MEDAS : Bases de données et Systèmes d'informations
XML

Raphaël Fournier-S'niehotta

Équipe Vertigo
Laboratoire CEDRIC
Conservatoire National des Arts & Métiers, Paris, France

le cnam

Plan du cours

1 XPath

2 XQuery

XPath

XPath est un langage de requêtes

- non XML,
- permettant l'accès à des parties d'une donnée XML via l'expression de chemin menant à un ensemble de nœuds (sous-arbres) d'un document XML.

XPath est utilisé dans

XQuery	XPath fait partie du langage d'interrogation XQuery,
XSLT	pour la sélection de la partie des données à transformer,
XML Schema	pour les expressions de contraintes d'unicité,
XPointer	pour l'identification de fragments,
XLink	pour ancrer les liens hypertextes,
...	

XPath

- XPath 1.0 est utilisé dans XSLT 1.0 (que nous étudierons dans ce cours),
- XPath 1.0 est inclus dans XPath 2.0,
- XPath 2.0 est inclus dans XQuery (que nous étudierons dans ce cours).

Nous commençons donc tout naturellement par étudier XPath 1.0.

Plan du cours

1. XPath
   - Les chemins de localisation
   - Child
   - Parent
   - Ancestor
   - Ancestor-or-self
   - Descendant
   - Descendant-or-self
   - Following
   - Following-sibling
   - Preceding
   - Preceding-sibling
   - Attribute
   - Test de noeud
   - Les prédicats

Une intuition

Permet de "pointer" (localiser) des sous-arbres d'un document XML.

Intuition

Par navigation : en navigant dans l'arbre de "pas en pas" à partir de la racine, on va chercher à atteindre certaines de ses parties.

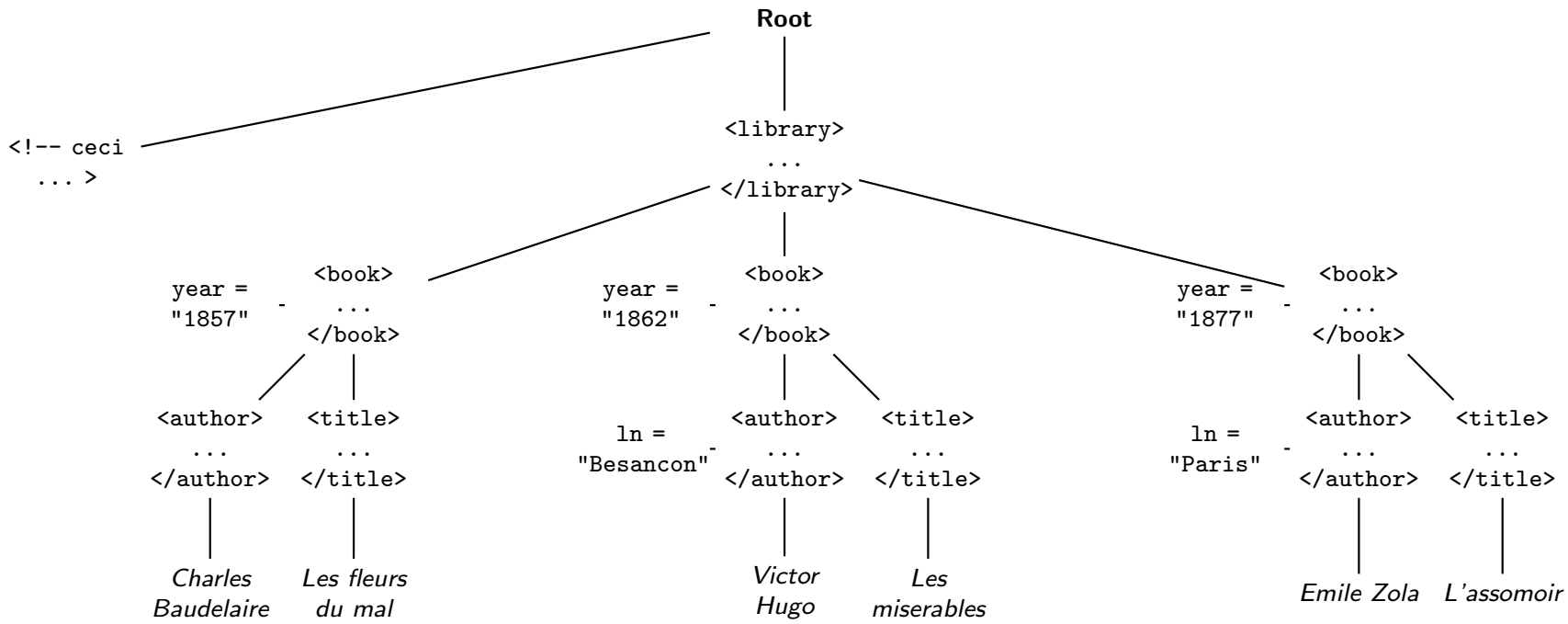
Commençons par un exemple

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Ceci est un fichier XML d'exemple pour le cours NFE204 -->

<!DOCTYPE biblio [
<!ELEMENT biblio (book)*>
<!ELEMENT book (author*,title)>
<!ATTLIST book year CDATA #REQUIRED>
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ATTLIST author ln CDATA #IMPLIED>
]>

<library>
  <book year="1857" >
    <author>Charles Baudelaire</author>
    <title>Les fleurs du mal</title>
  </book>
  <book year="1862" >
    <author ln="Besancon" >Victor Hugo</author>
    <title>Les miserables</title>
  </book>
  <book year="1877" >
    <author ln="Paris" >Emile Zola</author>
    <title>L'assomoir</title>
  </book>
</library>
```

Listing 1 – Un simple fichier XML

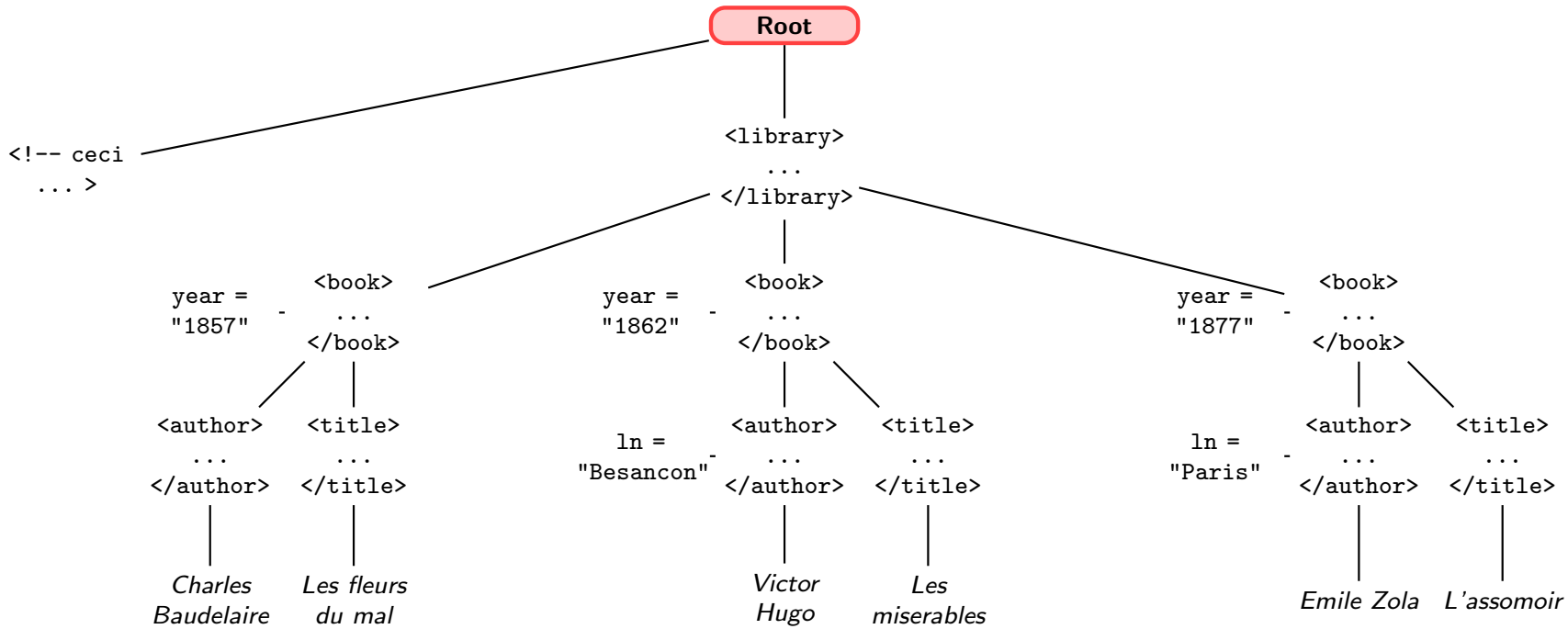
Commençons par un exemple (suite)

