Bases de données natives XML

Plan général

Plan général

- **Révisions**
  - XDM
- **Histoire des bases XML**
  - Comment stocker ses documents ?
- **XQuery**
  - FLOWR
  - Jointures
  - Groupage
  - Fenêtres
- **XQuery et fonctions**
  - Appel de fonctions
  - Définition de fonctions
  - Modules de fonctions
- **Modification des documents**
  - Update Facility
  - ML Update
  - eXist Update Extension
- **Bases XML et Web**
  - RestXQ
  - Applications Web et XQuery

Révisions

XDM

- **XML Data Model**
  - Le modèle de données derrière un document XML
- **Il autorise 3 types d'objets**
  - Des noeuds
    - Eléments
    - Attributs
    - Texte
  - Des valeurs atomiques
    - Celles définies par les types simples de XML Schéma
  - Des fonctions
    - Même si une fonction n'est pas contenue dans un document XML
    - XDM est ouvert et prévoit de stocker des objets de type fonction

XDM

- **On peut organiser ces objets en séquences**
  - Une liste ordonnée, possiblement hétérogène, de noeuds, de valeurs atomiques, de fonctions
  - La séquence vide est appelée **`empty sequence`**
  - Une séquence contenant une séquence est aplatie
    - Pas de séquences imbriquées
- **Une séquence se construit avec des parenthèses**
  - **`('chaîne',12.54,p[refDoc])`**
  - **`('chaîne',('de','vélo'))`** -> **`('chaîne','de','vélo')`**

Introduction

Hégémonie des bases relationnelles

- **Longtemps, on a utilisé des bases de données relationnelles**
  - Nombreuses qualités, transactionnelles, performantes en écriture / mises à jour
  - Capacité de paramétrer les index pour optimiser les recherches
  - Grande liberté sur la modélisation des données, réponse aux besoins métiers
  - Langage d'écriture et de recherche normalisé
    - Et donc portabilité du code !
- **Mais pas adaptées au stockage de données hétérogènes**
  - De structure peu prévisible, voire totalement inconnue
  - Le pire des cas étant les documents textuels structurés
- **On a alors utilisé des contournements**
  - Basés sur des systèmes clé valeur et/ou des modèles génériques
  - Du coup pas du tout performants

Disparition du SGML

- **2001 voit la fin de SGML**
  - Les outils permettant la manipulation sont tous en fin de vie
    - Principalement pour des raisons économiques
  - SGML est trop compliqué à apprendre
    - Les coûts de formation sont importants
    - La normalisation des méthodes est impossible
- **Les éditeurs - documentaires - commencent à regarder XML**
  - Se demandent comment stocker leurs documents existants
  - XSLT existe depuis 2 ans, XML semble être une opportunité
- **Besoin de stocker les documents dans une base**
  - Pour requêtage

Arrivée des bases NoSql

- **En 2000-2010, les bases NoSql sont apparues**
  - Capacité à stocker des données non structurées
  - Recherches full-text efficaces
  - Scalabilité horizontale
- **Peu sont encore efficaces pour stocker des documents**
  - Les modèles en grappes regroupent des données cohérentes entre elles, pour des questions de performance
  - Les modèles en graphes représentent efficacement des relations entre objets, et parcourent ces relations
  - Les modèles clé/valeur sont très efficaces, mais les jointures sont très difficiles
- **Les premiers travaux autour des `NXB` commencent en 2000**
  - Native XML Database
    - https://www.xml.com/pub/a/2001/10/31/nativexmldb.html

Bases Natives XML

- **A l'origine, une expression de besoin**
  - La base est spécialisée pour stocker des données XML
  - Elle conserve toutes les informations du modèle intactes
  - Les documents entrent dans la base et sont restitués
  - Une NXD n'est pas forcément une base autonome
    - Standalone database
- **Les premiers NXD apparaissent en 2001**
  - dbXml
- **Il faudra 15 ans avant de pouvoir utiliser des NXD en production**
  - Pour de grosses volumétries
  - Avec fiabilité
- **Mais l'expression de besoin initiale est respectée**

Les acteurs

- **Plusieurs acteurs se partagent le marché**
  - Open Source ou dual license
    - eXist-db
    - BaseX
    - Fusion DB
  - Commerciale
    - MarkLogic
    - Documentum
    - Tamino

Que mettre dans quoi ?

Data-centric ou Document-centric ?

- **Avec XML, on représente deux catégories de données**
  - Documents Data-Centric
    - Les documents data-centric utilisent XML pour le transport et la persistance
    - Dans le temps, la structure XML n'est pas importante
    - L'ordre des données n'est pas significatif
    - Peu ou pas de contenu mixte
    - Tout ce qui parle d'argent...
    - Données consommées par des machines.
  - Documents Document-Centric
    - Ecrits par des humains, pour des humains
    - Beaucoup de mixed-content
    - Structure non totalement prévisible, malgré des grammaires

Destination

- **Documents Data-Centric dans des bases traditionnelles**
  - Bases relationnelles, objets, ou hiérarchiques
- **Les documents Document-Centric sont stockés dans des NXB**
  - Pour conserver les notions d'ordre, de modèle, de contenu mixte
  - Pour pouvoir extraire une partie des documents stockés sans altérer leur structure
- **Ces règles ne sont pas strictes**
  - En fonction des usages qu'on doit faire des data (ou documents)
  - On pourra stocker des data dans des NXB, et des documents dans des SGBD

