste>Gossini</scenariste>
    <dessin>Uderzo</dessin>
    <pub>Dargaux</pub>
    <prix>18.95</prix>
  </BD>
  <BD  annee='2001' titre='Titeuf'>
    <scenariste>Zep</scenariste>
    <dessin>Zep</dessin>
    <pub>Dargaux</pub>
    <prix>10.00</prix>
  </BD>
 <BD annee='1980' titre='Lucky luck'>
    <scenariste>Gossini</scenariste>
    <scenariste>Moris</scenariste>
    <dessin>Moris</dessin>
    <pub>Dargaux</pub>
    <prix>10.00</prix>
  </BD>
</livre>
```

- Exemple

```
<BD>
{   for $a in document("livres.xml")//BD
    where $a/dessin eq 'Zep'
    return  $a/@titre   }
</BD>
```

- Fonctionnement La variable **$a** sera affectée successivement avec le contenu des nœuds **BD** du document livres.xml (clause **for**). Si l'un des nœuds fils **dessin** contient comme valeur **Zep,** on extrait alors le titre.

Annexe: Where - xquery vs SQL

- Xquery where is quite similar to its SQL synonym. The difference lies in the much more flexible structure of XML documents.

- Note: predicates are interpreted according to the XPath rules:
  - if a path does not exists, the result is **false**, no typing error!
  - if a path expression returns several nodes: the **result is true if there is at least one match**.

Des requêtes équivalentes

```
for $b in doc('books.xml')//book
where $b/author[firstname = 'John' and lastname = 'Smith']
return <book>
          { $b/title,
            $b/price }
       </book>
```

- Ou

```
for $b in doc('books.xml')//book
                [author[firstname = 'John' and lastname = 'Smith']]
return <book>
          { $b/title,
            $b/price }
       </book>
```

Mais que donne cette requête?

```
for $b in doc('books.xml')//book [author/firstname = 'John'
                    and author/lastname = 'Smith']
return <book>
            { $b/title,
              $b/price }
          </book>
```

Est-ce vraiment notre sémantique (ce que nous cherchions)?

- Qu'advient-il de cet élément?

```
<book>
  <author><firtstname>Mary</firstname>
            <lastname>Smith</lastname>
  </author>
  <author><firtstname>John</firstname>
            <lastname>Travolta</lastname>
  </author>
</book>
```

Retenu

et que donne cette requête?

```
for $b in doc('books.xml')//book,
    $a in $b/author
where [$a/firstname = 'John' and $a/lastname = 'Smith']
return <book>
          { $b/title,
            $b/price }
       </book>
```

Est-ce vraiment notre sémantique (ce que nous cherchions)?

- Qu'advient-il de cet élément?

```
<book>
  <author><firtstname>Mary</firtstname>
          <lastname>Smith</lastname>
  </author>
  <author><firtstname>John</firtstname>
          <lastname>Travolta</lastname>
  </author>
</book>
```

Rejeté

Expressions FLWOR

- Forme générale

```
FOR $<var> in <forest>, $<var> ...
LET $<var> := <subtree>
WHERE <condition>
ORDER BY  ...                    // trie
RETURN <result>
```

Order by

- Dans un bloc FLWOR

```
for $v in e1 return e2
```

l'ordre de *e1* détermine l'ordre de la séquence résultante *e2*.

- On peut retrier via `order by`:

```
for $v in e1
order by e3 [ascending | descending]
           [empty greatest | least]
return e2
```

la valeur (atomization) de *e3* détermine l'ordre de la séquence résultante *e2*.

Exemples 1

- Résultat à la ORDER BY de SQL

```
for $x in (5,3,1)
order by $x
return $x
```

→ (1,3,5)

$x	order by
5	5
3	3
1	1

- Utilisation de l'information de position

```
for $x at $p in (5,3,1)
order by $p - $x
return $x
```

→ (5,3,1)

$x	$p	order by
5	1	-4
3	2	-1
1	3	2

Exemple 2

