مخبر هندسة الأنظمة المعقدة
LABORATOIRE D'INGÉNIERIE DES SYSTÈMES COMPLEXES

جامعة باجي مختار-عنابة
Université Badji Mokhtar-Annaba

Administration des bases de données avancées

Master I: Ingénierie des Logiciels Complexes (ILC)

Dr Kamilia MENGHOUR
Laboratoire d'Ingénierie des Systèmes Complexes
Université Badji Mokhtar-Annaba
K_menghour@yahoo.fr

Année Universitaire : 2024- 2025

Chapitre 3

Langages de création et d'interrogation des bases de données

Plan du cours

Chapitre 3 : *Langages de création et d'interrogation des bases de données (20%)*

- LDD,
- SQL,
- PHP,
- XML

Langages de requêtes XML

Langages de requêtes XML

- Les langages de requêtes basés sur SQL peuvent être utilisés uniquement avec des BD relationnelles.
- Les langages de requêtes XML, peuvent être utilisés sur n'importe quel document XML,
- Pour les utiliser avec des bases de données relationnelles, les données de la BD doivent être modélisées en XML ce qui permet de formuler des requêtes sur des documents XML virtuels.
- Plusieurs langages de requêtes ont été développés pour l'interrogation des documents XML, citons à titre d'exemple : XML-QL, XPath, XQL, Xquery, Lorel, Quilt, UnQL, XDuce,.
- Dans ce qui suite nous allons présenter brièvement XPath et XQuery

XPath

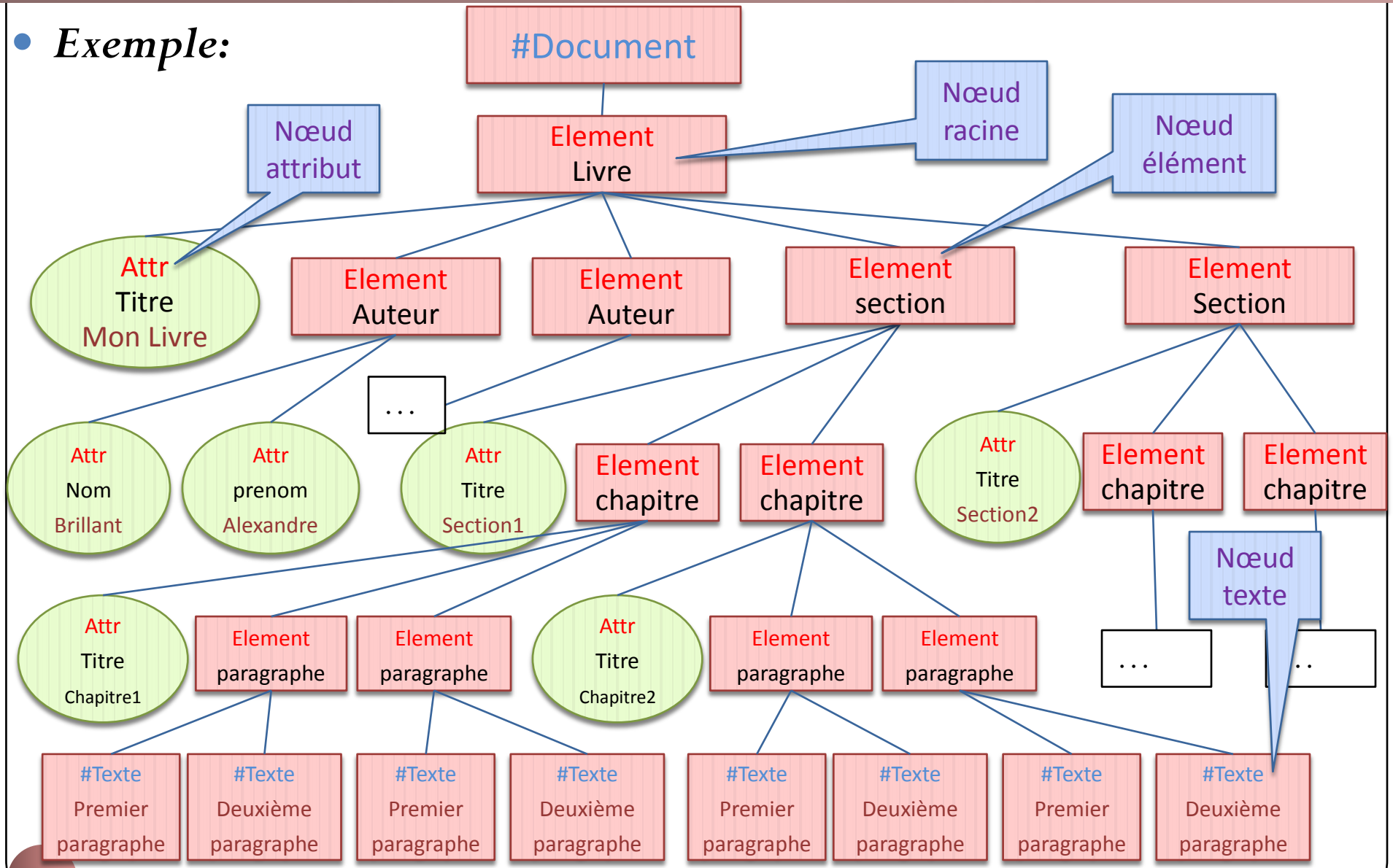
XML Path Language

XML Path Language (XPath)

- ***XPath*** est un langage pour l'interrogation des parties d'un document XML, à l'aide d'expressions de chemin.
- Il utilise une syntaxe qui ressemble à celle utiliser pour adresser des parties d'un système de fichiers , un URL ou un nom complet Unix, mais avec des conditions écrites entre [].
- ***XPath*** est central dans le monde XML, il intervient comme brique de base dans d'autres technologies XML :
  - XML Schémas (expression des contraintes d'unicité et de clefs),
  - les transformations XSLT,
  - XQuery
  - XLink
  - XPointer, etc.

Chemin dans une arbre XML

- ***Exemple:***



Syntaxe générale

- Le premier concept est celui de ***nœud courant*** : c'est l'endroit d'où l'on part.
- En première lecture on peut imaginer qu'il s'agit de la racine du document, mais n'importe quel nœud peut jouer ce rôle.
- À partir de là, on considère trois éléments :
  - ***Un axe :*** la direction dans laquelle on se dirige à partir du nœud courant (vers: le père, les fils, les frères de gauche, etc.).
  - ***Un filtre :*** le type de nœuds qui nous intéresse dans l'axe choisi (des nœuds quelconques, des éléments quelconques ou un élément précis, des commentaires, etc.) ;
  - ***un prédicat optionnel :*** des conditions supplémentaires pour sélectionner des nœuds parmi ceux retenus par le filtre dans l'axe.