```
/child::library/child::book/child::author
```



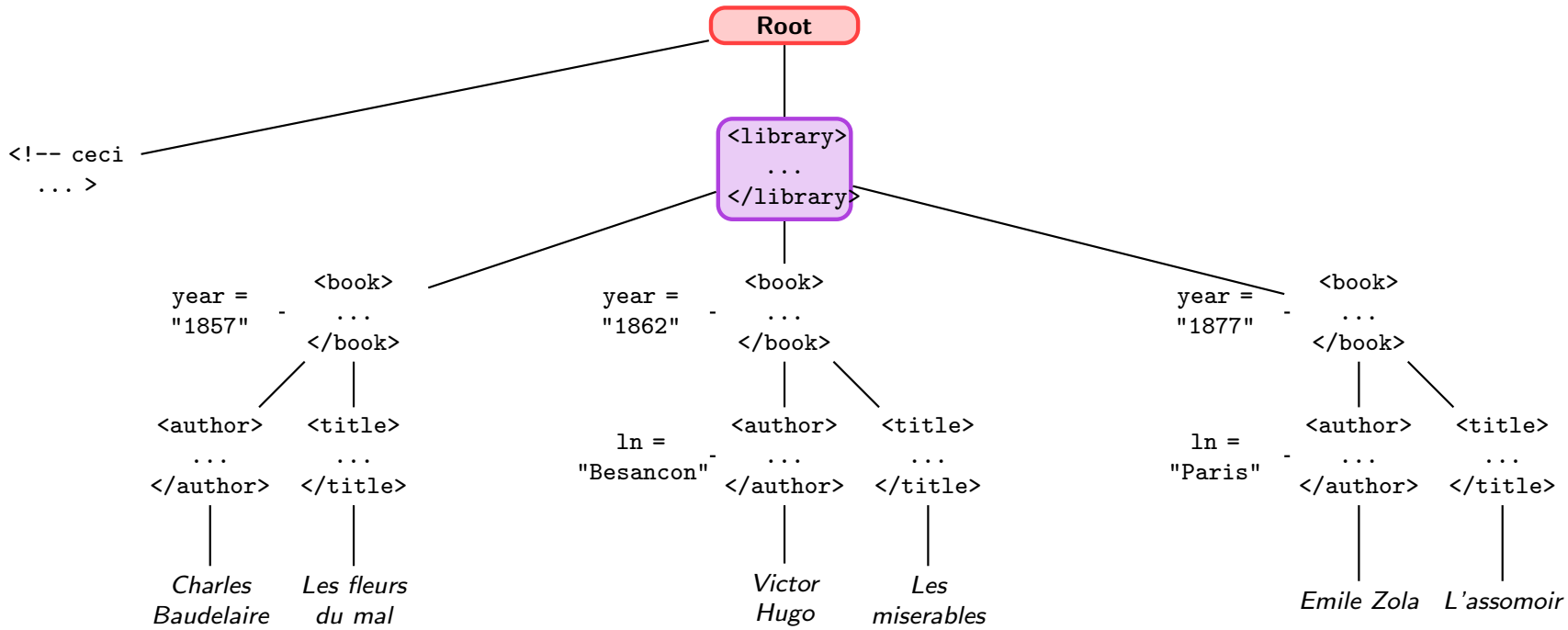
Commençons par un exemple (suite)

