XQuery

Pierre-Alain Reynier
http://www.lif.univ-mrs.fr/~preynier/XML/

Nous avons déjà vu :

- XML : présentation, exemple de XHTML
- Descriptions : DTDs, XML Schémas
- Navigation : XPATH
- Transformations : XSLT
- Implémentations Java : SAX, DOM

Aujourd'hui : XQuery

1- Présentation générale

2- Apprentissage par l'exemple

3- Implémentations / Liens avec SQL

4- Extension : XQuery Update

5- Une implémentation : eXist

XQuery :
Présentation générale

XQuery

- Langage d'interrogation de données développé par le W3C. Réunion de plusieurs langages (XML-QL, YATL, Lorel, Quilt)
- Recommandation W3C du 23/01/2007 : http://www.w3.org/TR/xquery/
- XQuery n'utilise pas la syntaxe XML, il existe une version XML intitulée XQueryX
- Certains moteurs de bases de données comme MonetDB ou eXist supportent les requêtes XQuery

Pourquoi XQuery ?

- XQuery s'impose comme le langage de requêtes
  - pour les bases de données XML natives
  - pour les documents XML textuels (XQuery Text)
  - pour l'intégration de données (BD virtuelles)
- Interrogation de bases relationnelles en XQuery
  - pour l'intégration et la publication de données
  - compétition avec les extensions de SQL

XQuery : généralités

- XSLT vs XQuery :
  - XSLT est adapté à la transformation de docs XML
  - XQuery est adapté à la recherche efficace de données dans de grandes collections de docs XML
  - dans certains cas, les deux peuvent être utilisés
  - XQuery est un langage typé
- Langage déclaratif analogue à SQL
  - SQL manipule des tables (ensemble de n-uplets)
  - XQuery manipule des séquence d'arbres (= forêts)

XQuery : généralités (suite)

- Langage puissant mais complexe
  - complexité des structures manipulées (arbres XML)
  - semi-structuration des données (problèmes de typage)
- Langage fonctionnel
  - requête = expression évaluée dans un certain contexte
  - valeur de la requête = séquence d'items
  - item = valeur atomique ou noeud de l'arbre document

XQuery : principes de base

- Requête XQuery =
    - prologue composé d'une suite de déclarations (namespaces, variables, fonctions...)
    - corps composé d'une expression dont la valeur est le résultat de la requête

```
declare function local:books-by-author ($root, $last, $first) {
   [....]
   return $b/title
};

<results> {
let $a := doc("biblio.xml")//author
for $last in distinct-values($a/last),
[...]   }
</results>
```

XQuery : principes de base

- XQuery = extension de XPATH 2.0
- Les expressions XPATH sont interprétées en XQuery
- Forme de requête la plus générale = Boucles

FLOWR expressions

```
for $<var> in <forest> [, $<var> in <forest>]+ //itération
let $<var> := <subtree>                         // assignation
order by $<var> descending                      // tri
where <condition>                               // élagage
return <result>                                 // construction
```

XQuery vs XSLT

```
   <html>
<head>
 <title> Liste de documents</title>
</head>
<body>
 <ol>
 {
 for $file in doc("database.xml")//file
 return
   <li><a href="{concat("http://www.lif.fr/",string($file/@url))}">
     {string($file/@name)}</a>
   </li>
 }
  </ol>
 </body>
</html>
```

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <html>
    <head>
      <title>Liste de documents</title>
    </head>
    <body>
      <ol>
        <xsl:apply-templates select="//file"/>
      </ol>
    </body>
  </html>
</xsl:template>

<xsl:template match="file">
  <li>
    <a href="concat('http://www.lif.fr/',./@url)"/>
      <xsl:value-of select="./@name"/>
    </a>
  </li>
</xsl:template>

</xsl:stylesheet>
```

XQuery : par l'exemple

XQuery : par l'exemple

- Exemples considérés extraits de :
  - XML Query Use Cases, Use Case XMP
    http://www.w3.org/TR/xquery-use-cases/
    - Les numéros des requêtes renvoient à cette numérotation
- Parsing en ligne de requêtes XQuery :
  http://www.w3.org/2007/01/applets/xqueryApplet.html

Document `bib.xml`

```
<bib>
 <book year="1994">
  <title>TCP/IP Illustrated</title>
  <author><last>…</last><first>W.</first></author>
  <publisher>Addison-Wesley</publisher>
  <price> 65.95</price>
 </book>
 <book year="1992">
  <title>Advanced Programming in the Unix …</title>
  <author><last>…</last><first>W.</first></author>
  <publisher>Addison-Wesley</publisher>
  <price>65.95</price>
 </book>
…
```

Document `bib.xml` (suite)

