XQuery

Quelques bases

Responsable : Patricia Serrano Alvarado

Master INFO 1ère année

Plan

- Le pourquoi de XQuery
- Le modèle de données
- Les expressions de base et complexes
- Les fonctions
- Références

A-t-on besoin d'un nouveau langage de requêtes ?

- Dans le début des années 2000, la plupart des données sont stockées dans de bases de données relationnelles
- Le langage SQL est mûr et complètement adopté par les entreprises
- Est-ce que SQL est bien adapté pour les données XML ?
- En quoi les données XML sont elles différentes des données relationnelles ?

Imbrication de données

- Les données relationnelles sont plates : lignes et colonnes
- Les données XML sont imbriquées et la profondeur peut être irrégulière et imprédictible
- Les relations peuvent représenter de données hiérarchiques grâce aux clés étrangères ou aux types de données structurées
- En XML il est naturel de chercher les objets à de niveaux inconnus dans la hiérarchie : "Trouve ce qui est rouge"
- XPath est un langage compact et bien adapté pour ce type de requête : //*[@color="Red"]

Métadonnées

- Les données relationnelles sont uniformes et répétitives
  - Tous le comptes de banque ont une structure similaire
  - Les métadonnées peuvent être factorisées dans un système de catalogue
- Les données XML sont très variables
  - Chaque page web est différente
  - Chaque objet XML a besoin d'une auto description
  - Les métadonnées sont réparties tout au long du document
  - Les requêtes peuvent accéder aussi bien aux métadonnées qu'aux données

    "trouve les éléments dont le nom est égal au contenu"

    //*[name(.)=string(.)]

Séquences hétérogènes

- Les requêtes relationnelles renvoient un ensemble uniforme de lignes
- Une requête XML peut renvoyer de types mixtes et de structures complexes
  - Ce qui est rouge : un drapeau, une cerise, une robe, etc.
  - Dans la réponse, les types « element » peuvent être mélangés aux valeurs atomiques
- Les requêtes XML doivent pouvoir faire de transformations structurelles

Ordonnancement

- Les lignes d'une relation ne sont pas ordonnées
  - Dans la réponse à une requête, n'importe quel ordre peut être dérivé à partir des valeurs des relations

- Les éléments dans un document XML sont ordonnés

Information manquante

- Les données relationnelles sont denses
  - Chaque ligne a une valeur dans chaque colonne
  - Une valeur nulle est nécessaire pour les données manquantes

- Les données XML peuvent être clairsemées
  - Les éléments manquantes ou inapplicables peuvent être vides ou tout simplement ne pas apparaître
  - Ceci donne à XML un degré de liberté qui n'est pas présent dans les bases de données relationnelles

En conclusion

- Les données XML sont suffisamment différentes des données relationnelles pour avoir son propre langage

XQuery statuts actuel

- XQuery 1.0: An XML Query Language
- XQuery 1.0 W3C Recommendation January 2007
- W3C Candidate Recommendation 3 November 2005
- XQuery 1.0: An XML Query Language (Second Edition)
- W3C Recommendation 14 December 2010
  - http://www.w3.org/TR/xquery/