```
/child::library/child::book/child::author
```



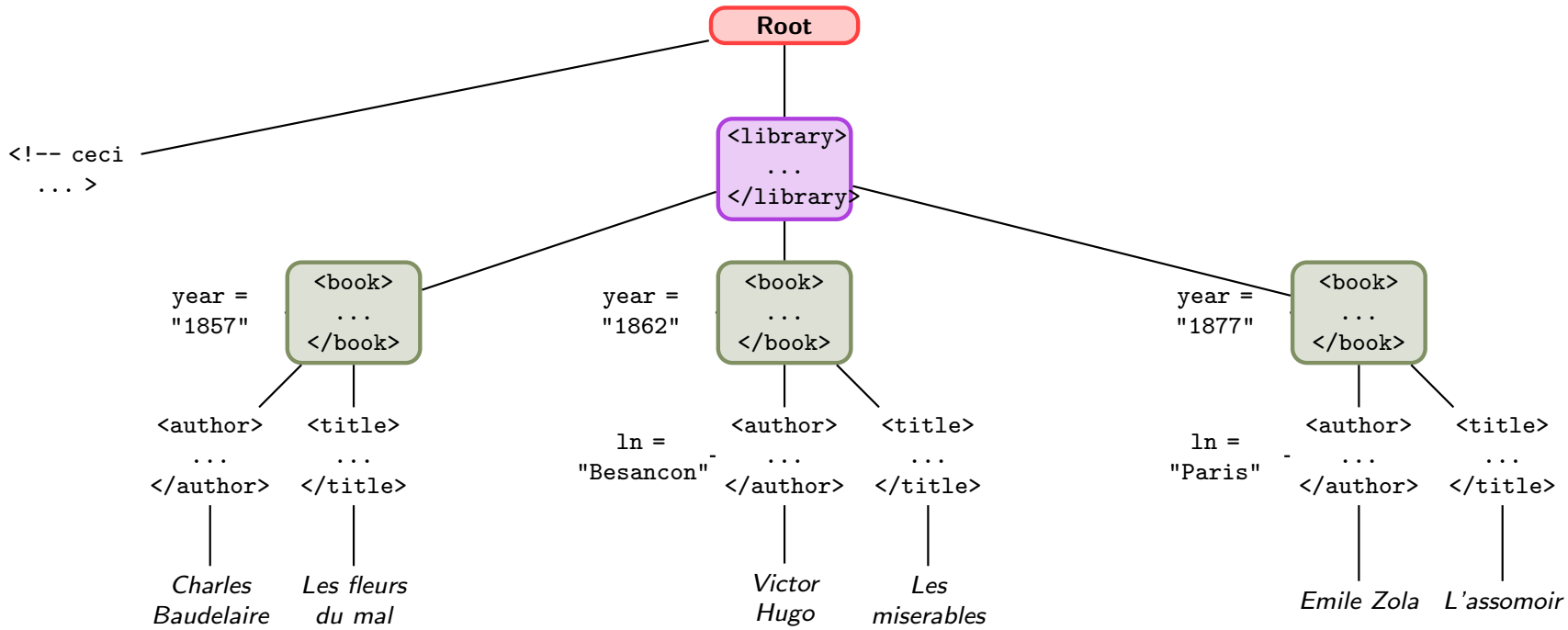
Commençons par un exemple (suite)

/child::library/child::book/child::author



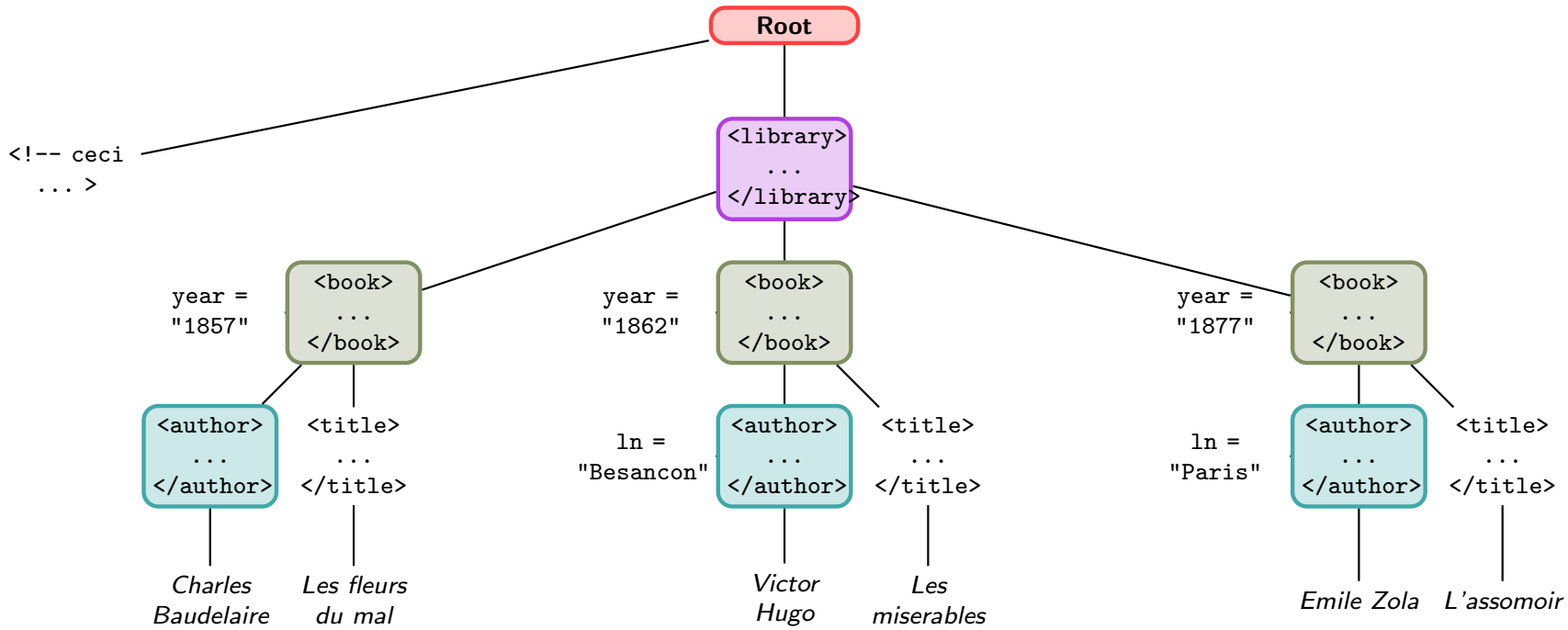
Commençons par un exemple (suite)

/child::library/child::book/child::author

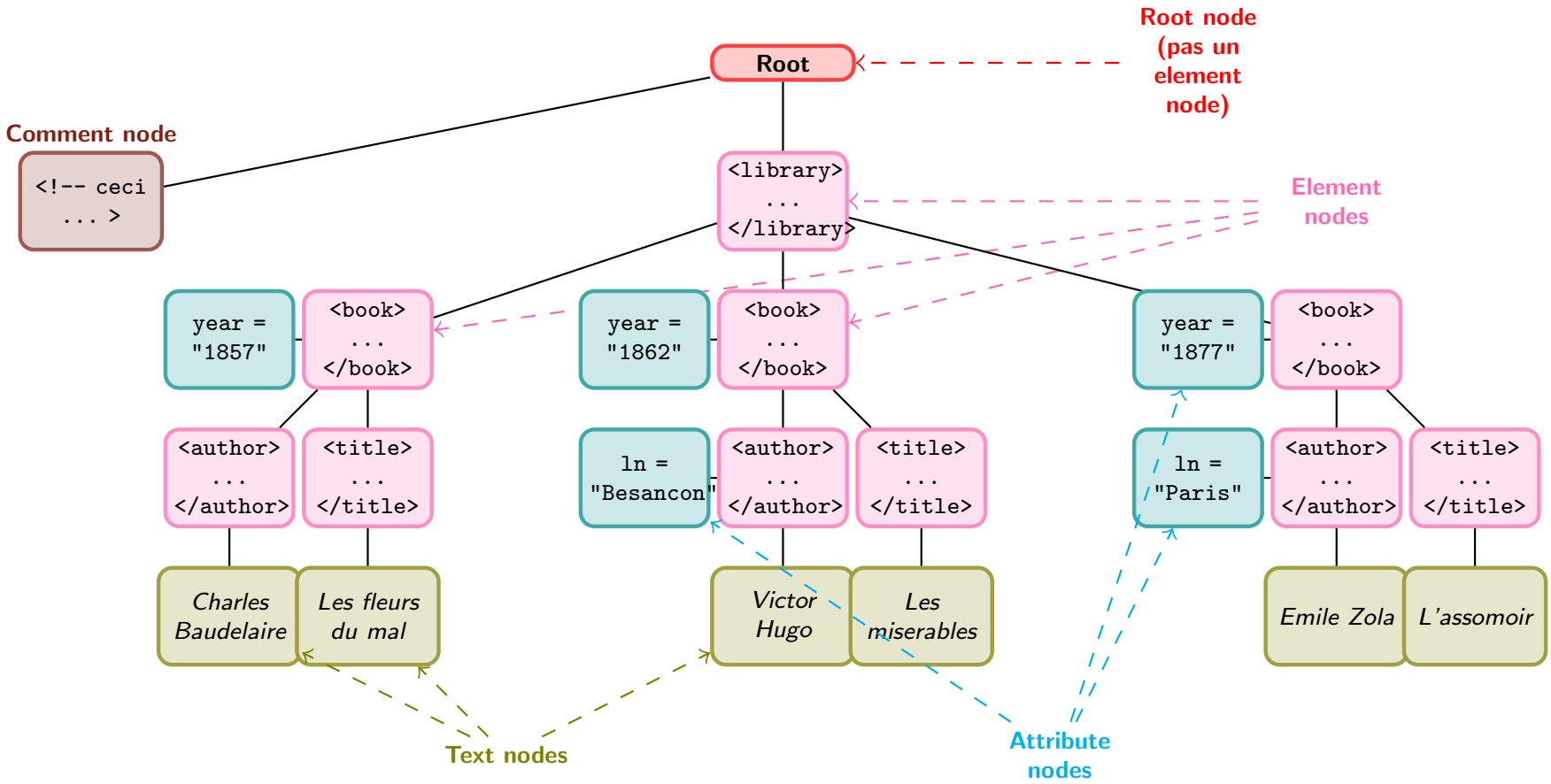


Commençons par un exemple (suite)

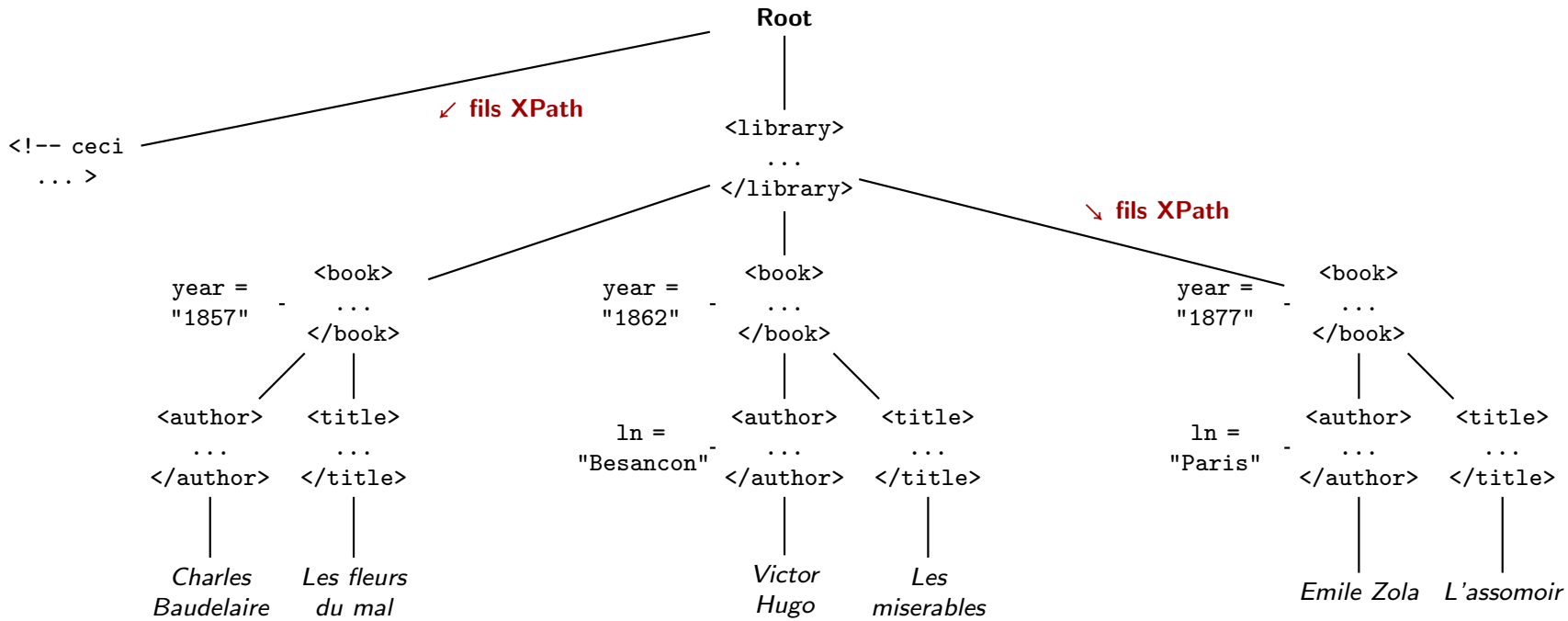
/child::library/child::book/child::author



L'arbre XML vu par XPath



L'arbre XML vu par XPath



⇒ Un fils au sens de XPath est différent d'un sous-élément du modèle de données XML (`http://www.w3.org/TR/xpath-datamodel/`)

le cnam

Expression de chemin XPath (définition simplifiée)

Expression de chemin absolu (évaluée à la racine) :
`/step1/step2/step3/.../stepN`

Chaque `stepi` est une expression de la forme
`axis::node-test[predicate]`
où

`axis` est la direction du pas (`child` par défaut)
Ex : `child` "descend" sur les fils, `attribut` "descend" sur les attributs.

`node-test` est le type de nœud à retenir
Ex : `book` teste si l'élément atteint (p.e. nœud ou attribut) est un élément nommé `book`. Si `*`, tout nom.

`[predicate]` (optionnel) filtre les nœuds selon une expression booléenne