```
<book year="2000">
 <title>Data on the Web</title>
 <author><last>…</last><first>S.</first></author>
 <author><last>…</last><first>P.</first></author>
 <author><last>…</last><first>D.</first></author>
 <publisher>Morgan Kaufmann Publishers</publisher>
 <price>39.95</price>
 </book>
 <book year="1999">
  <title>The Economics of Technology and ...</title>
  <editor><last>Gerbarg</last><first>Darcy</first>
<affiliation>CITI</affiliation></editor>
  <publisher>Kluwer Academic Publishers</publisher>
  <price>129.95</price>
 </book>
</bib>
```

Premiers exemples XPath

Titres des livres du document :

```
<title>TCP/IP Illustrated</title>
<title>XML Bible</title>
<title>Advanced Programming in the Unix environment</
title>
<title>Data on the Web</title>
<title>The Economics of Technology and Content for Digital
TV</title>
```

Requête XQuery (= XPath !!) :

```
doc("biblio.xml")//book/title
```

Premiers exemples XPath

Titres des livres du document publiés après 1994 :

```
<title>Data on the Web</title>
<title>The Economics of Technology and Content for Digital
TV</title>
```

Requête XQuery (= XPath !!) :

```
doc("biblio.xml")//book[@year>1994]/title
```

Premiers exemples XPath

Titres des livres du document publiés après 1994, dont le prix est inférieur à 100 dollars :

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

Requête XQuery (= XPath !!) :

```
doc("biblio.xml")//book[@year>1994][price<100]/title
```

Premiers exemples XPath

Titres des livres du document publiés après 1994, résultats regroupés dans une balise bib :

```
<bib>
  <title>Data on the Web</title>
  <title>The Economics of Technology and Content
    for Digital TV
  </title>
</bib>
```

Requête XQuery : (Observez la différence !!)

```
<bib>
    {doc("biblio.xml")//book[@year>1994]/title}
</bib>
```

Q1. Enoncé et réponse attendue

• Titre et année des livres publiés par Addison-Wisley après 1991 (document reformatté, résultats regroupés dans une balise bib):

```
<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
  </book>
  <book year="1992">
    <title>Advanced Programming in …</title>
  </book>
</bib>
```

Q1. Requête : synthèse

```
<bib>
   {
   for $b in document("bib.xml")//book
   where $b/publisher = "Addison-Wesley" and
         $b/@year > 1991
   return
      <book year="{$b/@year}">
         {$b/title}
      </book>
   }
</bib>
```

Remarque sur les accolades

Les accolades permettent de distinguer
- les éléments statiques (balises à écrire dans le document produit)
- les éléments à évaluer (expressions XPath, boucles..)

```
<bib>
    {
    for $b in document("bib.xml")//book
    where $b/publisher = "Addison-Wesley" and
        $b/@year > 1991
    return
        <book year="{$b/@year}">
            {$b/title}
        </book>
    }
</bib>
```

Remarque sur les accolades

L'enchaînement de plusieurs éléments à évaluer peut se faire :

- en entourant chaque élément d'accolades, ou
- en séparant les éléments par des virgules

```
for $b in document("bib.xml")//book
return
   <res>
      {$b/t}
      {$b/p}
      {$b/c}
   </res>
```

```
for $b in document("bib.xml")//book
  return
     <res>
        { $b/t, $b/p, $b/c }
     </res>
```

Q2. Enoncé et réponse attendue

• Créer une liste de paires titre-auteur avec chaque paire incluse dans un élément `result` :

```
<results>
 <result>
  <title>Data on the Web</title>
  <author><last></last><first>S.</first></author>
 </result>
 <result>
  <title>Data on the Web</title>
  <author><last>...</last><first>D.</first></author>
</result>
...
</results>
```

Q2. Requête : jointure

```
<results>
    {
    for $b in document("bib.xml")//book,
        $t in $b/title,
        $a in $b/author
    return
        <result>
            {$t}
            {$a}
        </result>
    }
</results>
```

Q4. Enoncé et réponse attendue

• Pour chaque auteur, lister le nom de cet auteur et les titres des livres qu'il a écrits, regroupés dans un élément `result` :