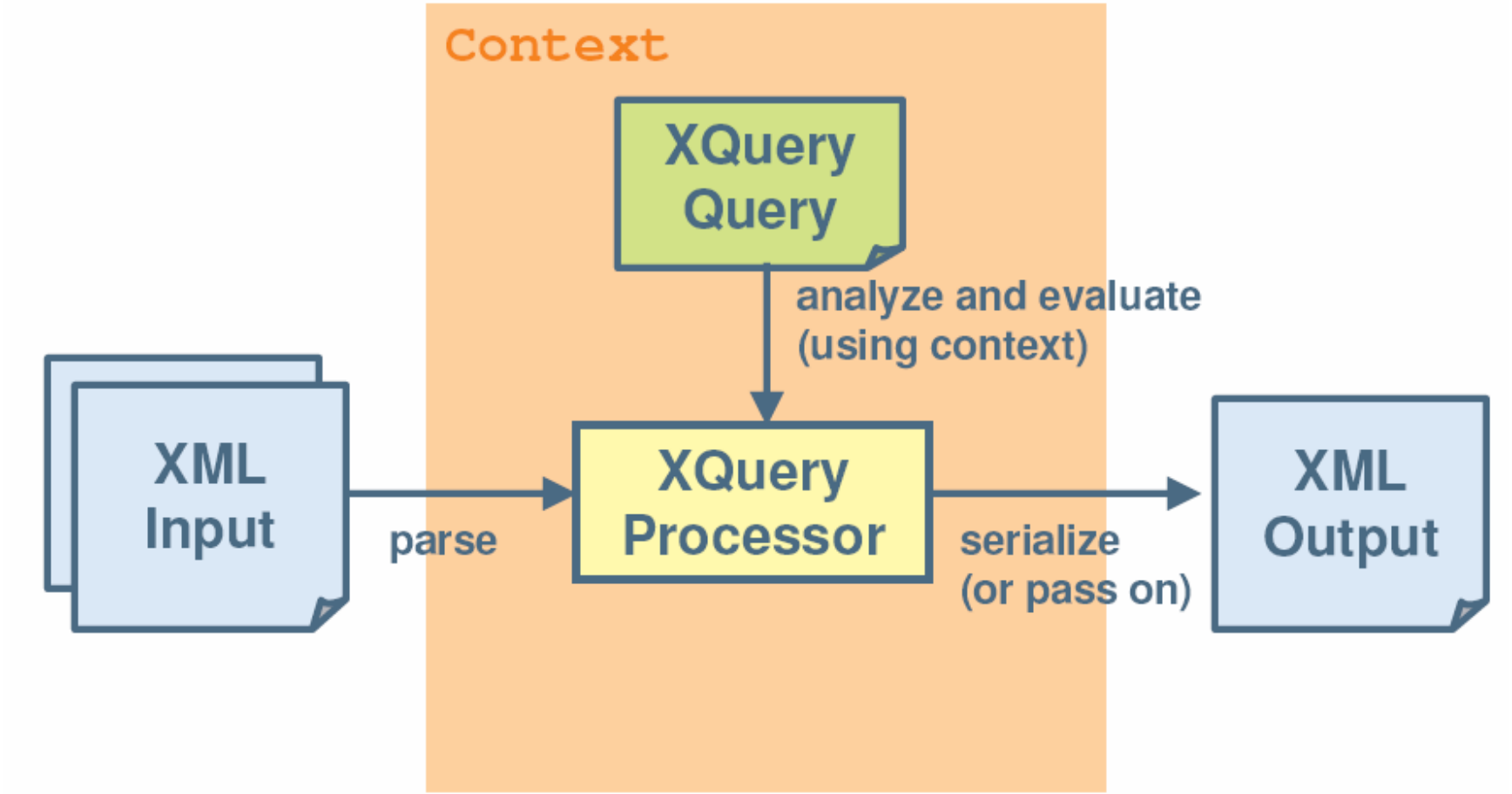
XQuery

- Langage permettant de
  - Sélectionner des éléments/attributs à partir de documents XML
  - Combiner de données à partir de multiples documents
  - Modifier les données
  - Calculer de nouvelles données
  - Rajouter de nouveaux éléments/attributs aux documents sortants
  - (Re)ordonner les résultats
  - Etc.

Exemples d'utilisation

- Extraire de l'information à partir d'une base de données relationnelle afin de l'utiliser dans un service Web
- Générer des rapports de résumés à partir de données stockées dans une base de données XML
- Rechercher des documents textuels sur le Web
- Sélectionner et transformer de données XML en XHTML (ou un autre syntaxe basée sur XML) pour être publiées dans le Web
- Intégration en un document XML plusieurs documents XML
- Distribution en plusieurs documents XML un grand document XML