Ex : `[attribute::supplier='store']` teste si le nœud a un attribut supplier dont la valeur est `store`.
Possibilité d'utiliser des fonctions prédéfinies dans XPath dans les prédicats, par exemple : `[position()=2]` teste si le nœud est en seconde position, `[position()=last()]` teste si le nœud est en dernière position.

le cnam

Chemin de localisation : intuition

Intuition

Un chemin de localisation XPath est une suite de mouvements (steps) , chaque mouvement étant éventuellement suivi de tests sur les nœuds atteints.

Chaque *pas* "ramène" un ensemble de nœuds (de sous-arbres associés) à partir desquels est évalué le pas suivant.

Contexte d'évaluation

Une expression de chemin est évaluée par rapport à un *contexte*.

Un contexte $[< N1, N2, ..., Nn >, Nc]$ consiste en :

- Une liste de nœuds $< N1, N2, ..., Nn >$ de l'arbre XML
- Un nœud courant Nc appartenant à la liste de nœuds

Concernant le contexte

La taille du contexte n indique la taille de la liste de nœuds. On peut la récupérer à l'aide de la fonction `last()` ;

La position du nœud contexte $c \in [1, n]$ indique la position du nœud courant au sein de la liste de nœud du contexte. On peut la récupérer à l'aide de la fonction `position()`.

Chemin de localisation : intuition