```
for $x at $p in reverse(1 to 10)
  let $y := $x * $x
  where $y <= 42
  order by 5 - $p
  return ($p,$x)
```

→ (10,1,9,2,8,3,7,4,6,5,5,6)

5 - $p

$x	$p	$y	where	order by	return
10	1	100	false	4	(1,10)
9	2	81	false	3	(2,9)
8	3	64	false	2	(3,8)
7	4	49	false	1	(4,7)
6	5	36	true	0	(5,6)
5	6	25	true	-1	(6,5)
4	7	16	true	-2	(7,4)
3	8	9	true	-3	(8,3)
2	9	4	true	-4	(9,2)
1	10	1	true	-5	(10,1)

Order by

- Exemple

```
for $a in doc("livres1.xml")//BD
        order by $a/@titre,$a/pub descending
        return  $a
```

- Fonctionnement agit avant le return ici va ordonner les BD dans l'ordre d'abord des titres puis des pub ces derniers pris dans l'ordre descendant.

Attention: si on trie sur un critère qui peut valoir () pour certains items, ceux si sont considérés comme les plus petits ou les plus grand suivant les implémentations. Il y a moyen de fixer le comportement ☺

- Exemple

```
for $a in doc("livres1.xml")//BD
        stable order by $a/@titre empty greatest,
                                $a/pub descending empty least
        return  $a
```

L'expression de XQuery

XQuery Expr :=Constants | Variable | FunctionCalls |
PathExpr | ComparisonExpr | ArithmeticExpr |
LogicExpr | FLWRExpr | ConditionalExpr |
QuantifiedExpr | TypeSwitchExpr |
InstanceofExpr | CastExpr | UnionExpr |
IntersectExceptExpr | ConstructorExpr |
ValidateExpr

Expressions conditionnelles

if (expr_bool) then … else…

- Parenthèse obligatoire
- Le « else » est non facultatif, mais on peut le laisser sans instruction:

```
else()
```

Expression booléenne:

- vaut faux pour:
  xs:boolean false / nombre 0 ou NaN / une chaine vide / une séquence vide
- Génère une erreur pour une séquence d'items atomiques
- À vrai sinon!

Exemples

```
if (2)     then "true" else "false"     true
if (2,2)   then "true" else "false"     erreur
if (0)     then "true" else "false"     false
if (())    then "true" else "false"     false
if ("")    then "true" else "false"     false
if (" ")   then "true" else "false"     true
```

- Exemple

```
<BDs>
  {  for $b in document("livres.xml")//BD
     return
      if ($b/@annee > 2000)
      then <BD>{$b/@titre} récent</BD>
       else <BD>{$b/@titre} </BD>
   }
</BDs>
```

- Résultats

```
<BDs>
    <BD titre="Asterix le gaulois"/>
    <BD titre='Titeuf'> récent </BD>
    <BD titre='Lucky luck'/>
</BDs>
```

```
<livre>
  <BD titre='Asterix le gaulois'>
    <scenariste>Gossini</scenariste>
    <dessin>Uderzo</dessin>
    <pub>Dargaux</pub>
    <prix>18.95</prix>
  </BD>
  <BD annee='2001' titre='Titeuf'>
    <scenariste>Zep</scenariste>
    <dessin>Zep</dessin>
    <pub>Dargaux</pub>
   <prix>10.00</prix>
  </BD>
 <BD annee='1980' titre='Lucky luck'>
    <scenariste>Gossini</scenariste>
    <scenariste>Moris</scenariste>
    <dessin>Moris</dessin>
    <pub>Dargaux</pub>
   <prix>10.00</prix>
  </BD>
</livre>
```


104

Exemple

```
<BDs>
  {  for $b in document("livres.xml")//BD
     return
      if ($b/@annee > 2000)
      then <BD>{$b/@titre} récent</BD>
      else <BD>{$b/@titre} </BD>
   }
</BDs>
```

- Fonctionnement
  - Requête qui va tester pour chaque élément **BD** de **livres.xml** si la valeur de son attribut **année** est supérieure à 2000 alors renvoyer en plus la valeur ***'récent***' et ***rien en plus*** sinon.

105

Sémantique des expressions FLWR

- Expression FLWR :