Syntaxe générale

- ***Une étape*** est constitue d'un axe, filtre et prédicat.

  axe::filtre[prédicat]

- L'enchaînement de plusieurs étapes constitue une chemin XPath :

  axe1::filtre1[prédicat1]/axe2::filtre2[prédicat2]

- Exemple :

  parent::*/child::node()[position()=2]

- Il est possible de faire une disjonction de requêtes ***XPath*** avec le signe | ; on obtient alors l'union des deux ensembles de nœuds correspondants.

  axe1::filtre1[prédicat1]/axe2::filtre2[prédicat2] |
  axe3::filtre3[prédicat3]

- Chaque étape renvoie un ensemble de nœuds.

Chemins XPath

- Un chemin XPath est une suite d'étapes: [/]étape /étape /.../étape
- Deux variantes :
  - **Un chemin absolu :** Le nœud de départ est toujours la racine de l'arbre XML. Une expression Xpath utilisant un chemin absolu est facilement identifiable car elle commence par le caractère "/".

Exemple: récupérer le prénom de la personne décrite dans notre arbre XML:

- Etape1:noeud "**repertoire**".
- Etape2:descendre au nœud enfant "**personne**".
- Etape3:descendre au nœud enfant "**prenom**".

L'***expression Xpath*** correspondante ressemblera alors à ça: /étape1/étape2/étape3

- **Un chemin relatif** : accepte n'importe quel nœud de l'arbre XML comme point de départ. Une expression Xpath utilisant un chemin relatif est facilement identifiable car elle ne commence pas par le caractère "/".

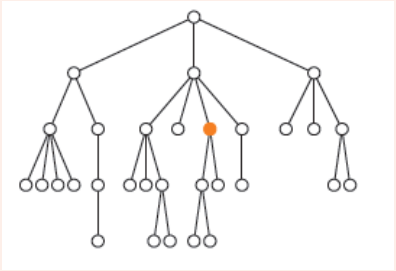
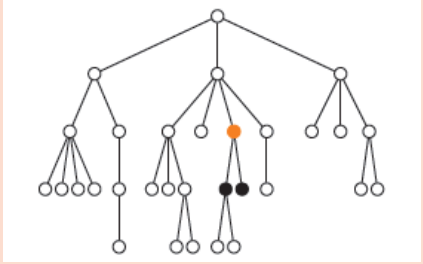
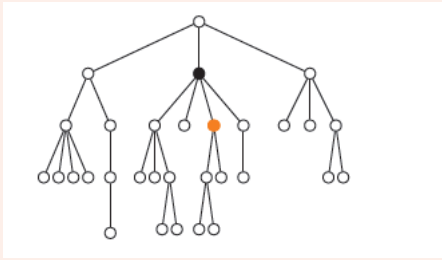
Exemple: récupérer le prénom de la personne. Dans cet exemple, notre point de départ est le nœud décrivant le numéro de téléphone portable de Kadri:

- Etape1:nœud "telephone" dont l'attribut est "portable".
- Etape2:remonter au nœud parent "telephones".
- Etape3:aller nœud frère"prenom".

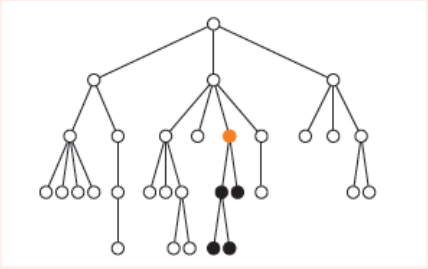
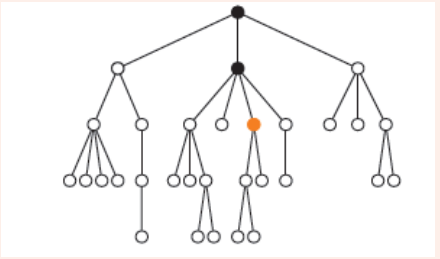
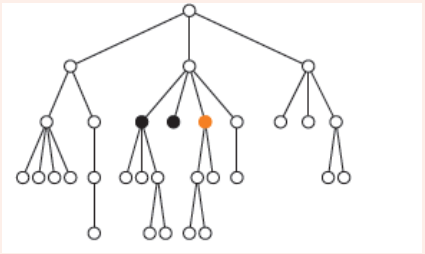
L'***expression Xpath*** correspondante ressemblera alors à ça: étape1/étape2/étape3

Chemins XPath: Les axes

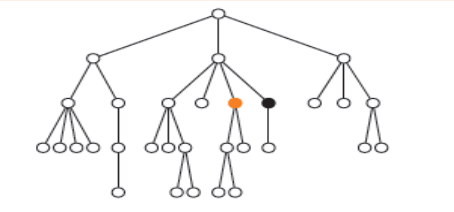
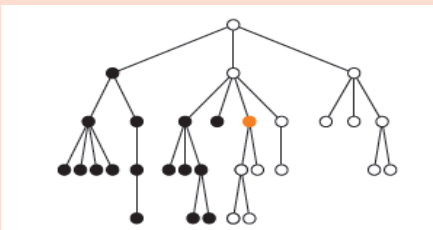
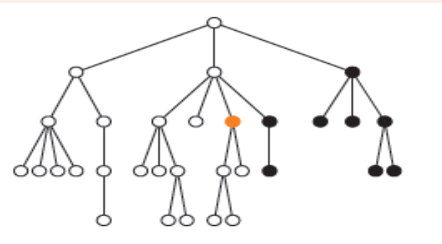
- Chacun des ***axes*** donne une relation qui relie les *nœuds sélectionnés* au *nœud courant*. Les axes qu'on peut les utiliser dans les expressions XPath sont les suivants:

Self	Oriente la recherche vers le nœud courant	
Child	Oriente la recherche vers les enfants du nœud courant	
Parent	Oriente la recherche vers le parant du nœud courant	

Chemins XPath: Les axes

descendant	Oriente la recherche vers les descendants du nœud courant	
descendant-or-self	Oriente la recherche vers les le nœud courant et ses descendants	
ancestor	Oriente la recherche vers les ancêtres du nœud courant	
ancestor-or-self	Oriente la recherche vers le nœud courant et ses ancêtres	
preceding-sibling	Oriente la recherche vers les frères précédents du nœud courant	

Chemins XPath: Les axes

following-sibling	Oriente la recherche vers les frères suivants du nœud courant	
preceding	Oriente la recherche vers les nœuds précédents le nœud courant	
following	Oriente la recherche vers les nœuds suivants le nœud courant	
Attribute	Oriente la recherche vers les attributs du nœud courant	
Namespace	Oriente la recherche vers les nœuds de type espace de nom du nœud courant,	

Chemins XPath: Filtres

- Deux façons de filtrer les nœuds d'un axe:
  - Par leur nom: Pour les nœuds qui ont un nom (Element, Attribute, ProcessingInstruction)
  - Par leur type: text(), comment(), processing-instruction(), *, node().