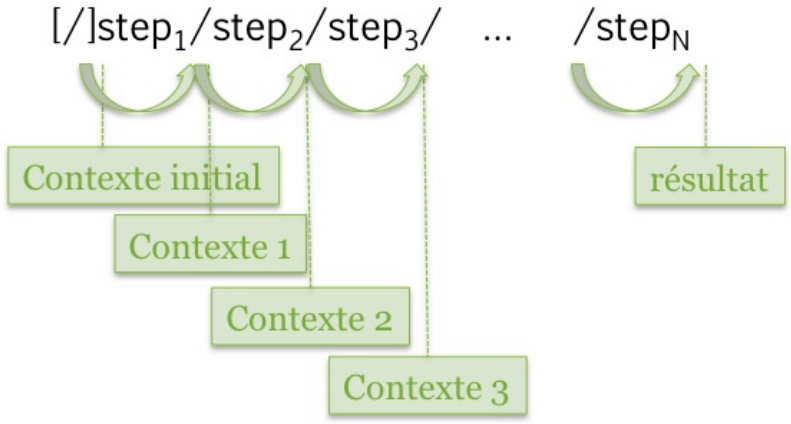


FIGURE 1 – Évaluation d'une requête XPath

Chemin de localisation : intuition (suite)

Un pas $step_i$ est évalué par rapport au contexte produit par l'évaluation de $step_{i-1}$.

Pour $i = 1$ (premier pas) si l'expression est absolue alors le contexte initial est $[< r >, r]$ où r est le nœud racine. Si l'expression est relative alors le contexte est défini par l'environnement.

Pour $i > 1$ si $\mathcal{N} =< N1, N2, ..., Nn >$ est le résultat de l'évaluation du pas $step_{i-1}$ alors $step_i$ est successivement évalué par rapport au context $[\mathcal{N}, Nj]$, pour chaque $j \in [1, n]$.

Le résultat de l'évaluation est l'**ensemble** de nœuds obtenu après l'évaluation du dernier pas.

À un niveau de détail plus fin, pour chaque pas de la forme `axis::node-test[predicate]`, le contexte passe (est transformé) successivement par l'axe (`axis`), puis le test de nœud (`node-test`), puis le prédicat (`[predicate]`).

Poussons la définition

Il existe deux types de chemins de localisation :

chemin absolu le résultat de l'évaluation ne dépend pas du nœud contexte car l'expression est évaluée à partir de la racine `/step1/step2/step3/.../stepN`