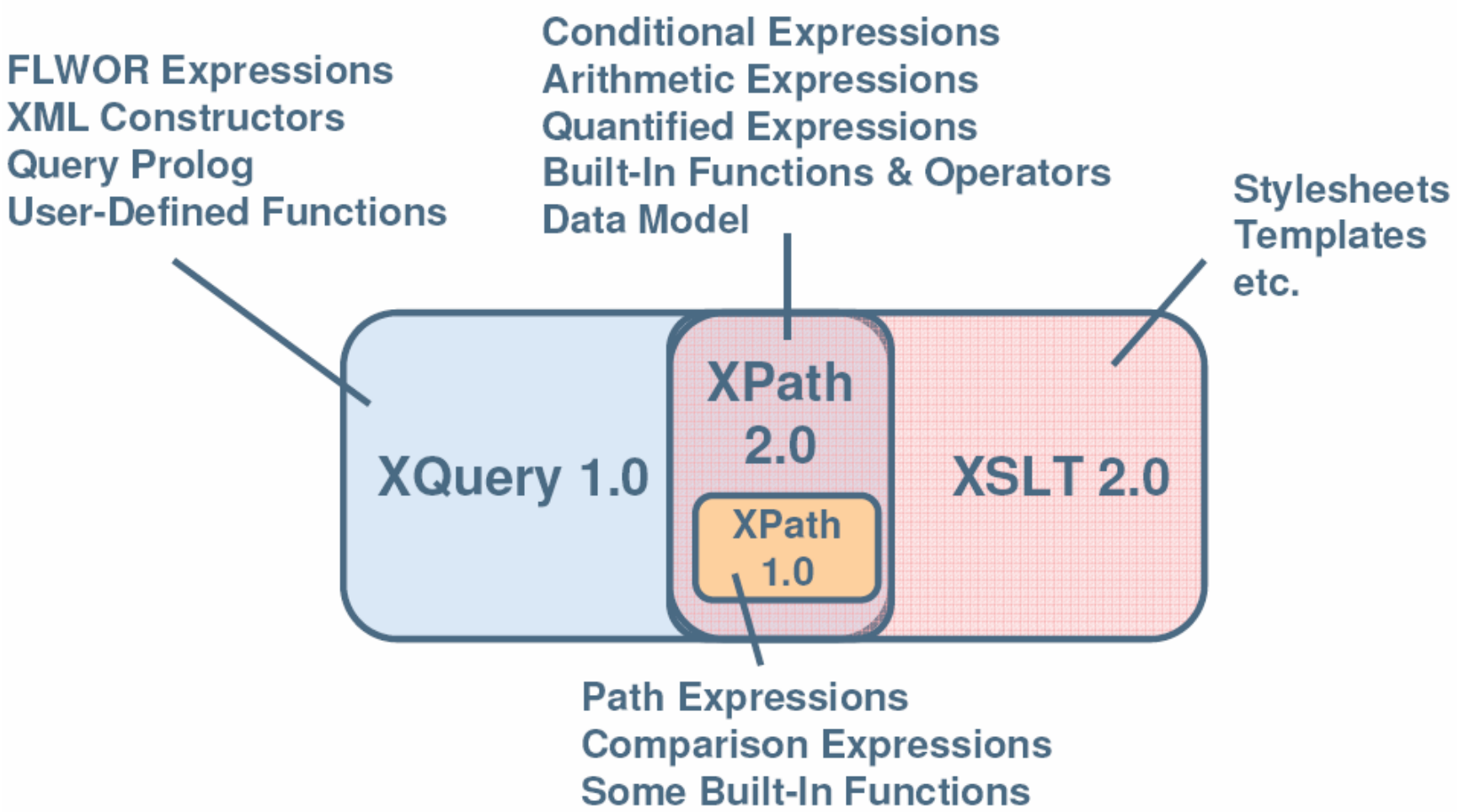
Le modèle de traitement de requêtes



Les documents en entrée

- Fichiers texte qui sont de documents XML
- Fragments de documents XML qui sont récupérés du Web grâce à un URI
- Une collection de documents XML qui sont associés à un URI en particulier
- De données stockées dans de bases de données XML
- De données stockées dans de bases de données relationnelles qui possèdent un *front-end* XML
- De documents XML en mémoire
- Etc.

XQuery, XSLT et XPath



La structure d'une requête XQuery

- Une requête XQuery est formée :
  - D'un prologue composé d'une suite de déclarations (optionnel)
  - D'un corps composé d'une expression dont la valeur est le résultat de la requête.

- La valeur d'une requête est la valeur de l'expression qui constitue son corps, évaluée dans le contexte défini par le prologue.