```
for $a in document("livres.xml")//BD
let $b := $a/scenariste
where $a/@title="Marsipulami"
return count($b)
```

- Equivalente à:

```
for $a in document("livres.xml")//BD
return  (let $b := $a/scenariste
            return
                if ($a/@title ="Marsipulami")
                then count($b)
                else  ()
        )
```

L'expression de XQuery

XQuery Expr :=Constants | Variable | FunctionCalls |
PathExpr | ComparisonExpr | ArithmeticExpr |
LogicExpr | FLWRExpr | ConditionalExpr |
QuantifiedExpr | TypeSwitchExpr |
InstanceofExpr | CastExpr | UnionExpr |
IntersectExceptExpr | ConstructorExpr |
ValidateExpr

Quantification

- some $var in expr1 satisfies expr2 :
  - il existe au moins un noeud retourné par l'expression expr1 qui satisfait l'expression *expr2*.

Il existe ∃

- every $var in expr satisfies expr:
  - *tous* les nœuds retournés par l'expression *expr1* satisfont l'expression *expr2*

Quelque soit ∀

- Requête

```
for $a in distinct-values(document("livres2.xml")//pub)
where every $b in document("livres2.xml")//BD[pub = $a]
satisfies $b/dessin=("Moris")
return <elt>{string($a)}</elt>
```

- Fonctionnement

Extraction des seuls éléments **pub** dont toutes les **BD** relative à **pub** ont un fils **dessin** qui a pour valeur *"Moris".*

- Résultat

()

Faut changer le fichier pour voir quelque chose!

- Requête

```
for $a in distinct-values(document("livres2.xml")//pub)
where every $b in document("livres2.xml")//BD[pub = $a]
satisfies $b/dessin=("Moris")
return <elt>{string($a)}</elt>
```

- Requête équivalente

```
for $a in distinct-values(document("livres2.xml")//pub)
let $b := document("livres2.xml")//BD[pub = $a]
where every $c in $b satisfies $c/dessin=("Moris")
return <elt>{string($a)}</elt>
```

Plan

1. Introduction et historique
2. Modèle de donnée Xquery: XDM
3. Syntaxe XQuery
4. Xquery avancé: Fonctions, quelques opérateurs, jointure
5. Conclusion
6. Exercice

Fonctions

- Fonctions et opérateurs prédéfinis par XQuery 1.0 /XPath 2.0:
  - Distinct-values(…)
- Fonctions racines (permettant l'accès au document xml) :
  - input,
  - collection(\url00),
  - doc(\url00), …
- Fonctions utilisateurs
  - Définition de fonctions personnelles

Fonctions utilisateurs: syntaxe

```
declare function prefix:fct_name($parameter as datatype)
   as returnDatatype
   {
    ...function code here...
   };
```

- Le prefix devant le nom d'une fonction est obligatoire
- Le $ devant les noms de paramètre est obligatoire
- Les datatypes sont ceux du XMLSchema
- Si pas de datatype indiqué: équivalent à "as item()*«
- On peut ajouter le nombr d'ocurence devant les paramètres et le retour (none (blank),*,?,+) (cf DTD)

Fonctions utilisateurs: exemple

Exemple: Fonction qui permet de renvoyer le nombre de fils **scenariste** d'un élément. Le résultat est typé **xsd:integer**.

```
define function NombreScenariste(BD $b) returns
   xs:integer
{
   count($b/scenariste)
}
```

Requête avec utilisation de la fonction

```
for $a in document("livres.xml")//BD
where $a/dessin eq "Zep"
return  NombreScenariste($a)
```

Résultat

1

Exemple librairie: (xmlspy altova /math.xq)