Accès aux données

Langages

- **A l'origine, les premières bases utilisaient XPath**
  - Permet d'accéder à des noeuds ou des valeurs d'un arbre
  - Pas multi-document, donc doit être enrichi
  - Apparition de la notion de `collection`
- **Rapidement apparait XQuery**
  - Langage non XML, permettant de requêter et de construire du XML
  - A ce jour, c'est le langage universellement utilisé par toutes les bases.
  - Chaque moteur enrichi XQuery avec des fonctionnalités propres
    - Mais le langage est suffisamment bien spécifié pour ne pas nécessiter trop d'extensions

XQuery

- **Langage définit par le W3C**
  - https://www.w3.org/XML/Query/
- **Largement adopté, dans toutes les communautés**
  - Toutes les NXD supportent XQuery aujourd'hui
  - Existence de processeurs indépendants
    - Implémentations sans bases de données
  - Implémentations en différents langages
    - Java
    - C++ / .Net
    - ERLang
  - Même les acteurs historiques convergent vers XQuery
- **Actuellement en version 3.1**
  - Depuis mars 2017
  - https://www.w3.org/TR/xquery-31/

XQuery est ...

- **Une extension de XPath**
  - Il reprend tous les concepts, toutes les librairies de fonctions, toute la syntaxe
  - Si vous connaissez XPath, c'est un genre de XPath++
- **Un langage pour requêter**
  - Extraire de l'information de un ou plusieurs documents XML
  - "Le SQL du XML"
- **Un langage de transformation**
  - Permet de convertir d'une structure vers une autre

XQuery est aussi ...

- **Un langage de mise à jour**
  - Modification de documents stockés
  - Création de nouveaux documents
- **Un langage de programmation**
  - Programmation fonctionnelle
  - Turing complete
  - Adapté au Web
- **Facile à apprendre**
  - Syntaxe simple

Constructions en XQuery

Créer un élément

- **Les constructions d'éléments se font en écrivant le XML statique**
  - Les expressions devant être évaluées dynamiquement sont entre accolades

```
<planning>
  <event>
    <date>{format-date(current-date(),"[FNn], [D1o] [MNn] [Y]")}</date>
  </event>
</planning>
```

Noms d'éléments dynamiques

- **Parfois, il faut calculer le nom des éléments**
  - On utilise alors la construction **`element {name-expr} { content }`**
  - Aussi valable pour les attributs

```
element {
  if((year-from-date(current-date()) mod 4) eq 0)
  then 'bisextile'
  else 'normal'
}
{current-date()}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<bisextile>2020-02-02+01:00</bisextile>
```

Evaluations imbriquées

- **L'imbrication est sans limite**
  - On peut insérer des sous-éléments dans des éléments calculés

```
<month name="{format-date(current-date(),'[MNn]')}">
       { <week number="{format-date(current-date(),'[W1]')}"/> }
</month>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<month name="February">
  <week number="5"/>
</month>
```

Namespaces dans les éléments créés

- **L'utilisation de namespace dans les éléments est implicite**
  - Il suffit d'écrire son élément avec l'espace de nommage, comme en XML

```
<ox:formation xmlns:ox="com:oxiane:formation">
  <ox:code>AE-DBX</ox:code>
  <ox:titre>Bases natives XML</ox:titre>
</ox:formation>
```

Namespaces dans les expressions XPath

- **Dans les expressions XPath, déclarer les espaces de nommage**
  - Il faut alors ajouter un prologue à l'expression XQuery
  - Chaque ligne de prologue se termine toujours par un point-virgule
  - **`declare namespace <prefix> = <namespace-uri> ;`**

```
declare namespace ox = "com:oxiane:formation";
<ox:formations>
  {
doc('catalogue.xml')/ox:content/ox:formation
}
</ox:formations>
```

Requêtes construites

Mettons des fleurs dans nos requêtes

- **La forme générale d'une requête est l'instruction `FLOWR`**
  - **`for $i in <expr>`** permet de définir l'ensemble requêté
    - **`$i`** est une variable qui contient l'item courant
    - Le nom de variable commence toujours par un **`$`** : **`$auteur`**
    - **`expr`** est une expression permettant d'aller chercher des items
    - Pour chacun des items trouvés, une boucle est constituée et l'item courant est stocké dans **`$i`**
  - **`let $var1 := <expr>`** permet de définir une variable
    - Comme les variables XSL, cette variable ne peut pas être modifiée
    - Dans l'expression définissant la variable, on peut utiliser les variables définies avant.
    - Ces variables définies dans le **`for`** sont évaluées à chaque itération

Mettons des fleurs dans nos requêtes

- **La forme générale d'une requête est l'instruction `FLOWR`**
  - **`order by <expr1> ascending, <expr2> descending`** définit le tri
    - Les items trouvés par le **`for`** seront triés dans l'ordre précisé
    - On peut enchainer plusieurs clauses de tri, séparées par des virgules.
      Si une égalité survient à la première clause, la seconde est utilisée, et ainsi de suite.
  - **`where <expr>`** permet de filtrer les items itérés
    - En général l'expression XPath utilise **`$i`** et les variables déclarées avec **`let`**
  - **`return <expr>`** permet de définir l'entité à renvoyer.
- **Le FLOWR renvoie une séquence**
  - Il est donc possible de renvoyer une séquence de documents