Exemple

- Livres publiés après 2000 ?

```
declare variable $bib := doc("bib.xml")//book;
declare function published-after($b, $y)
    {$b/@year > $y};
```

Prologue

```
<bib>
    {
    for $b in $bib
    where published-after($b, 2000)
    return
            <book year="{$b/@year}">
                    {$b/title}
            </book>
    }
</bib>
```

Corps

Le modèle de données

- Le modèle de données utilisé par XQuery est celui de XPath avec certaines restrictions
- Une valeur est une séquence ordonnée de zéro ou plusieurs items
- Un item est un nœud ou une valeur atomique
- Sept types de nœuds
  - Document
  - Element
  - Attribute
  - Text
  - Comment
  - Processing instruction
  - Namespace

Hypothèses à propos des valeurs

- Il n'y a pas de distinction entre un item et une séquence d'un seul item
- Il n'y a pas de séquences imbriquées
- Il n'y a pas de valeur nulle
- Une séquence peut être vide
- Une séquence peut contenir de valeurs hétérogènes
- Toutes les séquences sont ordonnées

Quelques règles de syntaxe

- XQuery est un langage sensible à la case tout comme XPath
- Les mots clés sont en minuscules
- Les expressions n'ont pas d'effet de bord
- Toutes les expressions rendent une valeur
- Les expressions sont composables (i.e., elles peuvent être composées d'autres fonctions)
- Pas de caractère de fin de ligne
- Les commentaires sont entourés de parenthèses et deux points
  - (: commentaire ici :)

Types d'expressions

- Expressions de base
- Expressions Path
- Prédicats
- Constructeurs d'éléments
- Expressions FLWR
- Expressions d'itération et d'ordonnancement
- Opérations arithmétiques
- Opérations sur des séquences
- Expressions conditionnelles
- Expressions quantitatives

Concepts et expressions de base

littérale et constructeur

- L'expression la plus simple est une *littérale*
- Une littérale représente une valeur atomique
  - 47 (littérale de type entier)
  - 4.7 E3 (littérale de type double car elle contient une puissance)
  - "4.7" (littérale de type string)
  - '4.7' (littérale de type string)

- Un *constructeur*, dans sa forme la plus simple, permet de créer de valeurs atomiques à partir d'une chaîne de caractères
  - true()
  - false()
  - date ("2006-05-15")

Les parenthèses et les virgules

- Les expressions peuvent être délimitées par de « parenthèses » qui désignent l'ordre dans lequel les expressions sont évaluées
  - (2+4)*5 donne 30 car l'addition est évaluée en premier

- L'opérateur « virgule » concatène deux valeurs pour construire une séquence
  - Exemples de *constructeurs* de séquences
    - (1, (2, 3), ( ), (4)) équivaut à 1, 2, 3, 4
    - 5 to 8 équivaut à 5, 6, 7, 8
    - () séquence vide
    - reverse(10 to 15) équivaut à 15, 14, 13, 12, 11, 10

Les variables et la clause let

- Les *variables* sont identifiées par un nom précédé d'un « $ »

- Les variables sont liées à une valeur avec le mot clé « let »
  - let $debut = 1, $fin = 10

- Les variables peuvent être définies dans plusieurs endroits
  - Prologue des requêtes
  - Expressions FLWR
  - Signature de fonctions

```
for $prod in
        doc("catalogue.xml")// produit
return $prod/num
```

```
declare function local :getNumProduit
      ($prod as element()) as element ()
      {$prod/num};
```

L'appel à fonctions

- XQuery fourni une librairie de fonctions de base
- Le programmeur peut également définir de fonctions
- L'argument des fonctions peut être une seule expression
  - substring($nomproduit, 1, 5)
- L'argument peut avoir besoin d'être d'un certain type

Expressions plus complexes

Expressions Path

- Les expressions Path sont basées sur la syntaxe de XPath
- Tous les axes considérés dans XPath sont supportés par XQuery à l'exception de namespace

- Axes *forward*
  - child
  - descendant
  - attribute
  - self
  - descendant-or-self
  - following-sibling
  - following
- Axes *reverse*
  - parent
  - ancestor
  - ancestor-or-self
  - preceding
  - preceding-sibling