```
(: Copyright (c) 2004 Altova GmbH, http://www.altova.com :)
(: Library Module containing some mathematical function declarations :)
(: Purpose: test library declaration, recursive user defined functions, numeric
   operators :)
xquery version "1.0" encoding "UTF-8";
module namespace math="http://www.xmlspy.com/xquery/math";
declare function math:NewtonIteration( $n, $k, $e )
{
   if ( abs(( $k * $k - $n) div $e ) < 1 ) then $k
            else math:NewtonIteration( $n, ( ( $k + $n div $k) div 2 ), $e )
};
....
```

Opérateurs

- Opérateurs permettent d'effectuer les traitements courants sur les valeurs, séquences et nœuds.
  - Arithmétiques : (**+**, **-**, **mod**, …)
  - De manipulation de séquences : **Concaténation, union, intersection, différence**
  - De comparaison pour valeurs atomiques, nœuds et séquences
  - Booléens **and**, **or**, **not**

Opérateur: except

Exclusion d'un type de neoud

- Requête

```
<livre>
        {document('livres.xml')//BD[1]/(* except pub)}
    }
</livre>
```

- Résultat

```
<livre>
    <scenariste>Gossini</scenariste>
    <dessin>Uderzo</dessin>
    <prix>18.95</prix>
</livre>
```

```
<livre>
  <BD titre='Asterix le gaulois'>
    <scenariste>Gossini</scenariste>
    <dessin>Uderzo</dessin>
    <pub>Dargaux</pub>
    <prix>18.95</prix>
  </BD>
  <BD  annee='2001' titre='Titeuf'>
    <scenariste>Zep</scenariste>
    <dessin>Zep</dessin>
    <pub>Dargaux</pub>
    <prix>10.00</prix>
  </BD>
 <BD annee='1980' titre='Lucky luck'>
    <scenariste>Gossini</scenariste>
    <scenariste>Moris</scenariste>
    <dessin>Moris</dessin>
    <pub>Dargaux</pub>
    <prix>10.00</prix>
  </BD>
</livre>
```

Opérateur: except

- Requête

<livre>
{document('livre.xml')//BD[1]/(* except pub) }
</livre>

- Fonctionnement
  - Construction d'un noeud **<livre>** avec tous les éléments du premier nœud ***BD*** du document source sauf les nœuds ***pub***.

Concaténation

- Requête

```
<livre> prix de la 3ieme BD suivi des scenaristes
        {document('livre.xml')//BD[3]/(prix,scenariste)}
</livre>
```

- Résultat

```
<livre> prix de la 3ieme BD suivi des scenaristes
     <prix>10.00</prix>
    <scenariste>Gossini</scenariste>
     <scenariste>Moris</scenariste>
</livre>
```

Jointure

– Pas de fonction prédéfinie mais mécanisme simple.

– Soit un autre fichier ISBN.xml qui contient des titres de livres et les codes ISBN correspondant à ces ouvrages.

```
<ISBN>
   <BD titre="Asterix le gaulois">
     <refer>763145-247</refer>
   </BD>
     …
</ISBN>
```

– Ex : Création d'un document résultat contenant pour chaque livre à la fois son titre, sa référence ISBN et le nom de ses scenaristes.

Jointure interne

Requête

```
for $a in document("ISBN.xml")//BD,
    $b in document("livres.xml")//BD
where $a/@titre = $b/@titre
return element livre {attribute titre {$a/@titre},
                      attribute ISBN {$a/refer},
                      element auteur
                          {attribute nom {$b/scenariste}}}
```

Fonctionnement

- Jointure sur: **titre** de biblio.xml et ISBN.xml
- Construction élément **livre** avec attributs **titre** et **ISBN**
- Valeurs des noeuds **titre** et **refer** de ISBN.xml.
- Construction fils **auteur** avec un attribut **nom** (concat scenaristes)

Jointure

Résultat

```
<livre titre="Asterix le gaulois" ISBN="763145-247">
  <auteur nom="Gossini" />
</livre>
<livre titre="Titeuf" ISBN="2368735-2">
  <auteur nom="Zep" />
</livre>
<livre titre="Lucky luck" ISBN="94687-2024">
  <auteur nom="GossiniMoris" />
</livre>
```

Jointure écriture presque équivalente: jointure externe☺

Requête