Node()	Oriente la recherche vers tous les types de nœuds (éléments, Attributs, commentaires, …)
Text()	Oriente la recherche vers les nœuds de type texte
Comment()	Oriente la recherche vers les nœuds de type commentaire.
*	Oriente la recherche vers tous les nœuds
Nom du nœud	Oriente la recherche vers les nœuds portant ce nom
processing-instruction ('cible')	les nœuds instructions, seulement les instructions cible si cet argument est fourni.

Chemins XPath: Prédicats

- Ils prennent la forme de tests que les nœuds sélectionnés devront vérifier. Ces tests peuvent impliquer des fonctions ou de nouveaux chemins XPath.

```
//child::chapitre[position()=2]
child::livre[@titre="Mon livre"]
```

Il est possible de combiner ces tests à l'aide des opérateurs logiques classiques (and, or et not) ou de les enchaîner :

```
//child::chapitre[@titre="Chapitre 1" or @titre="Chapitre 2"]
//child::paragraphe[contains(text(),"Xml") and position()=2]
//child::paragraphe[contains(text(),"Xml")][position()=2]
```

- Les deux dernières requêtes ne sont pas équivalentes :
  - la première renvoie le deuxième fils paragraphe si celui-ci contient le texte Xml ;
  - la seconde sélectionne tous les fils paragraphe qui contiennent le texte Xml et parmi ceux-ci renvoie le deuxième.

Chemins XPath: Les fonctions

- XPath possède de très nombreuses fonctions. Ces fonctions peuvent apparaître dans des prédicats ou être utilisées directement dans un évaluateur d'expressions XPath.
- ***Fonctions sur les éléments :***

Fonction	**Description**	***Exemple***: Xpath
position() :	retourne l'index de l'élément dans son parent	
last()	Permet de sélectionner le dernier nœud d'une liste	**/ROOT/AA/BB[position()=last()]** **<ROOT> <AA> <BB/> </AA>** **<AA> <BB/> <BB/> <BB/> </AA>** **<AA> <BB/> <BB/> </AA>** **</ROOT>**
count(*node -set*)	Permet de compter le nombre de nœud	**<?xml version="1.0" encoding="UTF-8"?>** **<ROOT> <AA/> <BB/> <AA/> <AA/> <BB/>** **</ROOT>** **count(/ROOT/AA) renvoie 2.** **count(/ROOT/BB) renvoie 3.** **count(/ROOT/*) renvoie 5.**
name(*node-set ?*)	Permet de sélectionner le nom du premier nœud passé en argument	**/ROOT/AA/*[name()='BB']** **<ROOT> <AA> <BB/> <CC/> </AA>** **<ROOT>**
namespace-uri(*node-set ?*)	retourne l'URI du namespace du premier nœud du nodeset passé en paramètre.	**//*[namespace-uri()=« URI_de_test »]** **<ROOT xmlns:test=« URI_de_test »>** **<AA> <test:BB/> </AA>** **<test:AA> <BB/> <test:BB/> <BB/> </AA>** **</ROOT>**

Chemins XPath: Les fonctions

- ***Fonctions de chaînes de caractères***

Fonction	**Description**	***Exemple***: Xpath
Concat(*string* , *string* , *string* *)	retourne le résultat de la concaténation des arguments.	**concat(« AB »,« CDE »,« EF ») retourne ABCDEEF**
starts-with (*string* , *string*)	retourne la valeur booléenne true si la première chaîne de caractères passée en argument commence par la chaîne de caractères passée en deuxième argument.	**starts-with(« ABCDE »,« ABC »>) retourne true.** **starts-with(« ABCDE »,« B ») retourne false.**
Contains (*string* , *string*)	retourne true si la première chaîne de caractères passée en argument contient la chaîne de caractères passée en deuxième argument sinon retourne la valeur false.	**contains(« ABCDE », « BC »)** retourne true. **contains(« ABCDE », « Z »)** retourne false.
string-length (*string ?*)	retourne le nombre de caractères de la chaîne.	
substring-before(*string* , *string*), substring-after(*string* , *string*), substring(*string* , *number* , *number ?*), normalize-space(*string ?*)…		

Chemins XPath: Les fonctions

- ***Fonctions numériques:***

Fonction	**Description**	***Exemple***: Xpath
sum(*node-set*)	retourne la somme, pour tous les nœuds de l'ensemble passé en argument,	<ROOT> <AA> 1 </AA> <AA> 2 </AA> </ROOT> sum(/ROOT/AA) retourne 3
floor(*number*)	retourne le plus grand nombre entier inférieur à l'argument du côté de l'infini positif.	floor(4.2) retourne 4 ; floor(-4.2) retourne -5.
ceiling(*number*)	retourne le plus petit (du côté de l'infini négatif) nombre entier qui ne soit pas inférieur à l'argument.	ceiling(4.2) retourne 5 ; ceiling(-4.2) retourne -4.
round(*number*)	retourne le nombre entier le plus proche de l'argument.	round(4.2) retourne 4 ; round(4.6) retourne 5 ; round(4.5) retourne 5 ; round(-4.2) retourne -4 ; round(-4.6) retourne -5 ; round(-4.5) retourne -4.

Chemins XPath: Les fonctions

- ***Fonctions booléennes:***

Fonction	Description
not(*boolean* **)**	retourne l'inverse de la valeur du booléen passé en argument : vrai (true) si l'argument est faux et vice-versa.
true()	La fonction **true** retourne true.
false()	La fonction **false** retourne false.