Considérez ce document XML

Fichier books.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<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">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</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>
```

Sélection de nœuds

- Ouvrir le document
  - Utilisation de la fonction doc() qui fait référence à un document via un URI
  - doc("books.xml")
  - doc("http://www.mysite.books.xml")

- Récupération de nœuds
  - Utiliser d'expressions Path
  - Exemple
    - doc("books.xml")/bookstore/book/title

      Résultat :

      <title lang="en">Everyday Italian</title>
      <title lang="en">Harry Potter</title>
      <title lang="en">XQuery Kick Start</title>
      <title lang="en">Learning XML</title>

Les prédicats (1)

- Les prédicats peuvent être
  - Des expressions booléennes
    - book[author = "Mark Twain"]
  - Des expressions numériques
    - chapter[2]
  - De tests d'existence
    - person[@married]
    - book[appendix]

- Les prédicats peuvent être utilisés dans des expressions Path
  - //book[author = "Mark Twain"]/chapter[2]

- Mais également dans d'autre type d'expressions
  - (1 to 100)[. mod 5 = 0]

Les prédicats (2)

- Exemple
  - doc("books.xml")/bookstore/book[price<30]

    Résultat :

```
<book category="CHILDREN">
  <title lang="en">Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

- En utilisant AltovaXML et le fichier queryBooks.xq contenant un query à la fois voici la ligne de commande :
  - AltovaXML /xquery queryBooks.xq

Opérateurs fréquemment utilisés dans les prédicats

- Comparaison de valeurs atomiques
  - eq, ne, lt, le, gt, ge
- Comparaison générale
  - =, !=, <, <=, >, >=
- Comparaison de nœuds en base à leur identité
  - is, is not
- Comparaison d'ordre dans le document
  - <<, >>
- Opérateurs logiques
  - and, or
- Négation
  - not

Opérateurs arithmétiques

- Opérateurs arithmétiques courants
  - +, -, *, div, idiv et mod
  - Si l'un des opérants est une séquence vide ces opérateurs rendent une séquence également vide
- Exemple
  - for $e in $emps

    return

        <emp>

            {$e/name,

            <pay> {$e/salary + $e/commission

            + $e/bonus} </pay>}

        </emp>

    sort by (pay)

Dans cet exemple les attributs $e/commission et $e/bonus sont optionnels. Dans la séquence produite par cette expression si $e/commission ou $e/bonus sont vides, l'élément correspondant sera vide

Opérateurs d'agrégation

- Opérateurs d'agrégation sur de données groupées
  - sum, avg, count, max, min

- Exemple
  - for $cat in distinct-values(doc("books.xml")//book/@category)
    let $books := doc("books.xml")//book[@category=$cat]
    order by $cat
    return <categorie cat="{$cat}" numLivres="{count($books)}"/>

    Résultat :

    <categorie cat="CHILDREN" numLivres="1"/>
    <categorie cat="COOKING" numLivres="1"/>
    <categorie cat="WEB" numLivres="2"/>

Opérateurs sur des séquences

- XQuery fournit des opérations définies pour combiner de séquences
  - union (ou |), intersect et except

- Exemples
  - doc("books.xml")//book[price>30]
    union
    doc("books.xml")//book[year=2005]

  - doc("books.xml")//book
    except
    doc("books.xml")//book[price>30]

Cette expression donne tous les éléments books du fichier books.xml

Cette expression donne les éléments books dont le prix est n'est pas supérieur de 30

Expressions conditionnelles

- Expression if-then-else
- Les trois clauses sont obligatoires
- L'expression if doit être entre ()
- L'expression then peut être vide : then ()

- Exemple
  - for $books in (doc("books.xml")//book)
    return if ($books/@category='WEB')
    then <web>{data($books/title)}</web>
    else <autre>{data($books/title)}</autre>

  Résultat :

  <autre>Everyday Italian</autre>
  <autre>Harry Potter</autre>
  <web>XQuery Kick Start</web>
  <web>Learning XML</web>

Quantificateurs

- Expressions permettant de vérifier si une condition est vrai pour quelques valeurs (some) ou pour chacun des valeurs (every) d'une séquence

- Les quantificateurs retournent *true* ou *false*

- Exemples
  - some $n in (5, 7, 9, 11) satisfies $n > 10

    Résultat : true

  - every $n in (5, 7, 9, 11) satisfies $n > 10

    Résultat : false

Constructeurs (1)

- Un constructeur crée de structures XML dans un query
- Il peut contenir toute sorte d'expression

- Avec de valeurs constantes

```
<highbid status = "pending">
    <itemno>4871</itemno>
    <bid-amount>250.00</bid-amount>
</highbid>
```

Cette expression construit un élément nommé highbid qui contient un attribut status et deux éléments nommés itemno et bid-amount.