chemin relatif le résultat de l'évaluation dépend du nœud contexte car l'expression est évaluée à partir du noeud contexte `step1/step2/step3/.../stepN`

Généralement dans deux situations différentes :

chemin absolu pour une interrogation "*à la bases de données*"

chemin relatif quand les expressions XPath sont intégrées dans un autre langage permettant lui-même de naviguer dans l'arbre (par exemple XSLT).

Expression de chemin XPath

Une expression de chemin XPath est une expression de l'une des formes suivantes :

(chemin absolu) `/step1/step2/step3/.../stepN`

(chemin relatif) `step1/step2/step3/.../stepN`

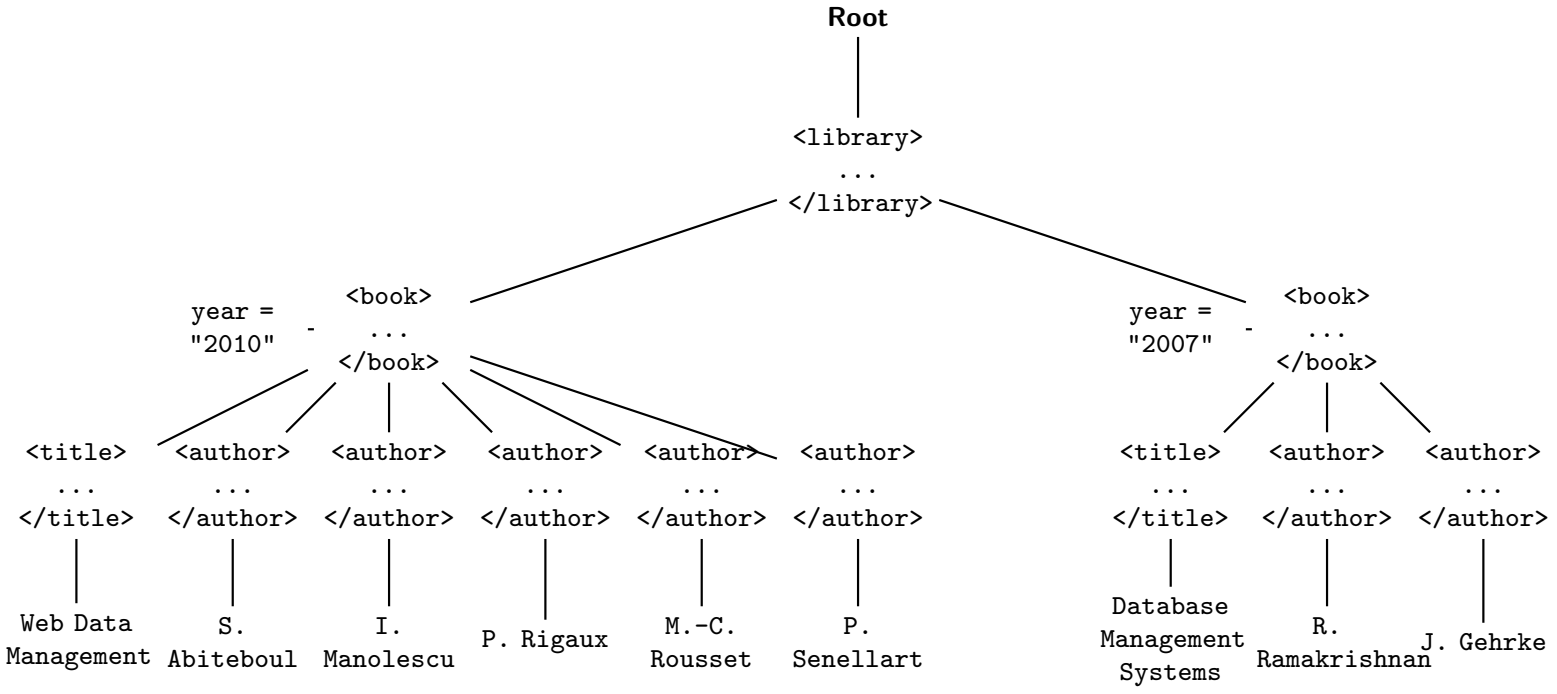
telle que chaque `stepi` (avec $1 \leq i \leq N$) est de la forme
`axis::node-test[predicate]`
où

`axis`	est la direction du pas (`child` par défaut),
`node-test`	est le type de nœud à retenir, et
`[predicate]`	(optionnel) filtre les nœuds selon une expression booléenne.

Axis

Axis name	Semantics
child	Selects all children of the current node
attribute	Selects all attributes of the current node
ancestor	Selects all ancestors (parent, grandparent, etc.) of the current node
ancestor-or-self	Selects all ancestors (parent, grandparent, etc.) of the current node and the current node itself