```
<results>
 <result>
  <author><last></last><first>S.</first></author>
  <title>Data on the Web</title>
 </result>
 <result>
  <author><last></last><first>W.</first></author>
   <title>TCP/IP Illustrated</title>
   <title>Advanced Programming in the Unix …</title>
   </result>
...
</results>
```

Q4. Requête : imbrication

```
<results>
{
let $a := document("bib.xml")//author
for $last in distinct-values($a/last),
$first in distinct-values($a[last=$last]/first)
order by $last, $first
return
    <result>
      <author> <last>{$last}</last>
      <first>{$first}</first> </author>
      { for $b in document("bib.xml")/bib/book
        where some $ba in $b/author
        satisfies ($ba/last = $last and
                   $ba/first=$first)
        return $b/title }
    </result>
}
</results>
```

Q4. Requête : avec déf. de fonction

```
declare function books-by-author ($last, $first)
as element()* {
   for $b in document("bib.xml")/bib/book
   where some $ba in $b/author
   satisfies ($ba/last = $last and
              $ba/first=$first)
   return $b/title  };
<results> {
let $a := document("bib.xml")//author
for $last in distinct-values($a/last),
$first in distinct-values($a[last=$last]/first)
order by $last, $first
return <result>
   <author> <last>{$last}</last>
   <first>{$first}</first> </author>
   books-by-author($last, $first) </result>
   }
</results>
```

Q6. Enoncé

• Pour chaque livre qui a au moins un auteur, lister le titre et les deux premiers auteurs et un élément vide “et-al” si le livre a plus de deux auteurs

Q6. Requête : count

```
<bib>
    {
     for $b in document("bib.xml")//book
    where count($b/author) > 0
    return
        <book>
        {$b/title}
        { for $a in $b/author[position() <= 2]
          return $a }
        { if (count($b/author) > 2
          then <et-al/> else () }
         </book>
      }
</bib>
```

Opérations sur les listes

XQuery propose des opérateurs pour manipuler les listes:

- concaténation
- opérations ensemblistes (union, intersection, différence)
- fonctions : remove, index-of, count, avg, min, max...

Exple: donner pour chaque éditeur le prix moyen de ses livres

```
for  $p in
   distinct-values(document("bib.xml")//publisher)
   let $a :=
      avg(document("bib.xml")//book[publisher=$p]/price)
   return
      <publisher>
         <name>{ $p/text() }</name>
         <avgprice>{ $a }</avgprice>
      </publisher>
```

Rq: comparaison de listes : L1 = L2 <=> disjonction sur les comp. elt à elt

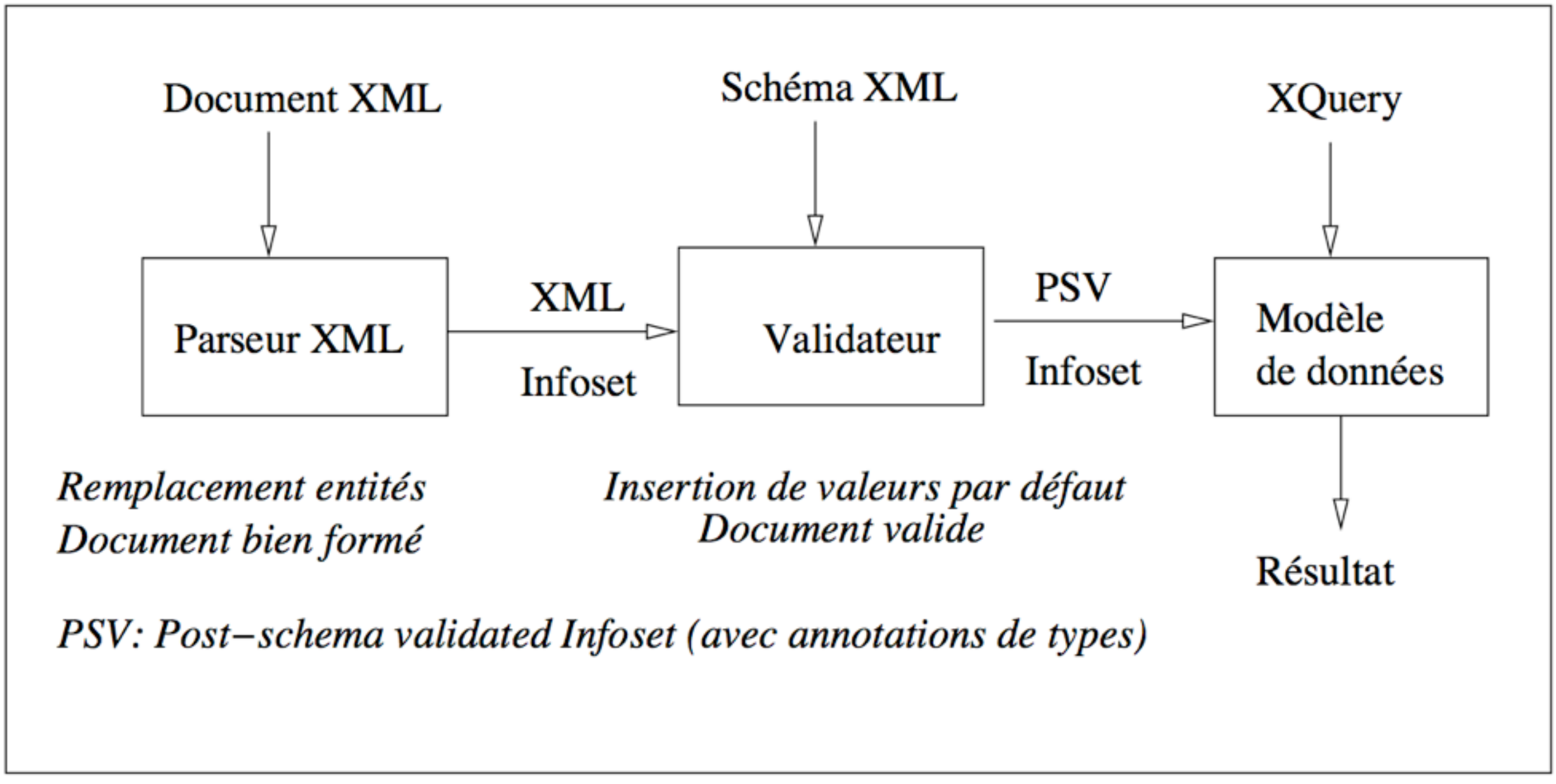
Quantifications

XQuery permet d'exprimer facilement les quantifications existentielles et universelles : some, every, satisfies

Exemple : Retrouver les documents mentionnant les activités "windsurfing" et "sailing"