- Avec de valeurs basées sur des expressions

```
<highbid status = "{$s}">
    <itemno> {$i} </itemno>
    <bid-amount>
        {max($bids[itemno = $i]/bid-amount)}
    </bid-amount>
</highbid>
```

Dans cet exemple, les valeurs des attributs et des éléments sont calculées par des expressions entre crochés

Constructeurs (2)

- Avec d'expressions

```
<highbid>
  {
    $b/@status,
    $b/itemno,
    $b/bid-amount
  }
</highbid>
```

Dans cet exemple, le constructeur contient une expression entre crochés qui génère un attribut et deux éléments

Constructeur (3)

- les éléments et les attributs peuvent être construits comme de nouveaux nœuds avec des identifiants propres
- Dans ce cas, le nom aussi bien que le contenu sont calculés par des expressions
- Dans le cas des éléments, les nœuds fils possèdent également des identifiants propres même s'ils sont calculés ou copiés à partir de nœuds existants
- Utilisation des mots clés
  - element
  - attribute

Constructeur (4)

- element et attribute doivent être suivis de deux expressions
  - Une expression pour calculer le nom
  - Une expression pour calculer le contenu

- Exemples
  - element

    {name($e)}

    {$e/@*, data($e)*2}

  - attribute

    {if $p/sex="M" then "father" else "mother"}

    {$p/name}

Constructeur (4)

- Ou quand les constructeurs sont constants:

```
element book {
   attribute isbn {"isbn-0060229357" },
   element title { "Harold and the Purple Crayon"},
   element author {
      element first { "Crockett" },
      element last {"Johnson" }
   }
}
```

Clause for

- XQuery permet l'itération sur une séquence de valeurs
- La boucle **for** est la forme la plus simple d'itération
  - for $n in (2, 3) return $n + 1
  - Le résultat de cette itération est (3, 4)
  - La variable $n prend successivement les valeurs de la séquence donnée après le mot clé *in*
- Possibilité de spécifier plus d'une variable
  - for $m in (2, 3), $n in (5, 10)
    return <fact>{$m} times {$n} is {$m * $n}</fact>

Résultat :

<fact>2 times 5 is 10</fact>
<fact>2 times 10 is 20</fact>
<fact>3 times 5 is 15</fact>
<fact>3 times 10is 30</fact>

Le nombre de tuples produits par une série de clause for est le produit des cardinalités des expressions d'itération

Clause FLWR (1)

- Une expression FLWR est une itération plus générale que for

- Cette expression permet d'avoir
  - multiples clauses for,
  - multiples clauses let,
  - une clause optionnelle where et
  - une clause return