descendant	Selects all descendants (children, grandchildren, etc.) of the current node
descendant-or-self	Selects all descendants (children, grandchildren, etc.) of the current node and the current node itself
following	Selects everything in the document defined after the closing tag of the current node
following-sibling	Selects all siblings after the current node
namespace	Selects all namespace nodes of the current node
parent	Selects the parent of the current node
preceding	Selects everything in the document that is defined before the start tag of the current node
preceding-sibling	Selects all siblings before the current node
self	Selects the current node

Quelques axes en exemple

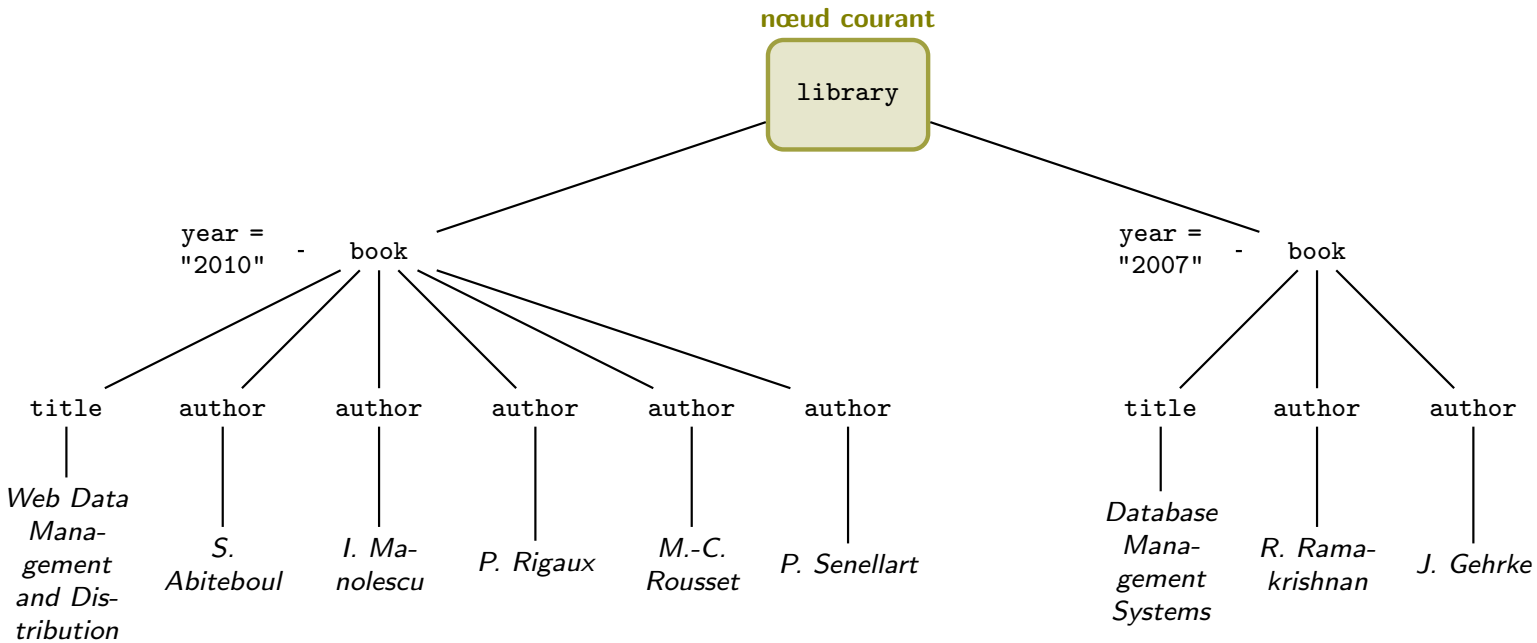


Plan du cours

1. XPath
   - Les chemins de localisation
   - Child
   - Parent
   - Ancestor
   - Ancestor-or-self
   - Descendant
   - Descendant-or-self
   - Following
   - Following-sibling
   - Preceding
   - Preceding-sibling
   - Attribute
   - Test de noeud
   - Les prédicats

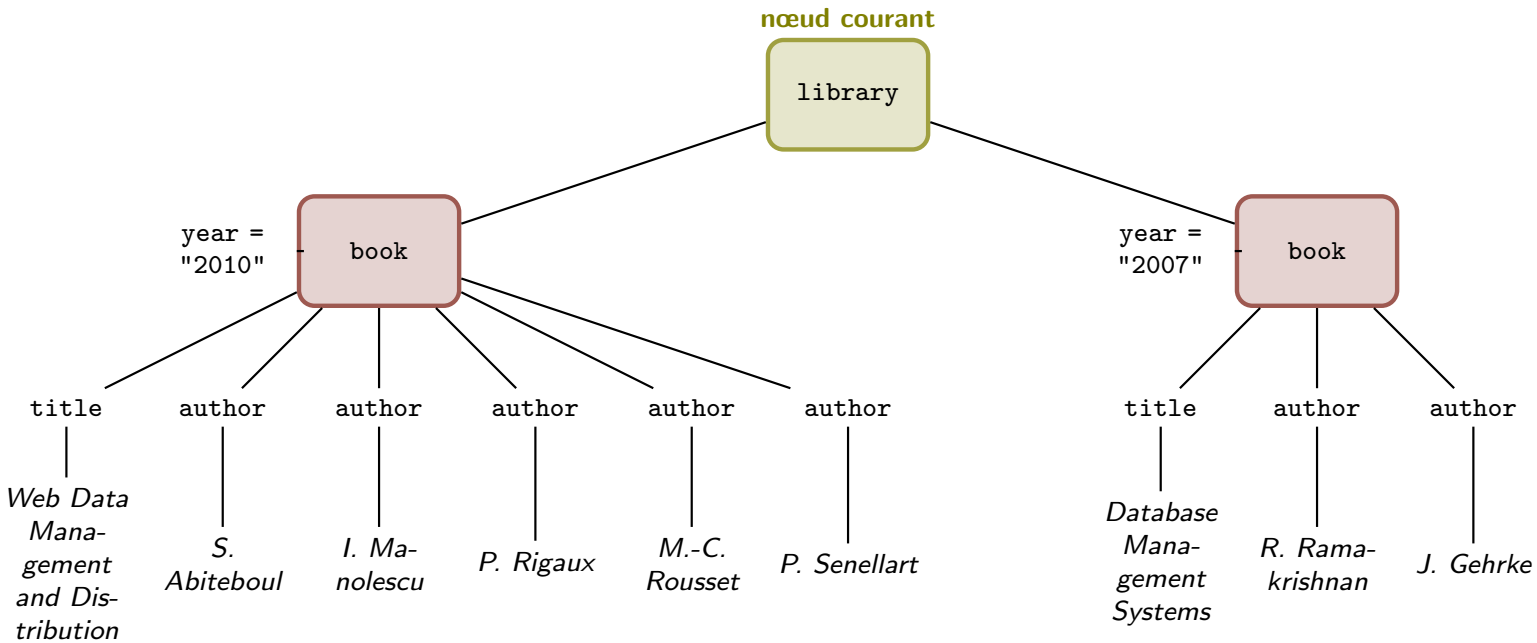
Quelques axes en exemple : Child

`child` : Selects all children of the current node.



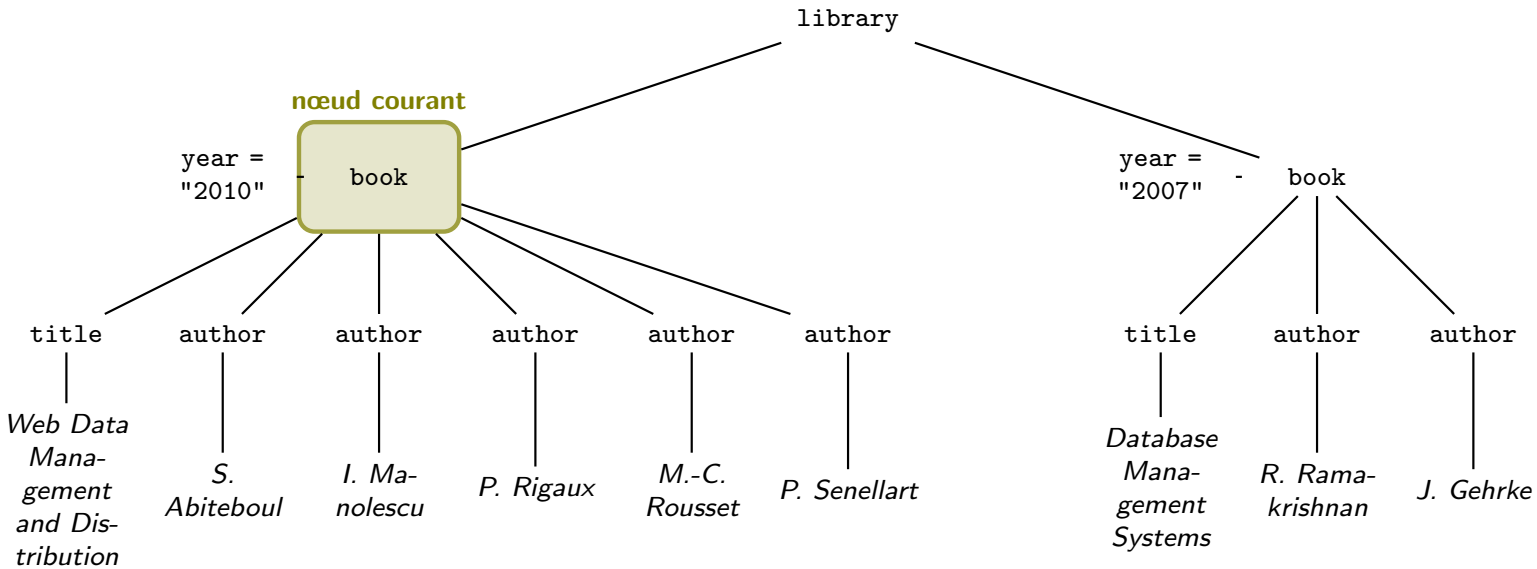
Quelques axes en exemple : Child

`child` : Selects all children of the current node.