```
for   $b in document("bib.xml")//book
where some $p in $b/paragraph
   satisfies (contains($p,"sailing")
      and contains($p,"windsurfing"))
return $b/title
```

Modèle d'un traitement XML



Aller plus loin : la W3C

- Les documents de la W3C dispos à http://www.w3.org sont longs mais exploitables : l'introduction clarifie le rôle du document et permet d'aller directement aux (sous-) sections intéressantes
- Spécifications XML (DTDs, Namespaces, XML Schémas)
- Spécifications XQuery
  - syntaxe de XQuery 1.0
  - fonctions XPath et opérateurs (op:equal, fn:text, fn:distinct-values...)
  - Modèle de donnée XQuery

Implémentations / Liens avec SQL

Implémentations de XQuery

Parmi celles qui sont libres et/ou open-source :

- Galax : complet mais pas très efficace
- Saxon : bien, efficace, fait également XSLT
- MonetDB : parmi les plus rapides
- eXist : interface très agréable
- QizX : assez bien
- BerkeleyDB XML : appartient à présent à Oracle

Liens avec SQL

Les versions récentes (2003) de SQL incluent :

- un type atomique natif XML (interrogeable à la XQuery)
- un ensemble de fonctions de publications XML permettant de créer des éléments XML à partir d'une BD relationnelle
- règles de mapping : export de tables relationnelles en XML

Avantages :

- manipulation unifiée
- efficacité des requêtes relationnelles bien exploitée
- transformation aisée

Désavantage :

- complexité

Liens avec SQL

Fonctions de publication XML :

```
select xmlelement(name Customer,
                  xmlattribute(c.city as city),
                  xmlforest(c.CustID, c.Name as CustName))
from customer c
```

Requêtes mixtes :

```
select customer, XMLExtract(order, '/order/@date')
from orders
where XMLExists(order,
                '/order[//desc/text()="Shoes"]') =1
```

La syntaxe exacte dépend parfois du fournisseur.

XQuery Update

XQuery Update

XQuery est un langage read-only : peut retourner une instance, mais pas modifier la BD.

Parallèle avec SQL :