Évaluation d'une expression XPath

- Soit l'expression ***Xpath*** suivante:

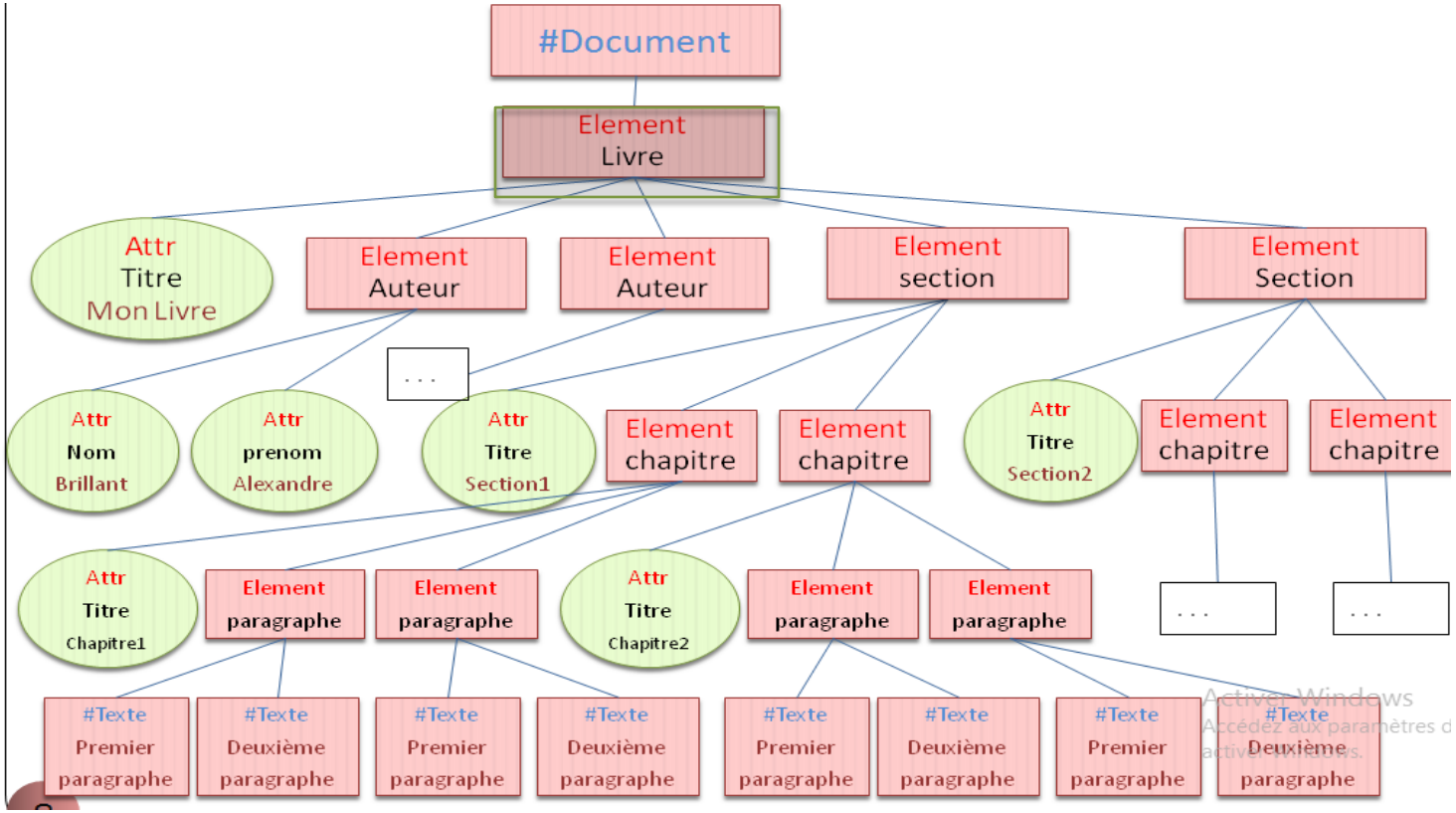
  /etape1/etape2/….

  1. À partir du nœud contexte, on évalue l'étape 1 ; on obtient un ensemble de nœuds ;
  2. On prend alors, un par un, les nœuds de cet ensemble, et on les considère chacun à leur tour comme nœud contexte pour l'évaluation de l'étape 2 ;
  3. À chaque étape, on prend successivement comme nœud contexte chacun des nœuds faisant partie du résultat de l'étape précédente.

- Pour évaluer une expression avec les navigateurs, utilisez ce formulaire: https://perso.univ-rennes1.fr/pierre.nerzic/XML/fichiers/xpath.html