Jointures compactes
Inner Join

- **Il est possible de réaliser des jointures**
  - Combiner ensemble plusieurs documents
- **On définit plusieurs sources dans le même `for`**
  - Le **`for`** itérera sur chaque paire des sources

```
declare namespace ox = "com:oxiane:formations";
declare namespace cl = "com:oxiane:client";
declare namespace html = "http://www.w3.org/1999/xhtml";
for
  $cli in doc('clients.xml')/cl:clients/cl:client,
  $fact in doc('factures.xml')/ox:documents/ox:facture[@clientId eq $cli/@id]
let $dateF := $fact/ox:date
order by $dateF descending
return
  <html:p>
{$cli/cl:name/text()}/{format-date($dateF, "[D]-[M]-[Y]")}: {$fact/ox:mtHt/text()}
  </html:p>
```

```
<p>EDF / 21-10-2020: 3200</p>
<p>Ministère des finances / 31-03-2020: 5760</p>
```

Jointures compactes
Outer Join

- **XQuery introduit la possibilité d'un manquant**
  - La précédente expression ne listait pas un client sans facture
- **`allowing empty` permet d'autoriser un `null` dans la jointure**
  - Attention: pas de virgule entre les deux **`for`**

```
declare namespace ox = "com:oxiane:formations";
declare namespace cl = "com:oxiane:client";
declare namespace html = "http://www.w3.org/1999/xhtml";
for $cli in doc('clients.xml')/cl:clients/cl:client
for $fact allowing empty
   in doc('factures.xml')/ox:documents/ox:facture[@clientId eq $cli/@id]
let $dateF := $facture/ox:date
order by $dateF descending
return
<html:p>
{$cli/cl:name/text()}/{format-date($dateF, "[D]-[M]-[Y]")}: {$fact/ox:mtHt/text()}
</html:p>
```

```
<p>EDF / 21-10-2020: 3200</p>
<p>Ministère des finances / 31-03-2020: 5760</p>
<p>Faculté de médecine / :</p>
```

Jointures avec FLOWR

- **Imbriquer les FLOWR**
  - Permet aussi de combiner plusieurs sources
- **C'est parfois plus lisible**
  - Et cela permet de limiter la jointure à l'endroit où elle est nécessaire

```
declare namespace ox = "com:oxiane:formations";
declare namespace cl = "com:oxiane:client";
declare namespace html = "http://www.w3.org/1999/xhtml";
for $cli in doc('clients.xml')/cl:clients/cl:client
return (<html:h2>{$client/cl:name/text()}</html:h2>,
<html:ul>{
  for $fact in doc('factures.xml')/ox:documents/ox:facture[@clientId eq $cli/@id]
  let $dateF := $facture/ox:date
  order by $dateF descending
  return <html:li>{
    format-date($dateF, "[D]-[M]-[Y]")}: {$facture/ox:mtHt/text()}
    </html:li>
}</html:ul>)
```

```
<h2>EDF</h2>
<ul><li>21-10-2020: 3200</li></ul>
<h2>Ministère des finances</h2>
<ul><li>31-03-2020: 5760</li></ul>
<h2>Faculté de médecine</h2>
<ul/>
```

Jointures multiples

- **Le mécanisme est extensible à l'infini**
  - En sélectionnant plusieurs sources dans le même **`for`**
  - En imbriquant les **`FLOWR`**
- **Il n'y a pas de limite**
  - Si ce n'est la lisibilité et la maintenabilité de la requête

Grouper

- **L'instruction `group by` permet de grouper les résultats**
  - Attention : **`group by`** ne conserve pas l'ordre du document.
  - Il faut souvent le combiner avec **`order by`**

```
for $formation in doc('formations.xml')/formations/formation
let $cat := $formation/categorie/normalize-space(text())
group by $cat
order by $cat
return (
  <h2>{$cat}</h2>,
  for $f in $formation
  return <p>{$f/titre/text()}</p>
)
```

Grouper
Changement de la variable

- **Dans l'exemple précédent, `$formation` représente deux choses :**
  - Le **`let`** définit une variable **`cat`** en utilisant **`$formation`**
    - **`$formation`** est ici un **`element(formation)`**
  - Dans le **`return`**, **`$formation`** est une séquence de **`element(formation)`**
    - Cette séquence regroupe tous les **`formation`** dont la catégorie est identique
    - On est obligé d'itérer sur **`$formation`** pour atteindre chaque **`formation`**
- **Sémantiquement, on se demande si la variable est plurielle !**

Grouper
Syntaxe générale

- **Il est souvent préférable d'utiliser la syntaxe générale**
  - **`group by $var as <type> := <expr>`**

```
for $formation in doc('formations.xml')/formations/formation
group by $cat as xs:string := $formation/categorie/normalize-space(text())
order by $cat
return (
  <h2>{$cat}</h2>,
  for $f in $formation
  return <p>{$f/titre/text()}</p>
)
```