```
for $a in document("ISBN.xml")//BD
return
   element livre {attribute titre {$a/@titre},
                  attribute ISBN {$a/refer},
                  for $b in document("livres.xml")//BD
                  where $a/@titre = $b/@titre
                  return
                     element auteur {attribute nom{$b/scenariste}}}
```

Plan

XPath et XQuery?

Dans certains cas, on peut écrire soit du Xpath directement soit une expression FLWOR

Quand il y a de la

- construction,
- de la définition de fonctions utilisateurs
- utilisation de variables
- ou encore du tri

→ il faut utiliser du **Xquery**

Quelques pièges à éviter

La recopie de nœud dans la construction

```
let $x := <foo><a/></foo>
let $y :=<b> {$x/a}</b>
return ( $x/a  is  $y/a )
```

true ou false ? False !

Le <a/> qui se trouve dans <b> est une *copie* de l'original <a/>

Que reste t'il à voir?

- La nouvelle norme: Xquery 3: window – group by …
- Update / insert /delete
- Implémentation: performance et complexité

La nouvelle norme xquery3

- **Clause Group By**
- **Clause Switch**
- **Clause Try-Catch**
- **….**

http://atomic.exist-db.org/HowTo/XQuery3/

Plan

Exercice: que retourne cette requête?

```
<livres>
  {for $x in fn:doc("livres.xml")//BD
  where $x/@annee > 1950 and
   some $y in $x/scenariste satisfies starts-with($y,"G")
  return
    <livre annee="{$x/@annee}">
          {element titre {fn:data($x/@titre)} }
        { for $z in $x/( dessin | pub )
            return
                  if(fn:name($z)="pub")
                  then <editeur>{$z/*}</editeur>
                 else  <graphisme>{$z/*}</graphisme>
          }
    </livre>
    }
</livres>
```

```
<livre>
  <BD titre='Asterix le gaulois'>
    <scenariste>Gossini</scenariste>
    <dessin>Uderzo</dessin>
    <pub>Dargaux</pub>
    <prix>18.95</prix>
  </BD>
  <BD  annee='2001' titre='Titeuf'>
    <scenariste>Zep</scenariste>
    <dessin>Zep</dessin>
    <pub>Dargaux</pub>
   <prix>10.00</prix>
  </BD>
 <BD annee='1980' titre='Lucky luck'>
    <scenariste>Gossini</scenariste>
    <scenariste>Moris</scenariste>
    <dessin>Moris</dessin>
    <pub>Dargaux</pub>
   <prix>10.00</prix>
  </BD>
</livre>
```

Exercice: solution

```
<livres>
  <livre annee="1980">
    <titre>Lucky luck</titre>
    <graphisme>Moris</graphisme>
    <editeur>Dargaux</editeur>
  </livre>
</livres>
```

Exercice: document Guide.xml

```
<Guide Version="3.0">
  <Restaurant type="auberge" categorie="**">
        <Nom>Le Vieux Moulin</Nom>
        <Adresse><Rue>du Salève</Rue><Ville>Drize</Ville> </Adresse>
        <Gérant>Dufourneau</Gérant>
  </Restaurant>
  <Restaurant type="italien" categorie="***">
        <Nom>Mamma Mia</Nom>
        <Adresse><Rue>Georges Favon</Rue><Ville>Genève</Ville> </Adresse>
        <Gérant>Maldini</Gérant>
  </Restaurant>
  <Restaurant type="gastronomique" categorie="***">
        <Nom>L'Epicurien</Nom>
        <Adresse><Rue>Bourg-de-Four</Rue><Ville>Genève</Ville> </Adresse>
  </Restaurant>
  <Bar>
        <Nom>Le Tuyau</Nom>
        <Adresse><Rue>Passage Monetier</Rue><Ville>Genève</Ville> </Adresse>
  </Bar>
</Guide>
```

Document Repertoire.xml