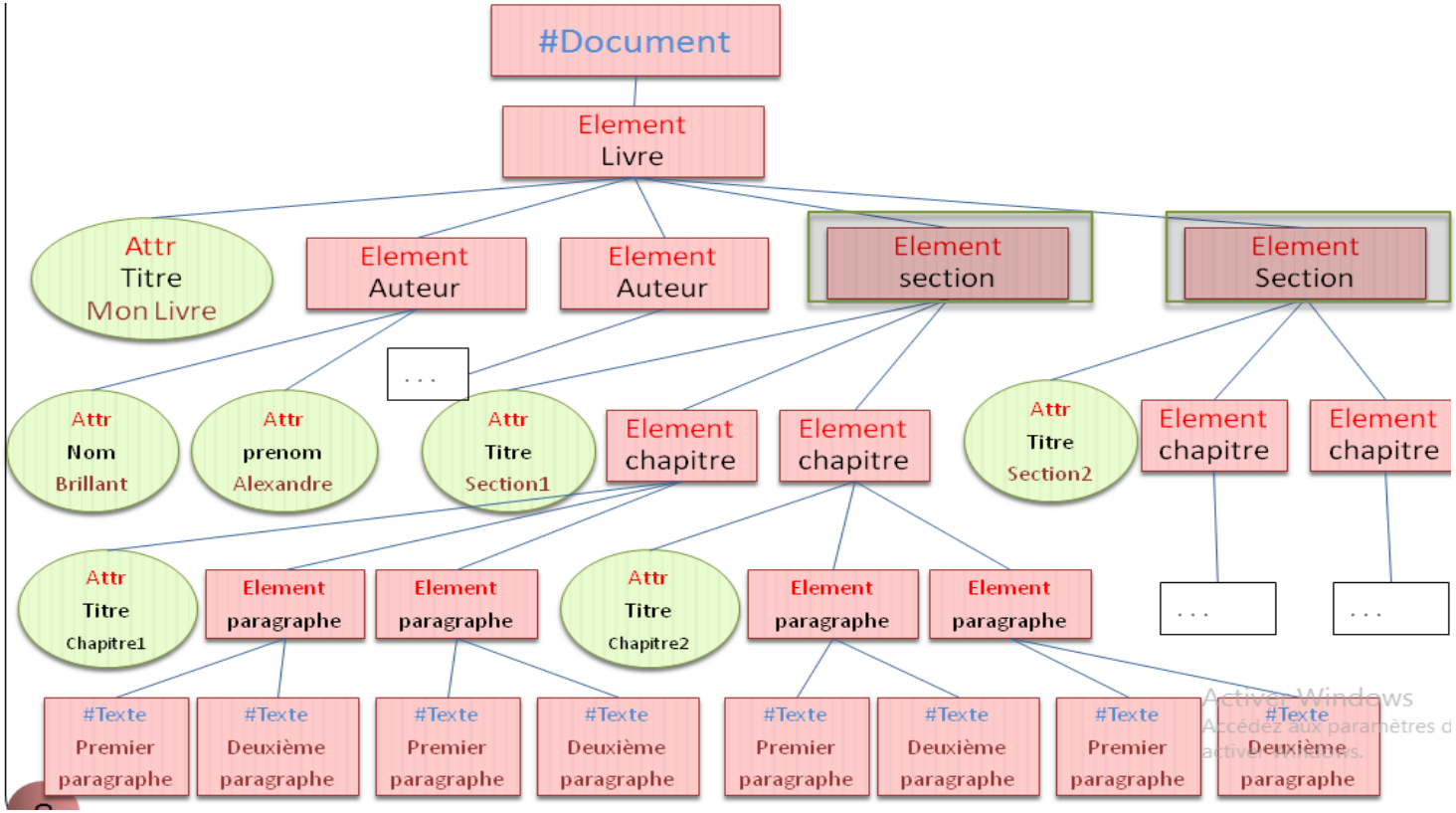
Évaluation d'une expression XPath

- Exemple : /Livre/section/child::chapitre[@titre="Chapitre 1"]



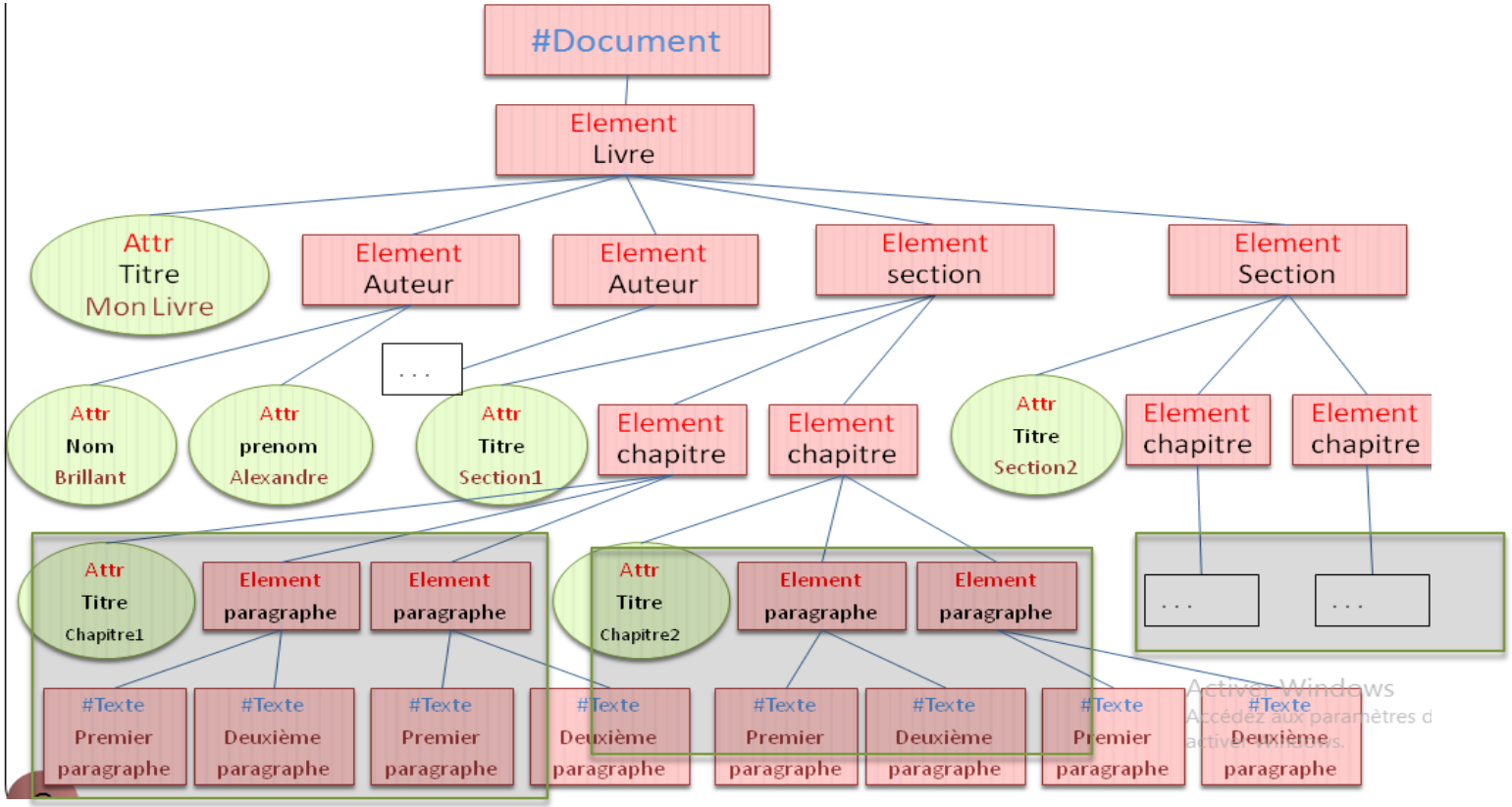
Évaluation d'une expression XPath

- Exemple : /Livre/section/child::chapitre[@titre="Chapitre 1"]