Fenêtrage

Clause window

- **La clause `window` permet de grouper des items adjacents**
  - De façon plus sophistiquée que **`group by`**
- **L'ordre du document est conservé**
  - Contrairement à **`group by`**
- **Des groupes d'items adjacents sont constitués**
  - Commençant ou finissant par une condition
- **`tumbling window` donne des fenêtres adjacentes**
- **`sliding window` donne des fenêtres chevauchantes**

Variables de groupage

- **Les conditions de groupage : `start ... when` et `end ... when`**
  - **`for tumbling window $w in (1, 4, 3, 12, 5, 13, 8) ...`**
- **Plusieurs variables de fenêtre sont accessibles**
  - Le premier item lors de l'évaluation de la fenêtre : **`start $s`**
  - La position du premier item : **`start $s at $s-pos`**
  - L'item qui précède **`$s`** (hors fenêtre) : **`previous $s-prev`**
  - L'item qui suit **`$s`** dans la fenêtre : **`next $s-next`**

```
xquery version "3.0";
for tumbling window $w in (1, 4, 3, 12, 5, 13, 8)
start $s previous $s-prev when (($s mod 2 = 0) or empty($s-prev))
return <window>{$w}</window>
```

```
<window>1</window>
<window>4 3</window>
<window>12 5 13</window>
<window>8</window>
```

Variables de groupage

- **Les conditions de groupage : `start ... when` et `end ... when`**
  - **`for tumbling window $w in (1, 4, 3, 12, 5, 13, 8) ...`**
- **Plusieurs variables de fenêtre sont accessibles**
  - Le dernier item lors de l'évaluation de la fenêtre : **`end $e`**
  - La position du dernier item de la fenêtre : **`end $e at $e-pos`**
  - L'item qui précède **`$e`** dans la fenêtre : **`previous $e-prev`**
  - L'item qui suit **`$e`** (en dehors de la fenêtre) : **`next $e-next`**

```
xquery version "3.0";
for tumbling window $w in (1, 4, 3, 12, 5, 13, 8)
start when true()
end $e next $e-next when (($e mod 2 = 0) or empty($e-next))
return <window>{$w}</window>
```

```
<window>1 4</window>
<window>3 12</window>
<window>5 13 8</window>
```

Usage de start et end

- **`start` est toujours obligatoire**
  - Il faut définir quand une fenêtre commence
- **`end` est optionnel**
  - Si il est omis, la fenêtre se termine lorsque la suivante commence

```
let $data := <properties>
      <propname>prenom</propname>
      <propvalue>Christophe</propvalue>
      <desc>Le prénom de la personne</desc>
      <propname>nom</propname>
      <propvalue>Marchand</propvalue>
      <desc>Le nom de la personne</desc>
    </properties>
for tumbling window $w in $data/*
start $s when local-name($s) = 'propname'
end $e when local-name($e) = 'propvalue'
return element {string($s)} {string($e)}
```

```
<prenom>Christophe</prenom>
<nom>Marchand</nom
```

Sliding window

- **Le `sliding window` permet de faire glisser les fenêtres**
  - Utile pour des évènements temporels se recouvrant
  - Ou pour des moyennes glissantes
- **Chaque item peut apparaître dans plusieurs fenêtres**

```
xquery version "3.0";
for sliding window $w in (1, 4, 3, 12, 5, 13, 8)
start at $s-pos when true()
end at $e-pos when $e-pos - $s-pos = 2 return <window>{$w}</window>
```

```
<window>1 4 3</window>
<window>4 3 12</window>
<window>3 12 5</window>
<window>12 5 13</window>
<window>5 13 8</window>
<window>13 8</window>
<window>8</window>
```

only end

- **Dans l'exemple précédent, la fenêtre se termine**
  - Lorsque la taille de la fenêtre est de 3
  - Ou lorsqu'il n'y a plus d'items pour finir la fenêtre
- **Cela explique les deux dernières fenêtres à 2 et 1 item**
- **`only end` force la fin de la fenêtre sur la condition seulement**
  - Et pas sur la fin des items

```
xquery version "3.0";
for sliding window $w in (1, 4, 3, 12, 5, 13, 8)
start at $s-pos when true()
only end at $e-pos when $e-pos - $s-pos = 2 return <window>{$w}</window>
```

```
<window>1 4 3</window>
<window>4 3 12</window>
<window>3 12 5</window>
<window>12 5 13</window>
<window>5 13 8</window>
```

Fenêtres : dangers

- **Les fenêtres XQuery sont différentes du `xsl:for-each-group`**
  - Certains items de la séquence peuvent ne pas être présents dans une fenêtre
    - Alors qu'avec **`xsl:for-each-group`** tous les items sont présents dans les groupes
  - Certains items peuvent apparaître dans plusieurs fenêtres
    - Cas des sliding window
- **Attention de ne pas confondre les usages**
  - Entre les fenêtres XQuery et les groupes XSL

Fonctions

Appels de fonctions

- **XQuery s'appuie sur XPath**
  - Toutes les fonctions de XPath sont donc disponibles, et appelables
- **XQuery fournit des fonctions supplémentaires**
- **On peut aussi définir ses propres fonctions**

- **Une fonction s'appelle par son nom et ses paramètres**
  - Le nom d'une fonction est un **QName**
  - **<function-name> (<expr>, ...)**

```
substring($name, 1, 3)
```