- FLWR vient de for, let, where et return

Le document books.xml

Fichier books.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<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">XQuery Kick Start</title>
  <author>James McGovern</author>
  <author>Per Bothner</author>
  <author>Kurt Cagle</author>
  <author>James Linn</author>
  <author>Vaidyanathan Nagarajan</author>
  <year>2003</year>
  <price>49.99</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>
```

Clause FLWR (2)

- Exemples
  - for $books in doc("books.xml")/bookstore/book
    where $books/price<30
    return $books/title

    Résultat : <title lang="en">Harry Potter</title>

    Cette requête donne le titre des livres dont le prix est inférieur à 30

  - for $x in doc("books.xml")/bookstore/book
    where $x/@category='WEB'
    order by $x/title
    return <titre>{data($x/title)}</titre>

    Résultat :
    <titre>Learning XML</titre><titre>XQuery Kick Start</titre>

    Cette requête ignore l'attribut lang et donne le contenu des titre des livres dont la catégorie est WEB et cela de manière ordonnée

Clause FLWR (3)

- Exemples
  - for $i in (1 to 3)
    let $j : (1 to $i)
    return ('(',$i,', ', $j, ') ')

    Résultat : (1, 1) (2, 12) (3, 123)

    Cette expression produit la séquence de trois paires de valeurs
    $i=1, $j=1
    $i=2, $j=(1,2)
    $i=3, $j=(1,2,3)

  - Le nombre de tuples produits par une série de clause for et let est équivalent au produit des cardinalités des expressions d'itération de la clause for (dans l'exemple 3)

Clause FLWR (4)

- Exemples
  - for $i in document("items.xml")/*/item
    let $b := document("bids.xml")/*/bid[itemno = $i/itemno]
    where count ($b) > 10
    return
    <popular-item>
    {$i/itemno,
    $i/description,
    <bid-count> {count ($b)} </bid-count>}
    </popular-item>

Cet exemple illustre comment une expression FLWR peut avoir de caractéristiques similaires aux jointures de SQL

Les fonctions XQuery (1)

- Il existe à peu prés 100 fonctions XQuery

- Les opérateurs et les fonctions prédéfinies sont communs à XPath et XQuery et sont décrits dans : http://www.w3.org/TR/xpath-functions/

- Quelques fonctions :
  - String
    - substring, contains, matches, concat, normalize-space, tokenize
  - Date
    - current-date, month-from-date, adjust-time-to-timezone
  - Nombre
    - round, avg, sum, ceiling
  - Séquence
    - index-of, insert-before, reverse, subsequence, distinct-values

Les fonctions XQuery (2)

- Nœud
  - data, empty, exists, id, idref
- Nom
  - local-name, in-scope-prefixes, QName, resolve-QName
- Erreur
  - erreur, trace, exactly-one
- Document et URI
  - collection, doc, root, base-uri

Fonctions définies par l'utilisateur (1)

- Une fonction
  - Peut prendre zéro ou plusieurs paramètres
  - Peut retourner un résultat
  - Est composée d'une signature et d'un corps

- La signature de la fonction est composé de
  - Mot clé « declare function »
  - Nom de la fonction
  - Les paramètres
  - Le résultat

- Le corps de la fonction est une expression entre { }