```
select... from... where ...
```

sans

```
insert into table... update table...
```

En pratique, besoin de modifier les données XML. XQuery Update Facility :"Candidate Recomm." du 9 juin 2009 (devrait bientôt être une Recomm.)

http://www.w3.org/TR/xquery-update-10/

XQuery Update

Pouvoir expressif :

- Insert
- Delete
- Update
- Copy avec nouvelle identité

Extension de XQuery :

- simplifie la compréhension du langage
- difficulté d'ajouter des effets de bord

Sémantique bien définie, concision, implém. efficace

XQuery Update

Les requêtes sont classifiées en deux types :

- avec mise à jour
- sans mise à jour

5 nouveaux types d'expressions :

- insert, delete, replace, rename : avec màj
- transform : sans màj

Gestion d'une liste des mises à jour à effectuer

Exemples - insertions

```
insert node <year>2005</year>
after document(“bib.xml”)/books/book[1]/published
```

```
insert node $article/author
as last into document(“bib.xml”)/books/book[3]
```

```
insert nodes {$new-police-report}
as last into document(“insurance.xml”)//policies/
  policy[id=$pid]/driver[licence=$licence]/
  accident[date=$dateacc]/police-reports
```

Pour chaque élément dans `$new-police-report`, ajouter à la liste des updates un ajout de dernier fils à l’élément `police-reports` correspondant.

Exemples - substitutions

```
replace document("bib.xml")/books/book[1]/publisher
with document("bib.xml")/books/book[2]/publisher
```

Augmentation du prix d'un livre :

```
replace value of document("bib.xml")/books/book[1]/price
with document("bib.xml")/books/book[1]/price*1.1
```

Mise à jour d'un inventaire :

```
for $p in /inventory/part
let $deltap := $changes/part[partno = $p/partno]
return
  replace value of $p/quantity
  with $p/quantity + $deltap/quantity
```

XQuery / SQL

Fonction	Relationnel	XML
Query	SQL select	XQuery
Update	SQL update	XQuery Update
Full-text	SQL MMS	XQuery Full-Text
Scripting	PL/SQL	XQuery Scripting

XQuery update n'est pas un langage de programmation. Il manque :

- contrôle de la portée des actions
- contrôle de l'atomicité
- combiner résultat et effets de bord
- traitement des erreurs

Un langage de scripts

Il existe un tel langage pour XQuery : XQuery-P

- définit un mode d'exécution séquentiel
- définit des blocs d'instructions avec de nouvelles variables
- on peut assigner ces variables

Ceci impose de définir un ordre sur l'évaluation d'une expression XQuery : grosse différence avec le langage traditionnel de requête (lequel de `select`, `from` et `which` doit être évalué d'abord ?)

Implémentations

XQuery Update est implémenté dans :

- eXist
- MonetDB

eXist

Une BD en XML : eXist

- Nous utiliserons en TP l'outil eXist pour maintenir au travers d'une interface web une BD en XML.
- Cet outil nous permet de :
  1. intégrer des documents XML
  2. interroger la base données (XQuery)
  3. modifier la base de données (XQuery Update)
  4. appliquer des feuilles de style aux documents produits (XSLT).

Principes généraux

- Le serveur eXist permet une interprétation en ligne des requêtes XQuery.

- On crée des requêtes qui produisent du html, de sorte que le résultat soit lisible par un browser.

- Les fichiers XQuery doivent être stockés sur le serveur.

- On peut passer des arguments à l'aide de la méthode GET de html.

Exemple 1 : initialisation

....

```
let $collection := "xmldb:exist:///db",
$dummy := xmldb:store($collection, "films.xml",
xs:anyURI("http://www.lif.univ-mrs.fr/~preynier/XML/files/films.xml"))

return
<html>...
<p>   {
    if (not ( doc( "/db/films.xml" ))) then
        <p> Erreur : la base de donnees est vide. </p>
    else
        <p> La base de donnees a ete (re)initialisee. </p>    }
</p>
....
</html>
```

Exemple 2 : affichage XQuery

```
...
declare function local:print_film($film){
<div> <p>
Titre : {$film/TITRE/text()}<br/>
Genre : {$film/GENRE/text()}
</p> </div>
};
```

```
<html>
....
    <ul>  {
       for $film in doc("/db/films.xml")//FILM
       return
     <li> {local:print_film($film)}  </li>
       }   </ul>
....
</html>
```

Exemple 3 : affichage XSLT

```
...
declare namespace transform="http://exist-db.org/xquery/transform";
...

if (not ( doc( "/db/films.xml" ))) then
   <html>
   ...
   Erreur : la base de donnees est vide.
   ...
   </html>
else
   transform:transform(doc( "/db/films.xml" )/FILMS,
                              xs:anyURI("affiche.xsl"),<parameter/>)
```