Appels de fonctions

- **XQuery définit un namespace par défaut pour les fonctions**
  - Celui des fonctions XPath : **`http://www.w3.org/2005/xpath-functions`**
- **Il n'est pas utile d'utiliser de préfix pour appeler ces fonctions**
  - Sauf si on redéfinit l'espace de nommage par défaut des fonctions
  - Pas recommandé
- **XPath 3.0 et 3.1 introduisent de nouvelles fonctions**
  - Pour les mathématiques, les maps, les arrays, les erreurs et la sérialisation
  - Il faut alors définir ces espaces de nommage pour appeler ces fonctions

```
declare namespace math = "http://www.w3.org/2005/xpath-functions/math";
declare namespace map = "http://www.w3.org/2005/xpath-functions/map";
declare namespace array = "http://www.w3.org/2005/xpath-functions/array";
declare namespace err = "http://www.w3.org/2005/xqt-errors";
declare namespace output = "http://www.w3.org/2010/xslt-xquery-serialization";
```

Définition de fonction

- **XQuery permet de définir ses propres fonctions**
  - Une fonction définie par l'utilisateur doit être dans un namespace
    - Les espaces de nommages XPath, XQuery, Xslt et autres sont réservés
    - L'espace de nommage par défaut `""` est interdit
- **L'appel d'une fonction personnelle se fait avec un préfixe**

```
declare namespace local = "com:oxiane:formation:xquery:functions:local";
declare function local:value2() as xs:integer {
2
};
<value>{local:value2()}</value>
```

Définition des paramètres et typage

- **Les paramètres se définissent après le nom de la fonction**
  - Entre parenthèses
  - Séparés par des virgules
- **Il faut typer les paramètres**
  - Comme il faut le faire quand on définit des fonctions en XSL
- **Une fonction renvoie une valeur**
  - Il faut donc typer une fonction pour définir son retour
- **Une déclaration de fonction se termine toujours par un `;`**

```
declare function local:calculate(
$value1 as xs:integer,
$value2 as xs:integer,
$value3 as xs:integer) as xs:double {
($value1 + $value2) div $value3
};
local:calculate(1,2,3)
```

Corps de la fonction

- **Le corps de la fonction est entouré d'accolades**
- **Les accolades contiennent une ou plusieurs expressions**
  - Tout type d'expression XQuery, FLOWR, XPath
- **Une fonction renvoie une valeur**
  - La valeur renvoyée est le résultat de l'évaluation des expressions
  - Converti dans le type de retour de la fonction
- **Le corps de la fonction peut se terminer par `return`**
  - Ce n'est pas obligatoire

```
declare function local:calculate(
$value1 as xs:integer,
$value2 as xs:integer,
$value3 as xs:integer) as xs:double {
let $operator as xs:integer := $value1 + $value2
return $operator div $value3
};
local:calculate(1,2,3)
```

Pourquoi écrire des fonctions ?

- **Réutilisation**
  - Factoriser du code dans une fonction permet d'éviter de dupliquer
  - Plusieurs appels différents utilisent le même code
- **Clarté**
  - Certaines opérations sont complexes
  - Les isoler dans une fonction permet de séparer ce code du reste
- **Récursivité**
  - Certains algorithmes nécessitent la récursivité.
  - Seul l'usage de fonction permet la récursivité
- **Tests unitaires**
  - A l'aide de XSpec, il est aisé de tester une fonction
    - Pour la mettre au point, et pour vérifier sa stabilité dans le temps
- **Faciliter le refactoring**
  - Il est plus facile de réécrire une fonction que de chercher dans tout le code pour trouver les endroits à modifier

Modules de fonctions

- **Afin de réutiliser des fonctions, on les définit dans des modules**
  - Un module est un fichier qui ne contient que des fonctions
    - Eventuellement des variables globales
  - Toutes dans le même espace de nommage
  - Son extension est conventionnellement **`.xqm`**
    - Alors qu'une requête est dans un fichier **`.xq`**
  - Il a un prologue particulier, qui définit le module et son espace de nommage
  - **`declare module namespace ox="com:oxiane:xq:module";`**
- **On importe alors un module d'un namespace depuis un fichier**
  - **`import module namespace ox="com:oxiane:calc" at "calcs.xqm";`**
    `ox:calculate(1,2,3)`

Signatures et visibilité

- **Une fonction doit avoir une signature unique**
  - Sa signature est définie par
    - le nom qualifié (namespace + nom)
    - L'arité : le nombre d'arguments
  - Attention, dans l'arité, le type des arguments n'est pas pris en compte
    - Ces deux définitions ont la même signature
    - `declare function local:lgth($s as xs:string) {$s=>string-length()};`
    - `declare function local:lgth($i as xs:int) {string-length(string($i))};`
- **Dans un module, on peut définir la visibilité des fonctions**
  - `declare %private function local:val() {math:pi()};`
  - `declare %public function local:get-pi-value() {local:val()};`
- **Une fonction `%private` ne sera pas accessible en dehors du module**
  - Une fonction qui ne définit pas de visibilité est publique