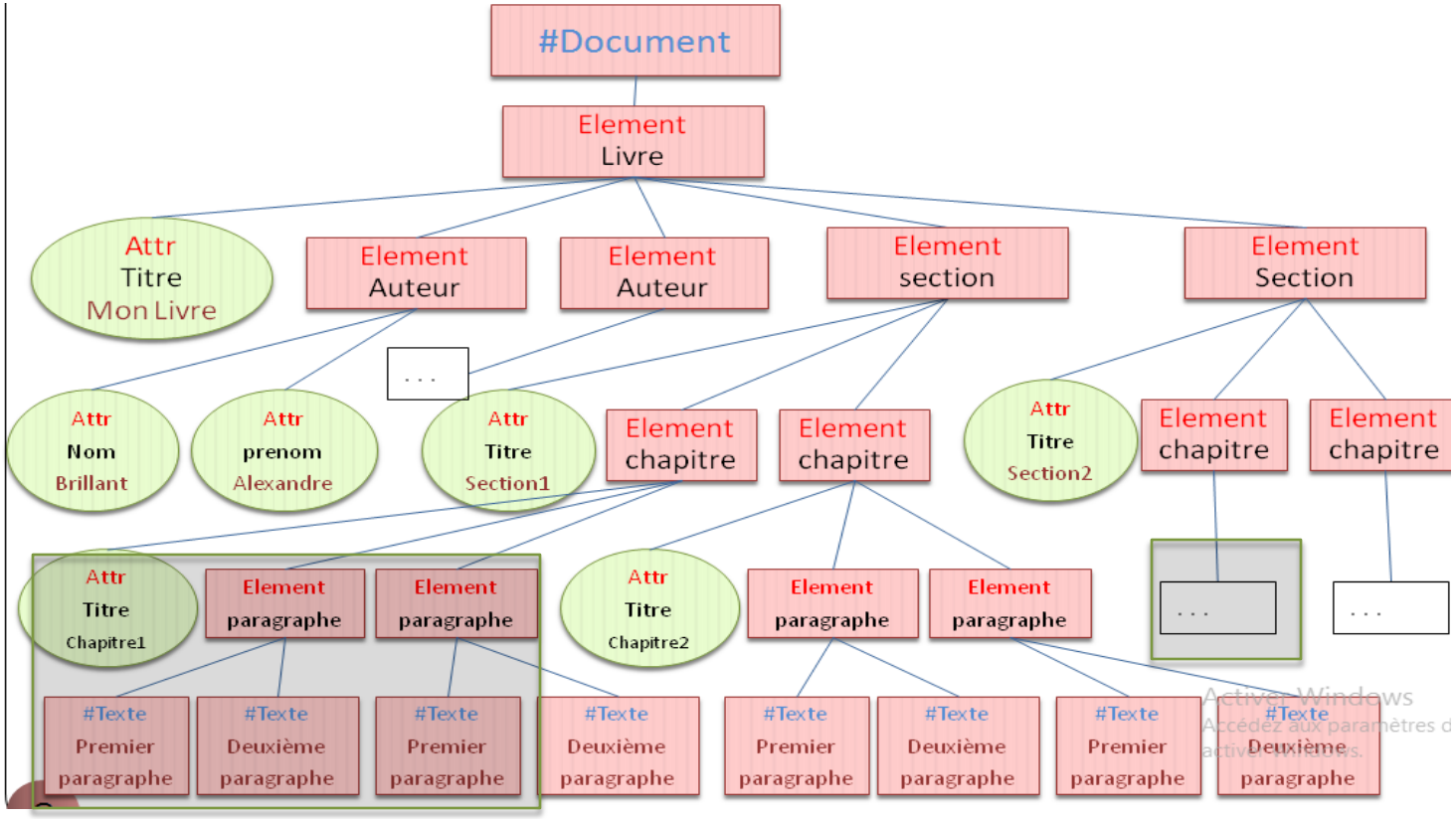
Évaluation d'une expression XPath

- Exemple : /Livre/section/child::chapitre[@titre="Chapitre 1"]



Évaluation d'une expression XPath

- Exemple : /Livre/section/child::chapitre[@titre="Chapitre 1"]



XQuery

XML Query Language

XQuery

- **XQuery** est un langage (nonXML) défini dans la recommandation du W3C du 23 janvier 2007 pour sa version1.0. La dernière version est consultable sur http://www.w3.org/TR/xquery/.
- **XQuery** est un langage de requêtes XML qui se repose sur **XPath**.
- **XQuery** est un langage procédural, où chaque requête est une expression.
- ***XPath*** est un sous-ensemble de ***XQuery***. C'est à dire que XQuery exécute n'importe quelle requête ***XPath*** directement.

XQuery

- ***XQuery*** diffère du ***SQL*** dans le fait qu'il est un langage procédural et ***SQL*** est un langage déclaratif, un non programmeur peut apprendre SQL, car il inclut des commandes déclaratives simples.
- L'utilisation de ***XQuery*** implique un programme procédural dans lequel l'utilisateur définit une séquence d'action, nécessaire pour générer le résultat attendu.

Exemple :

```
FOR $b IN document("publication.xml")//article
WHERE $b/auteur = "Seif Eddine"
AND $b/année =  "2001"
RETURN $b/titre
```

- Cette requête permet de retourner les titres d'articles écrits par «Seif Eddine » en 2001.

XQuery

- En général, une requête ***XQuery*** se place dans un fichier source *requete.xq*. Ce fichier contient, soit uniquement une requête, soit du code ***XML*** dans lequel il y a des requêtes placées entre {…}.

```
Xquery version "1.0" encoding"utf-8";
<liste-membres>
    {doc ("maVotation2.xml")/Ma-votation/Internaute/
       Membre/concat(@Prénom-p,' ', @Nom-p)
    }
</liste-membres>
```

- Résultat

```
<?xml version="1.0" encoding="UTF-8"?>
<liste-membres>
    Jean Solace Marcel Martin Jacques Dupont Edgard Morino
</liste-membres>
```

Modèle de données XQuery

- Une instance du modèle de données ***XQuery***, appelée ***valeur***, est une ***séquence*** d'***item***s où un ***item*** est soit un ***nœud*** soit une ***valeur atomique***.
- Une ***valeur*** peut être un entier seul (*séquence* à un élément) ou encore une *collection* de large documents XML (où chaque document est désigné par son *nœud racine*).
- Voici quelques exemple de valeurs :
  - (47) : ***séquence*** composée d'un seul *item*, cet *item* étant une *valeur atomique*
  - (<a/>) : ***séquence*** composée d'un seul *item*, cet *item* étant *un nœud élément*
  - (1, 2, 3) : ***séquence*** composée de trois *items*, ces *items* étant des *valeurs atomiques*
  - (47, <a/>, "Hello") : ***séquence*** composée de trois *items*, le premier *item* est une *valeur atomique* (entier), le deuxième *item* est un *nœud élément* et le dernier *item* est une *valeur atomique* (chaîne de caractères).
  - () : ***séquence*** vide
  - un ou plusieurs documents ***XML*** (une collection).

Modèle de données XQuery

Quelques détails concernant les ***séquences*** :

- Un ***séquence*** de taille 1 est équivalente à l'***item*** seul (47).
- Une ***séquence*** ne peut pas contenir une autre ***séquence***.
- La notion de valeur nulle n'existe pas en ***XQuery***.
- Une ***séquence*** est ordonnée (donc deux ***séquences*** contenant les mêmes items dans les ordres différents sont différentes).
- Les mots clef du langage ***XQuery*** doivent être écrits en minuscule.
- Des commentaires peuvent être ajoutés en respectant la syntaxe

(:ceci est mon commentaire :)

Les expressions XQuery

- Une expression ***XQuery*** prend en entrée une ***valeur*** (une séquence d'items) et renvoie une ***valeur***.
- Une valeur constante est elle-même une ***expression***
- Il existe 7 types d'***expression***s qui sont :
  1. Les expressions de chemins (Xpath),
  2. constructeurs d'élément,
  3. L'expression FLWOR(expression composée par les clauses : for, let, where , order by et return),
  4. Les expressions qui incluent les opérateurs et les fonctions,
  5. Les expressions de quantifications,
  6. Les expressions conditionnelles (if then else),
  7. Et les expressions qui testent ou modifient les types de données.