- XQuery ne supporte pas la surcharge dans la définition de fonctions

- L'appel récursive de fonctions est supportée

Fonctions définies par l'utilisateur (2)

- Exemple
  - declare function highbid(element $item) as decimal

    {max(document("bids.xml")//bid[itemno= $item/itemno]/bid-amount)}

- Appel à la fonction
  - highbid(document("items.xml")//item[itemno = "1234"])

Fonctions définies par l'utilisateur (3)

- Les paramètres et la valeur retournée peuvent être optionnels s'ils sont déclarés avec un signe d'interrogation

- Exemple
  - declare function defaulted (element? $e, anySimpleType $d)
    as anySimpleType
    {
      if (empty($e)) then $d
      else if (empty($e/*)) then $d
      else data($e)
    }
  - Appel à la fonction
    - defaulted ($e/commission, 0)

Déclaration de variables globales

- Une variable peut être déclarée dans le prologue d'une requête :

declare variable $*n* as *t* := *exp*;

- $*n* est le nom de la variable,
- *t* (optionnel) est un type de séquence, le type de la variable,
- *exp* est une expression, l'expression de la variable, dont la valeur sera celle de la variable

- Exemple
  - declare variable $books := doc("books.xml ");

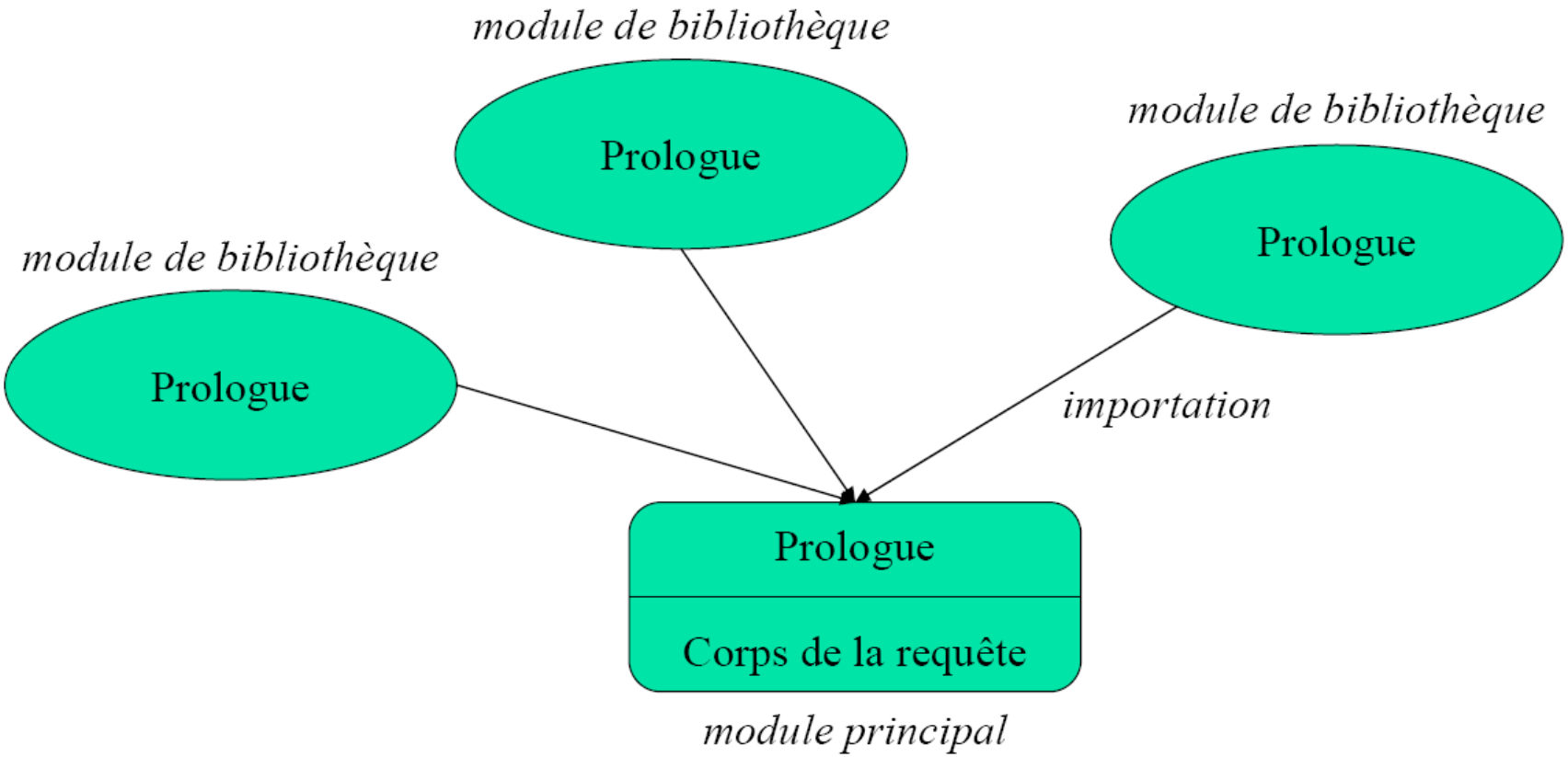
Modules

- Une requête XQuery peut être construite à partir d'un ou plusieurs modules

- Un module est une unité compilable séparément qui contient un ensemble de déclarations : le prologue et éventuellement une expression à évaluer (le corps d'une requête)

- Deux types de modules :
  - Les modules principaux comportant un prologue et le corps d'une requête
  - Les modules de bibliothèque sont réduits à un prologue

Structure d'une requête XQuery



Prologue

- Le prologue d'une requête peut comporter les déclarations suivantes :
  - Déclaration de traitement des espaces,
  - Déclaration d'un URI de base,
  - Importation de schémas
  - Importation de modules,
  - Déclaration d'espace de noms,
  - Déclaration d'espace de noms par défaut pour les éléments et les fonctions,
  - Déclaration de variable globales,
  - Déclaration de fonctions,
  - …

Références

- Don Chamberlin. XQuery: An XML query language. IBM SYSTEMS JOURNAL, VOL 41, NO 4, 2002.
- Tutorial XQuery : http://www.w3schools.com/xquery/
- XQuery 1.0: An XML Query Language. W3C Candidate Recommendation 3 November 2005http://www.w3.org/TR/xquery/
- An Introduction to XML and Web Technologies. Anders Moller et Michel Schwartzbach. Ed. Addison Wesley 2006.
- Querying XML with XQuery (Advances in Database Systems) by Yannis Papakonstantinou and Ioana Manolescu. Springer; March 2006
- XQuery: The XML Query Language by Michael Brundage; Addison-Wesley Professional, February 2004. With a foreword by Michael Rys. Book web site at www.qbrundage.com/xquery/
- Definitive XQuery by Priscilla Walmsley. Prentice Hall; March 2006
- Altova XML : http://www.altova.com/altovaxml.html
  - XML validating parser, XSLT 1.0 engine, XSLT 2.0 engine (schema-aware), XQuery 1.0 engine (schema-aware)
- Fonctions Xquery : http://www.xqueryfunctions.com/xq/