XQuery et Bases XML

Collections

- **Les bases de données XML sont découpées en `collections`**
- **Une collection est un regroupement logique de documents**
  - Logique métier plus que technique
- **La fonction `collection(...)` permet d'accéder au contenu**
  - La syntaxe du paramètre est **`implementation defined`**
  - Cela signifie que chaque processeur XQuery a sa propre syntaxe
- **Les moteurs de bases XML utilisent en général un nom de collection**
  - **`collection("factures")/*`** ramène tous les documents dans la collection facture
- **Les processeurs indépendants utilisent des filtres de fichiers**
  - **`collection("file:///a/b/c?select=*.xml&recurse=true")`** sous Saxon
    - Renvoie tous les fichiers dont l'extension est xml dans le répertoire /a/b/c de façon récursive

Collections - Usage

- **`collection(...)` renvoie une séquence**
  - En général de documents, donc de noeuds
- **Certaines instructions nécessitent de trier cette séquence**
  - **`collection("factures")/facture[client/@id eq '123']`**
  - Ainsi, toute la séquence est chargée en mémoire, ce qui peut être couteux
- **Préférer en général des instructions FLOWR**
  - Tester en fonction des moteurs, les résultats sont souvent différents

```
for $fact in collection("factures")
where $fact/facture/client/@id = '123'
return ...
```

Insertion de documents

- **Une base de données contient des documents**
  - Il faut donc un moyen pour importer des documents
- **En général, un document existe en dehors de la base**
  - Sous la forme de fichier, de ressource réseau, etc..
- **Il faut importer le document dans la base**
- **Il n'existe pas de fonction XQuery pour faire cela**
  - Chaque base de donnée propose ses fonctions propres
  - MarkLogic
    - `xdmp:document-insert($uri, $root, $options)`
  - BaseX
    - `CREATE DB db/path/to/resources`
  - eXist
    - `xmldb:store($collection, $fileName, $node)`

Besoins

- **Assez rapidement, il a fallu mettre à jour les données stockées**
- **Deux types d'usage**
  - Remplacement de documents complets
  - Modifications de noeuds à l'intérieur du document
- **Remplacement de documents**
  - La base de données sert de "source" pour une chaîne de publication
  - A intervalles réguliers, des documents sont ajoutés, supprimés ou modifiés
  - Il faut donc remplacer ces documents dans la base
- **Modifications de noeuds du documents**
  - La base de données sert à stocker des informations vivantes
  - Très régulièrement modifiées
  - Les mises à jour doivent être ciblées et rapides

Mises à jour dans les bases de données

- **L'intérêt d'une base de données est la possiblité de modifier les données**
  - XQuery ne propose pas de méthode pour mettre à jour les données
- **Plusieurs acteurs ont proposé des solutions différentes**
  - W3C a proposé XQuery Update Facilities
    - Porté par MarkLogic et Oracle
    - Implémenté par aucun des deux
    - Le langage de référence, même si très peu utilisé et implémenté
  - eXist-db et MarkLogic ont chacun développé leur langage de mise à jour
    - Les fonctions sont souvent semblables à XQUF, mais les concepts un peu différents

XQuery Update Facility

XQuery Update Facility

- **En mars 2011, une nouvelle spécification W3C est publiée : XQUF**
  - XQuery Update Facility
  - Elle propose des extensions à XQuery pour modifier le XDM
- **Huit mois plus tard, la version 3.0 de ce langage est publiée**
  - Plus un alignement des numéros de version avec XQuery 3.0 qu'une vraie révolution
- **Le problème est que la spécification est arrivée très tard**
  - Les moteurs existants avaient déjà leur propres langages de modification
  - BaseX propose une implémentation complète de XQUF

XQUF: Instructions de modification

- **Des instructions simples permettent de modifier le XDM**
  - `insert node ( attribute { 'id' } { 'zx45' } ) into /n`
    - Insère un noeud dans un autre. Ici un attribut dans le résultat d'une expression XQuery
  - `delete node //n`
    - Supprime tous les noeuds renvoyés par l'expression
  - `replace node /n with <a/>`
    - Remplace un noeud par un autre
  - `replace value of /n with 'new value'`
    - Remplace la valeur d'un noeud par autre chose
  - `for $n in //original return rename node $n as 'renamed'`
    - Renomme chaque noeud

XQUF insert

- **Permet d'insérer un ou plusieurs noeuds dans un contenu**
  - **`insert nodes (<a/>, <b/>) as first into /n`**
    - Insère des noeuds à la première - ou dernière - position dans un noeud
  - **`insert node (<a/>) into /n before /n/b`**
    - Insère un noeud dans un noeud avant - ou après - un autre

XQUF delete

- **Permet de supprimer un ou plusieurs noeuds**
  - **`delete node /n/p[@id eq 'sdf']`**
    - Supprime le ou les noeuds désigné par l'expression.
  - **`delete nodes /n/p[currentDate() - @date > xs:dayTimeDuration("P365D")]`**
    - Supprime le ou les noeuds désigné par l'expression.
  - On peut utiliser indifférement **`node`** ou **`nodes`**