Récupérer un document ou une collection

- Une ***requête XQuery*** prend généralement en entrée un *document* ou une *collection de documents*.

- Ces entrées sont spécifiées à l'aide des fonctions suivantes :
  - doc() prend en entrée l'URI d'un document XML et renvoie le nœud racine du document.
  - collection() prend en entrée l'URI d'une collection et renvoie une séquence composée des nœuds racine des documents de la collection.

Récupérer un document ou une collection

- Une ***requête XQuery*** prend généralement en entrée un *document* ou une *collection de documents*.

- Ces entrées sont spécifiées à l'aide des fonctions suivantes :
  - doc() prend en entrée l'URI d'un document XML et renvoie le nœud racine du document.
  - collection() prend en entrée l'URI d'une collection et renvoie une séquence composée des nœuds racine des documents de la collection.

XPath/XQuery

- Toute ***expression XPath*** est une requête ***XQuery***.
- Voici une expression ***XPath / XQuery*** recherchant tous les films parus en 2005.

```
collection('movies')/movie[year=2005]/title
```

- Le résultat est un ***séquence*** de nœuds title : :

```
<title>A History of Violence</title>
<title>Match Point</title>
```

- ***Remarque***
- ***L'expression XPath*** est évaluée pour chaque ***Document*** de la séquence collection('movies').

XQuery: Constructeurs

- ***XQuery*** permet la construction de nouveaux éléments, dont le contenu peut mêler des éléments littéraux, des valeurs, ou le résultat d'autres requêtes (composition !).

```
<titles>
{collection('movies')//title}
</titles>
```

Remarque

- Une expression *E* doit être encadrée par des accolades {} pour être reconnue.

XQuery: Constructeurs explicites

- Plusieurs constructeurs explicites existent, dont en particulier :
  - "attribute {nom-att} {valeur-att}" : pour créer un attribut ;
  - "element {nom-element} {valeur-element}" : pour créer un élément ;
  - "document {valeur-document}" : pour créer le document, pratique pour mettre des instructions de traitement ou des commentaires avant la racine ;
  - "comment {valeur-commentaire}" : pour créer un commentaire ;
  - "processing-instruction {cible} {valeur-pi}" : pour créer une instruction de traitement.

XQuery: Constructeurs explicites

- Plusieurs constructeurs explicites existent, dont en particulier :
- ***Exemple***:

```
xquery version "1.0" encoding "utf-8";

element liste-membres {doc("maVotation2.xml")/Ma-votation/
   Internaute/Membre/ concat(@Prénom-p, ' ', @Nom-p)}
```

- Le résultat de cette requête sera :

```
<liste-membres>
Jean Solace Marcel Martin Jacques Dupont Edgard Morino
</liste-membres>
```

- Bien sûr, le résultat est du XML bien formé.

XQuery: Variables

- Des variables peuvent être utilisées dans les expressions ***XQuery***.

Par exemple :

```
<employee empid="{$id}">
    <name>{$name}</name>
    {$job}
    <deptno>{$deptno}</deptno>
    <salary>{$SGMLspecialist+100000}</salary>
</employee>
```

- où les variables *$id*, *$name*, *$job*, *$deptno* and *$SGMLspecialist* sont instanciées (une valeur est associée à chaque variable).

XQuery: L'expression FLWOR

- L'opérateur ***FLWOR*** est un opérateur complexe qui permet de parcourir une ou plusieurs ***séquences*** (et d'effectuer ainsi des opérations de type "jointure").
- La forme de cet opérateur comprend un certain nombre de clauses dans l'ensemble suivant:
  - For : itération sur une séquence.
  - Let : définition ou instanciation d'une variable.
  - Where : prédicat.
  - Order by : tri du résultat.
  - Return : constructeur de résultat.
- Plus exactement, l'expression régulière décrivant la succession des clauses est: "(for|let)+where?order_by?return". Par conséquent, la forme minimale comporte un "for« ou un "let« et un "return« .

XQuery: L'expression FLWOR

- Dans l'une de ses formes les plus simples, ***FLW0R*** est une alternative à ***XPath***.

- ***Exemple***:

```
//actor[birth_date>=1960]/last_name
```

- est équivalent à l'expression (requête) ***XQuery*** suivante :

```
let $year:=1960
for $a in doc("SpiderMan.xml")//actor
where $a/birth_date >= $year
return $a/last_name
```

XQuery: L'expression FLWOR

- Les clauses for et let. instancient des variables.
- for instancie une variable en lui faisant prendre (successivement) les valeurs des items d'une séquence en entrée.
- ***Par exemple*** : for $x in /company/employee
  - instancie la variable x en lui faisant successivement prendre pour valeur chacun des éléments employee de la séquence issue de l'évaluation de /company/employee.

- let instancie une variable en lui faisant prendre la valeur de la séquence
- complète
- ***Par exemple :*** let $x := /company/employee
  - instancie la variable x en lui faisant prendre la valeur de la séquence issue de l'évaluation de /company/employee.

XQuery: L'expression FLWOR

- Instanciation d'une variable :

```
let $years:=(1960,1978)
return $years
```

- Parcours d'une séquence :

```
for $a in (1,2,9)
Return <number> {$a} </number>
```

Résultat:

```
<number> 1 </number>
<number> 2 </number>
<number> 9 </number>
```

XQuery: L'expression FLWOR

- Afin d'illustrer le fonctionnement de cet opérateur, prenons l'exemple simple suivant :

```
for $i in (1 to 3), $j in (4 to 6)
let $k := ($i to $j)
return <res>{$i}/{$j}/{$k}</res>
```

- Dans cet exemple, le couple "($i,$j)" prend ses valeurs dans le produit cartésien "(1,2,3) × (4, 5, 6)". Pour chaque couple, "$k" est la séquence des entiers entre ces deux valeurs. Le résultat sera donc :

```
<res>1/4/1 2 3 4</res>
<res>1/5/1 2 3 4 5</res>
<res>1/6/1 2 3 4 5 6</res>
<res>2/4/2 3 4</res>
<res>2/5/2 3 4 5</res>
<res>2/6/2 3 4 5 6</res>
<res>3/4/3 4</res>
<res>3/5/3 4 5</res>
<res>3/6/3 4 5 6</res>
```