```
<Répertoire Version="1.0">
  <Hôtel confort="***">
        <Nom>Suisse</Nom>
        <Adresse><Rue>Cornavin</Rue><Ville NP="1201">Genève</Ville></Adresse>
  </Hôtel>
  <Hôtel confort="**">
        <Nom>Les Tourelles</Nom>
        <Adresse><Rue>Georges Favon </Rue><Ville NP="1204">Genève</Ville></Adresse>
  </Hôtel>
  <Hôtel confort="****">
        <Nom>Edelweiss</Nom>
        <Adresse><Rue>Rigistrasse</Rue><Ville NP="6002">Lucerne</Ville></Adresse>
  </Hôtel>
  <Hôtel confort="***">
        <Nom>Hôtel de la Truite</Nom>
        <Adresse><Rue>Poste</Rue><Ville NP="1342">Le Pont</Ville></Adresse>
  </Hôtel>
</Répertoire>
```

Les questions

- Q1: le nom et l'adresse des restaurants de Genève
- Q2 (jointure): le nom des restaurants se trouvant dans la même rue que l'hôtel Les Tourelles
- Q3 (agrégation): combien y a t il de restaurants dans le guide
- Q4 (imbrication de "for"): lister les valeurs de tous les attributs des hôtels (sous éléments compris) dont le libellé de la rue ou de la ville contient "Pont"

Solution pour Q1

- Q1: le nom et l'adresse des restaurants de Genève

```
FOR $r IN document("Guide.xml")/Guide/Restaurant
WHERE $r/Adresse/Ville = "Genève"
RETURN <Restaurant>
                <Nom> { $r/Nom } </Nom>
                <Adresse> { $r/Adresse/Rue } </Adresse>
        </Restaurant>
```

- Résultat de la requête:

```
<Restaurant>
        <Nom>Mamma Mia</Nom>
        <Adresse>Georges Favon</Adresse>
</Restaurant>
<Restaurant>
        <Nom>L'Epicurien</Nom>
        <Adresse>Bourg-de-Four</Adresse>
</Restaurant>
```

Solution pour Q2

- Q2 (jointure) : le nom des restaurants se trouvant dans la même rue que l'hôtel les Tourelles

```
FOR $r IN document("Guide.xml")/Guide/Restaurant
        $h IN document("Répertoire.xml")/Répertoire/Hôtel
WHERE $r//Rue = $h//Rue AND $h//Nom = "Les Tourelles"
RETURN <Nom> { $r/Nom } </Nom>
```

- Résultat de la requête:

```
<Nom>Mamma Mia</Nom>
```

Solution pour Q3

- Q3 (agrégation): combien y a t il de restaurants dans le guide

```
LET $r:=document("Guide.xml")    //Restaurant
RETURN
    <NombreRestaurants> count($r) </NombreRestaurants>
```

- Résultat de la requête:

```
<NombreRestaurants>3</NombreRestaurants >
```

Solution pour Q4

- Q4 (imbrication de "FOR"): lister les valeurs de tous les attributs des hôtels (sous-éléments compris) dont le libellé de la rue ou de la ville contient "Pont"

```
FOR $h IN document("Répertoire.xml")/Répertoire/Hôtel
WHERE $h/Adresse/(Rue|Ville) CONTAINS ("Pont")
RETURN
        <Result>
                FOR $a IN $h//@* RETURN <Attribut> $a </Attribut>
         </Result>
```

- Résultat de la requête:

```
<Result>
  <Attribut>***</Attribut>
  <Attribut>1342</Attribut>
<Result>
```

Quelques built-in functions

- `fn:document(xs:anyURI)=> document?`
- `fn:empty(item*) => boolean`
- `fn:index-of(item*, item) => xs:unsignedInt?`
- `fn:distinct-values(item*) => item*`
- `fn:distinct-nodes(node*) => node*`
- `fn:union(node*, node*) => node*`
- `fn:except(node*, node*) => node*`
- `fn:string-length(xs:string?) => xs:integer?`
- `fn:contains(xs:string, xs:string) => xs:boolean`
- `fn:true() => xs:boolean`
- `fn:date(xs:string) => xs:date`
- `fn:add-date(xs:date, xs:duration) => xs:date`

Voir Functions and Operators W3C specification ou

http://www.xqueryfunctions.com/xq/