XQUF replace

- **Remplace un noeud ou son contenu par autre chose**
  - Il n'est possible de remplacer qu'un seul noeud
  - Le noeud à remplacer doit avoir un parent
  - On ne peut pas remplacer un attribut par un non-attribut, et réciproquement
  - **`replace node /n/p with /a/b`**
    - L'élément **`p`** est remplacé par une copie de l'élément **`b`**
  - **`replace value of node /n/p with /a/b/text()`**
    - Le contenu de l'élément **`p`** est remplacé par le texte de **`/a/b`**

XQUF rename

- **Renomme un élément ou un attribut**
  - `rename node /n/p as "t"`
- **Utilisation de namespace**
  - `rename node /n/p as QName("http://www.w3.org/1999/xhtml", "html:p")`
    - `/n/p` est changé de namespace, et on lui spécifie le préfix `html`

XQUF copy modify return

- **Toutes les expressions de modification ne renvoient rien**
  - Empty sequence
- **Il est possible d'écrire une requête qui renvoie une copie modifiée**
  - L'item original n'est pas modifié, mais une copie en est faite, elle est modifiée, puis retournée
  - ```
    copy $ret := /doc("foe.xml")//original[@id eq 'x1']
    modify rename node $ret as "modified"
    return $ret
    ```
    ```

XQUF Pending Update List

- **Le concept fondamental de XQUF est la Pending Update List**
- **Une instruction est soit une requête, soit une collection de modifications**
  - On ne peut pas mélanger
- **Les instructions de modification ne sont pas exécutées tout de suite**
  - Elles sont collectées et assemblées en une liste de modifications en attente
  - A la fin de l'évaluation de l'instruction, toutes les modifications sont exécutés en même temps
- **Si deux instructions de modifications se suivent**
  - La seconde instruction ne peut pas modifier des items créés par la première

```
copy $doc := <doc><a/></doc>
modify (insert node <b/> into $doc,
for $n in $doc/child::node()
return rename node $n as 'justRenamed')
return $doc
```

```
<doc>
  <justRenamed/>
  <b/>
</doc>
```

XQUF - Fonctions de modification

- **Il est possible d'écrire des fonctions qui modifient du contenu**
  - La syntaxe est la même que la déclaration d'une fonction XQuery
- **Une fonction ne peut pas renvoyer des données, et modifier des données**
  - Soit sélection
  - Soit mise à jour
- **Les fonctions de mise à jour doivent être annotées `%updating`**

```
declare %updating function local:addVAT($price as element(), $tx as xs:integer) {
  insert node <pTTC>{(data($price) * (100+$tx)/100)}</pTTC> after $price
};

local:addVAT(<price>100</price>,20)
```

XQUF Conclusion

- **XQuery Update Facility est un langage complet**
  - Il permet la mise à jour des données dans les bases qui l'implémentent
- **Très peu d'implémentations disponibles à ce jour**
  - BaseX
  - Monet DB
  - DB XML (Berkley / Oracle)
  - XMLmind

eXist-db Update Extension

Présentation

- **eXist-db utilise sa propre syntaxe de modification**
  - Et ne semble pas prendre la direction de XQuery Update Facility
- **eXist-db met directement à jour les données**
  - Au fur et à mesure de l'évaluation de l'expression
- **C'est contraire au Pending Update List de XQUF**
  - Et eXist-db souhaite conserver ce fonctionnement
- **Plus intuitif que PUL, mais plus dangereux**
  - Il est possible de supprimer les éléments en cours de parcours
- **A ce jour, non transactionnel !**
  - Fusion-DB propose une approche transactionnelle de ce langage
  - Mais encore en bêta, donc non utilisable en prod

Limitations

- **Mise à jour de données persistées**
  - XQUE a été créé pour modifier les documents persistés dans la base
    - Et pas les fragments temporaires créés au sein d'une requête
  - Pas d'équivalent du **`copy modify return`**
- **Ce code n'a aucun effet, et renvoie une empty-sequence**

```
let $node:= <root><a/></root>
return
  update insert <b/> into $node/a
```

Update

- **Toutes les instructions de mise à jour commencent par `update`**
  - Vient ensuite la nature de la modification à apporter
    - `insert`
    - `delete`
    - `replace`
    - `value`
    - `rename`
- **`update` renvoie toujours une empty-sequence**

update insert

- **`update insert expr (into | following | preceding) exprSingle`**
  - Permet d'insérer **`expr`**
  - dans, après, avant **`exprSingle`**
- **`expr` permet de construire une séquence d'items à insérer**
- **`exprSingle` est une expression désignant le nœud...**
  - dans lequel
  - après lequel
  - avant lequel
  - ... insérer
- **Si `exprSingle` contient plusieurs nœuds**
  - Ils sont tous mis à jour

update replace

- **`update replace expr with exprSingle`**
  - Remplace **`expr`** par **`exprSingle`**
- **`expr` doit désigner un nœud unique**
  - un élément
    - **`exprSingle`** doit être un élément unique
  - un attribut
  - un nœud text
    - La valeur sera la concaténation des string-value de **`exprSingle`**
- **`expr` ne peut pas être un élément racine de document**

```
update replace fname[. eq "Christophe"] with <fname>Tophe</fname>
```

update value

- **`update value expr with exprSingle`**
  - Modifie la valeur de chaque item de **`expr`**
  - Par l'ensemble du contenu de **`exprSingle`**
- **Si `expr` est un attribut ou un text-node**
  - La concaténation des string-value de **`exprSingle`** est utilisée