XQuery: L'expression FLWOR

- Un autre example :

```
for $i in (-3,-2,-1,0,1,2,3)
let $j := $i + 1
return <suivant>{$j} est le nombre suivant {$i}</suivant>
```

- Résultat

```
<suivant> -2 est le nombre suivant -3</suivant>
<suivant> -1 est le nombre suivant -2</suivant>
<suivant> 0 est le nombre suivant -1</suivant>
<suivant> 1 est le nombre suivant 0</suivant>
<suivant> 2 est le nombre suivant 1</suivant>
<suivant> 3 est le nombre suivant 2</suivant>
<suivant> 4 est le nombre suivant 3</suivant>
```

XQuery: L'expression FLWOR

- La clause ***where*** *exp* permet de filtrer le résultat par rapport au résultat booléen de l'expression *exp (= prédicat dans l'expression de chemin)*.
- Requête:

```
<livre>
{ for $a in document("bib.xml")//book
where $a/author[1]/Nom eq « Elmasri"
return $a/@title}
</livre>
```

- Résultat:

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

XQuery: L'expression FLWOR

- ***Exemple***. Nous allons rechercher les titres des sondages qui ont un mot-clé "Écologie".

```
xquery version "1.0" encoding "utf-8";
<liste>{
for $vote in doc("maVotation2.xml")/Ma-votation/Internaute/
   Membre/Vote[Mot-clé = 'Écologie']
return <vote>{$vote/@Titre}</vote> }
</liste>
```

- ***Résultat*** :

```
<?xml version="1.0" encoding="UTF-8"?> <liste>
<vote Titre="Pour l'énergie éolienne"/>
<vote Titre="Rénovation de la station d'épuration"/>
<vote Titre="Panneaux solaires"/> </liste>
```

XQuery: L'expression FLWOR

- ***Exemple***. Nous allons rechercher les titres des sondages qui ont un mot-clé "Écologie" ordonnés selon le nom de la matière (ordre alphabétique) :.

```
xquery version "1.0" encoding "utf-8";
<liste>{
for $vote in doc("maVotation2.xml")/Ma-votation/Internaute/
   Membre/Vote[Mot-clé = 'Écologie']
order by $vote/@Titre ascending
return <vote>{$vote/@Titre}</vote> }
</liste>
```

- ***Résultat*** :

```
<?xml version="1.0" encoding="UTF-8"?> <liste>
<vote Titre="Panneaux solaires"/> </liste>
<vote Titre="Pour l'énergie éolienne"/>
<vote Titre="Rénovation de la station d'épuration"/>
```

XQuery Tests: if-then-else

- ***Requête:***

```
<livres>
{ for $b in document("bib.xml")//book
where $b/author/Nom = "Rigaux"
return if ($b/@year > 2000)
        then <livre recent="true"> {$b/@title} </livre>
        else <livre> {$b/@title} </livre> }
</livres>
```

- ***Résultat:***

```
<?xml version="1.0"?>
<livres>
<livre title="Comprendre XSLT"/>
<livre recent="true" title="Spatial Databases"/>
</livres>
```

XQuery :Quantification

- ***Quantification existentielle*** :
- some $var in expr1 satisfies expr2: il existe au moins un noeud retourné par l'expression expr1 qui satisfait l'expression expr2.
- ***Quantification universelle*** :
- every $var in expr1 satisfies expr2: tous les nœuds retournés par l'expression expr1 satisfont l'expression expr2
- ***Requête:***

```
for $a in document("bib.xml")//author
where every $b in document("bib.xml")//book[author/la =
   $a/la]
satisfies $b/publisher="Morgan Kaufmann Publishers"
return string($a/Nom)
```

- ***Résultat:***

Scholl, Voisard, Abiteboul, Buneman, Suciu

XQuery :Jointure

- **Requête**:

```
for $b in document("bib.xml")//book
return element livre {
    attribute titre {$b/@title},
    for $a in $b/author
    return element auteur {
        attribute nom {$a/la},
        for $p in document("addr.xml")//person
        where $a/la = $p/name
        return attribute institut {$p/institution}
    }
}
```

XQuery :Jointure

- ***Requête***:

```
<livre titre="Comprendre XSLT">
    <auteur nom="Amann" institut="CNAM"/>
    <auteur nom="Rigaux"/>
</livre>,
<livre titre="Spatial Databases">
    <auteur nom="Rigaux"/>
    <auteur nom="Scholl" institut="CNAM"/>
    <auteur nom="Voisard" institut="FU Berlin"/>
</livre>,
<livre titre="Data on the Web">
    <auteur nom="Abiteboul"/>
    <auteur nom="Buneman"/>
    <auteur nom="Suciu"/>
</livre>
```

SQL/XML

SQL/XML

- Les langages de requêtes basés sur **SQL** utilisent des ordres SELECT modifiés dont les résultats sont transformés en **XML**.
- Un certains nombre de langages propriétaires de ce type son actuellement disponibles.
- En plus des langages propriétaires, un certain nombre de sociétés se sont réunis pour standardiser des extensions ***XML*** au langages ***SQL***.
- **SQL/XML** est une extension du langage ***SQL*** afin d'obtenir des flux ***XML*** depuis une BD relationnelle.
- **SQL/XML** ajoute un certain nombre de fonctions à ***XML***, de telle façon qu'il soit possibles de construire des éléments et des attributs ***XML*** à partir de données relationnelles.

SQL/XML

- Plusieurs fonctions ont été proposées:
- XMLElement - qui construit un élément XML à partir d'une valeur issue de la base de données
- XMLAttributes - qui place dans un attribut XML une valeur
- XMLAgg - fonction agrégat qui construit un élément composé à partir d'une liste de valeurs
- XMLConcat - qui concatène deux ou plusieurs éléments XML
- XMLForest : construit une séquence d'éléments XML
- ……

- ***Exemples:***

```
SELECT Ordres.Numero,
        XMLElement (NAME "Ordre",
        XMLAttributes (Ordres.Numero AS Numero),
        XMLElement (NAME "Date", Ordres.Date),
        XMLElement(NAME "client", Ordres.client)) AS XMLDocument
FROM Ordres
```

- Cette requête construit un ensemble résultant possédant deux colonnes. La première colonne contient un numéro d'ordre des ventes et la seconde colonne contient un *document XML*.
- Le document XML pour la ligne d'ordre de numéro 150 pourrait être par exemple :

```
<Ordre Numero ="150">
    <Date>10/29/02</Date>
    <Client>Ali Ahmed</Client>
</Ordre>
```

Merci de votre attention

Des Questions

K_menghour@yahoo.fr