```
update value fname[. eq "Christophe"] with "Tophe"
```

update delete

- **`update delete expr`**
  - Supprime tous les nœuds renvoyés par **`expr`**

```
for $country in //address/country
return
  update delete $country
```

update rename

- **`update rename expr as exprSingle`**
  - Renomme tous les items **`expr`**
  - Par le string-value du premier item de **`exprSingle`**
- **`expr` doit renvoyer des éléments et/ou des attributs**
- **`expr` ne peut pas être un élément racine de document**

```
for $city in //address[zipCode eq "92100"]/city
return
  update rename $city as 'locale'
```

Attention aux update dans les FLOWR

- **Lorsqu'on itère au sein d'un FLOWR**
  - Si on supprime plusieurs nœuds de l'itération
- **On risque de corrompre la base**
- **Dans le cas suivant, `update delete //address`**
  - Supprime des nœuds qui ne sont pas encore évalués par **`for $address in //address`**
- **Préférer toujours des modifications unitaires dans les FLOWR**

```
for $address in //address
return update delete //address
```

```
for $address in //address
return update delete $address
```

Bases XML et Web

Les besoins du Web

- **Dans l'immense majorité des cas, il faut publier des documents sur le Web**
  - Cela signifie travailler sur le contenu du document
  - Répondre aux requêtes utilisateurs
    - Quel document afficher
    - Quelle recherche effectuer
- **Bref, écrire une appli Web qui fournit du document**

RestXQ
Quésaco ?

- **RestXQ est un standard, permettant d'écrire des services Rest en XQuery**
  - Proposé par Adam Retter, jamais présenté au W3C WG XQuery, maintenu par EXQuery
  - Largement adopté par BaseX et eXist-DB, partiellement par MarkLogic
  - Standard de fait
- **Très facile à mettre en place**
  - Il suffit de déposer des modules XQuery sur le serveur de base de donnée
  - Un minimum de configuration du serveur
    - Déclaration des "applications" web

RestXQ Annotations

- **RestXQ fonctionne avec des annotations**
  - Assez semblable à JAX-RS dans l'esprit et dans la forme

```
declare
  %rest:path("/livres")
  %rest:GET
  %rest:form-param("auteurId","{$auteurId}", "")
function li:getLivres($auteurId as xs:string) as element() {
  for $li:=collection('/db/data/shared')//li:livre[//li:auteur/@refid=$auteurId]
  return $li/li:titre
};
```

Autres solutions Web

- **Les bases XML sont utilisées principalement en publication**
  - Comme back-end d'une application Web qui affiche les documents
  - Ou comme source de données d'une chaîne de publication papier
- **Il est alors assez naturel d'heberger les services d'exposition**
  - Soit les services REST / Web Services
  - Soit directement les applications Web
- **Chaque éditeur propose des solutions propres**
  - Allant du plus standard
  - Au plus spécifique mais plus complet

Conclusion

De vrais standards

- **Les langages de manipulation de données sont normalisés**
  - XQuery
  - XQuery Update Facility
- **Parfois des solutions proches du standard**
  - XQUE - eXist-DB & Fusion-DB
  - Update MarkLogic
- **Relative portabilité des développements**

Solutions commerciales péreines

- **Les solutions commerciales**
  - Robustes
  - Fiables
  - Supportent la scalabilité
- **Les acteurs sont présents depuis longtemps**
- **Ils proposent des solutions pratiques**
  - Projection de données
    - Par exemple dans des solutions RDF

Solutions OSS suffisantes

- **Les produits Open-Source sont souvent suffisants**
  - Pour service de source de données
  - Pour alimenter du Web
- **Ils ont cependant des limitations**
  - Volumétrie
    - BaseX & eXist-DB
  - Transactions
    - eXist-DB
  - Mises à jour bloquantes
    - BaseX

Marché très vivant

- **De nouvelles solutions font leur apparition**
  - Fusion-DB, entièrement compatible avec eXist-DB
    - Mais ajoute le support de grosses volumétries et des transactions
    - Utilisation du moteur de persistance de FaceBook, RocksDB
    - OSS
  - XmlMind
    - Apporte dans sa dernière version, le support XQuery et XQUF
    - OSS et commercial, en fonction des fonctionnalités
  - XQErl, nouvelle base écrite en ERLang
    - Et donc support des très grosses volumétries et des hautes performances

Incontournables

- **Dans une organisation de gestion documentaire**
  - Les bases XML sont incontournables
- **Comme l'étaient les bases relationnelles il y a 20 ans...**

- **Merci !**

Bibliographie

Bibliographie

- XML and databases http://www.rpbourret.com/xml/XMLAndDatabases.htm
- Recommandation XQuery 3.1 https://www.w3.org/TR/xquery-31/
- Note XQuery Update Facility
  https://www.w3.org/TR/2017/NOTE-xquery-update-30-20170124
- Adam Retter's XQuery and XML Databases
  http://static.adamretter.org.uk/xmlss-19.pdf
- Priscilla Walmsley: XQuery 2nd Edition - O'Reilly - 2016 - 978-1-491-91510-3
- BaseX Documentation : http://docs.basex.org/wiki
- eXist-db Documentation : http://exist-db.org/exist/apps/doc/