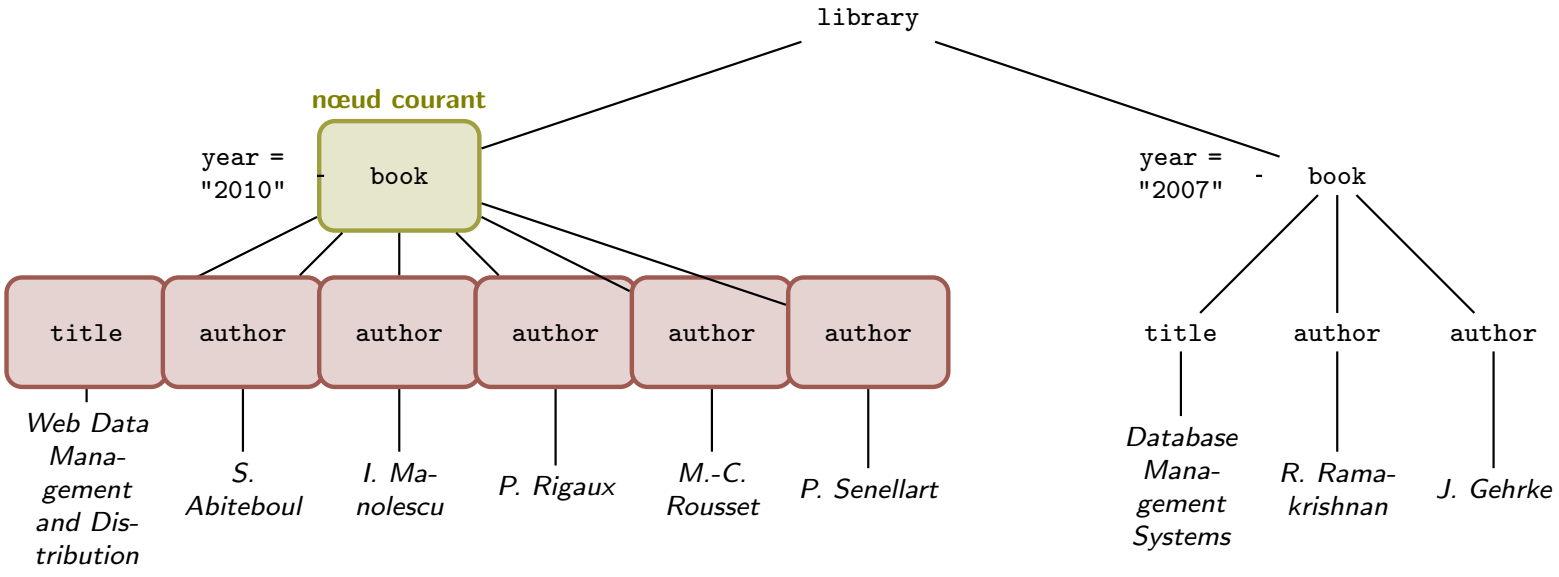
Quelques axes en exemple : Child

`child` : Selects all children of the current node.



Quelques axes en exemple : Child

`child` : Selects all children of the current node.

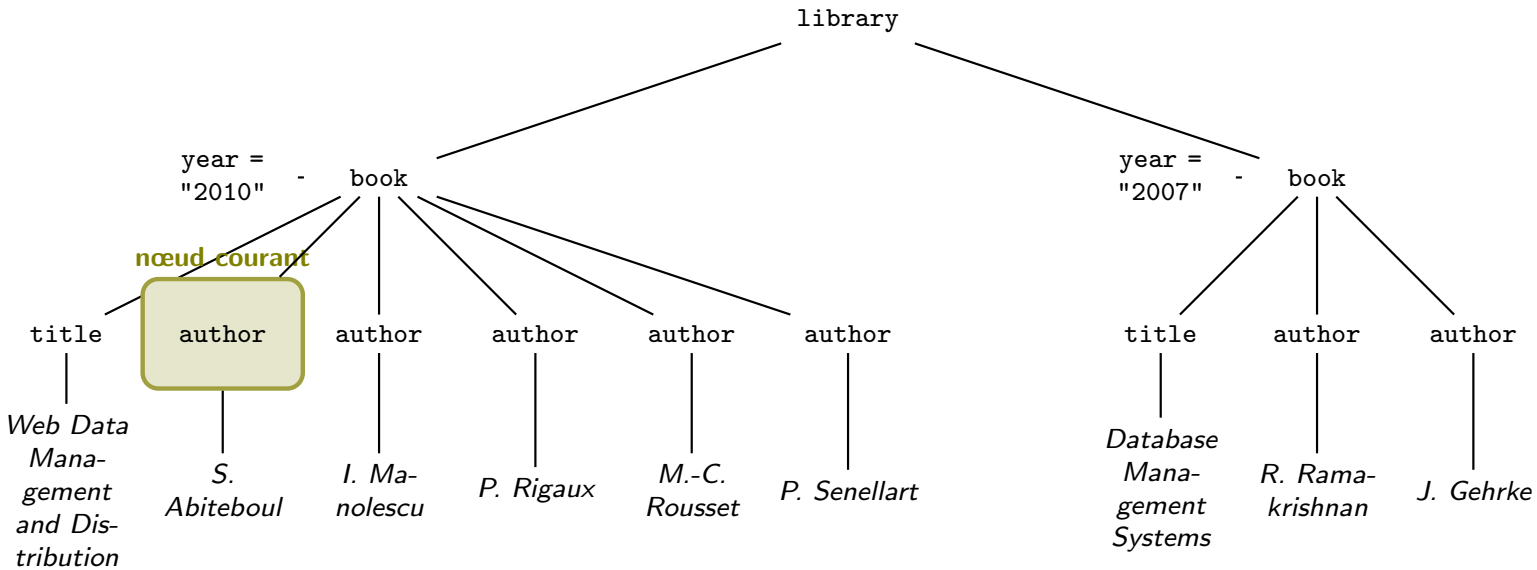


Plan du cours

1. XPath
   - Les chemins de localisation
   - Child
   - Parent
   - Ancestor
   - Ancestor-or-self
   - Descendant
   - Descendant-or-self
   - Following
   - Following-sibling
   - Preceding
   - Preceding-sibling
   - Attribute
   - Test de noeud
   - Les prédicats

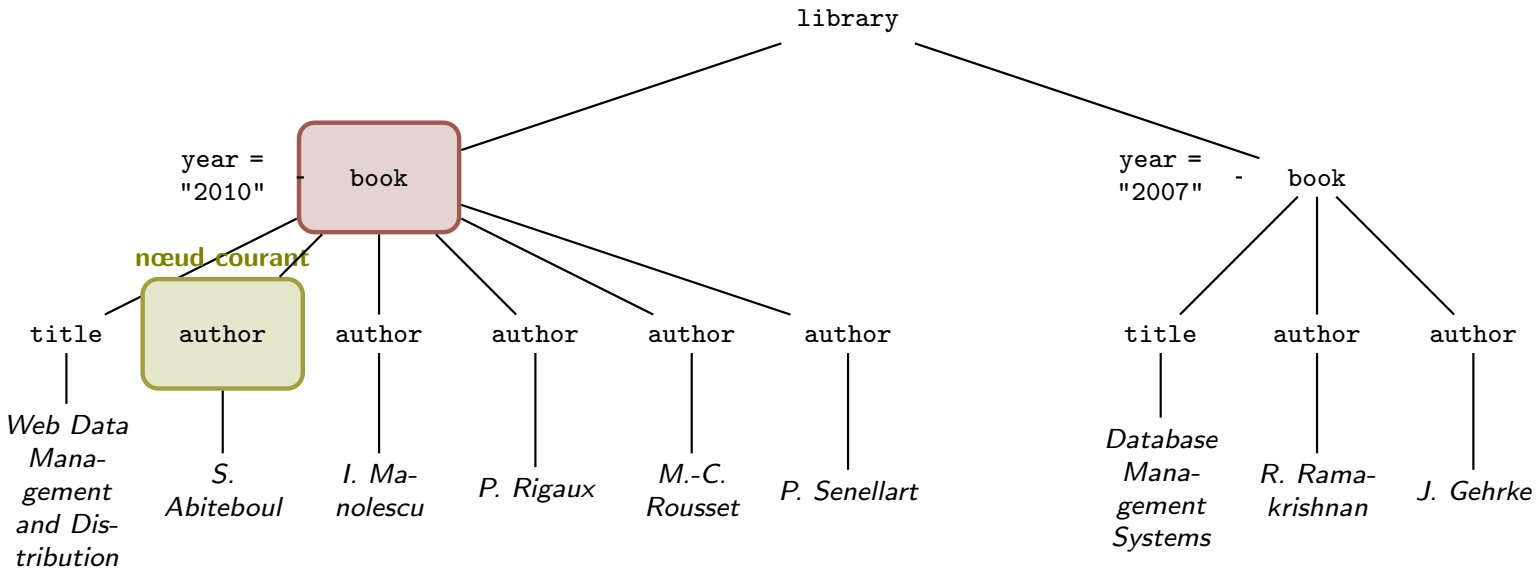
Quelques axes en exemple : Parent

`parent` : Selects the parent of the current node.



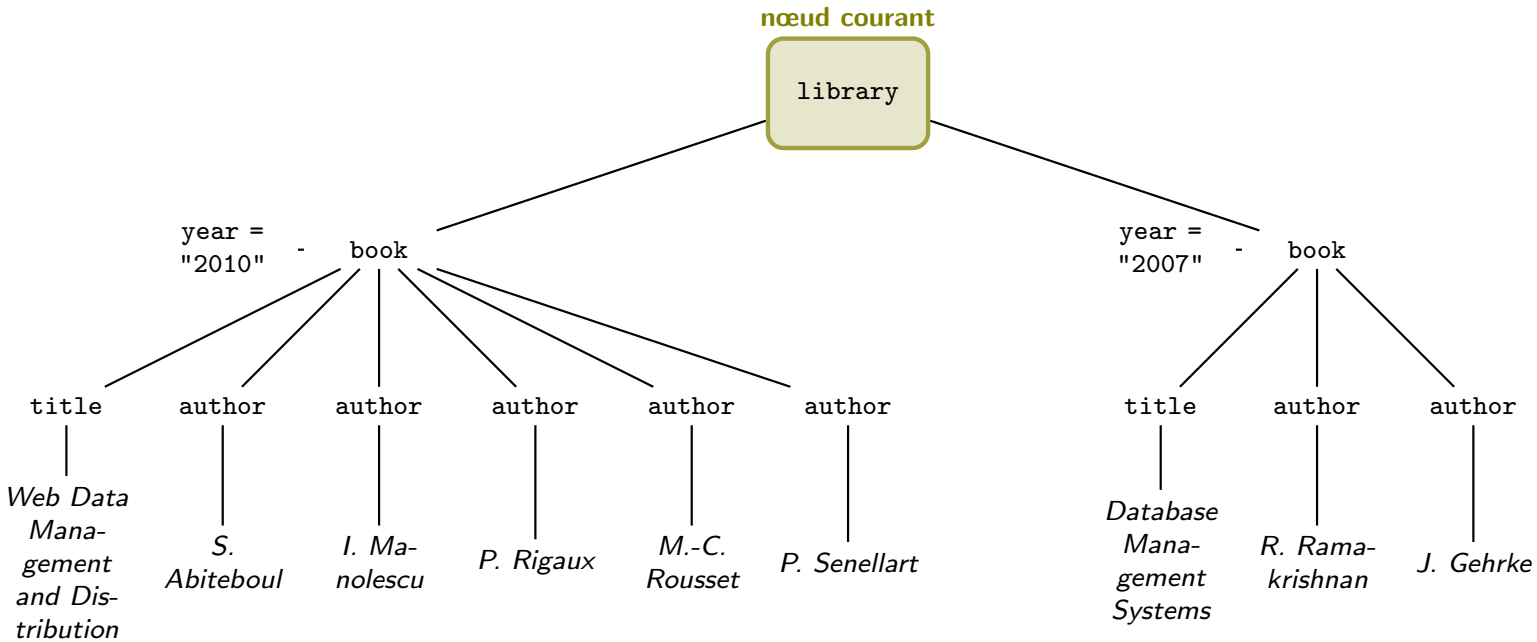
Quelques axes en exemple : Parent

`parent` : Selects the parent of the current node.



Quelques axes en exemple : Parent

`parent` : Selects the parent of the current node.

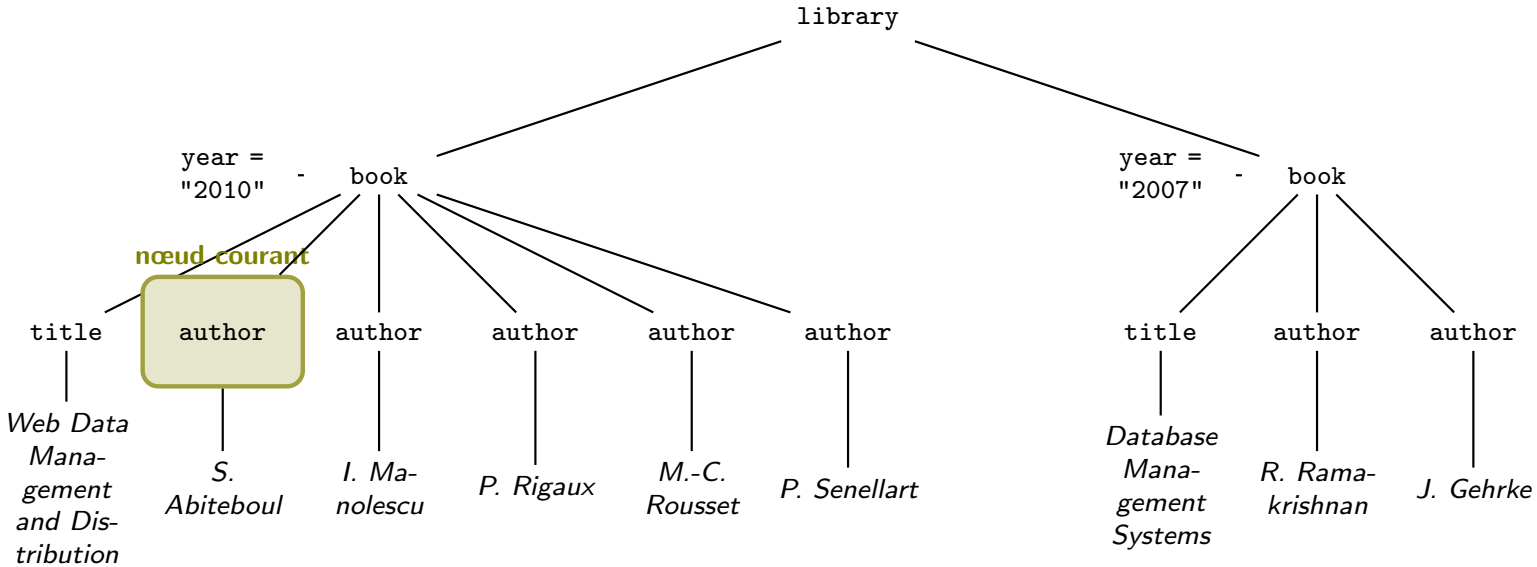


Plan du cours

1. XPath
   - Les chemins de localisation
   - Child
   - Parent
   - Ancestor
   - Ancestor-or-self
   - Descendant
   - Descendant-or-self
   - Following
   - Following-sibling
   - Preceding
   - Preceding-sibling
   - Attribute
   - Test de noeud
   - Les prédicats

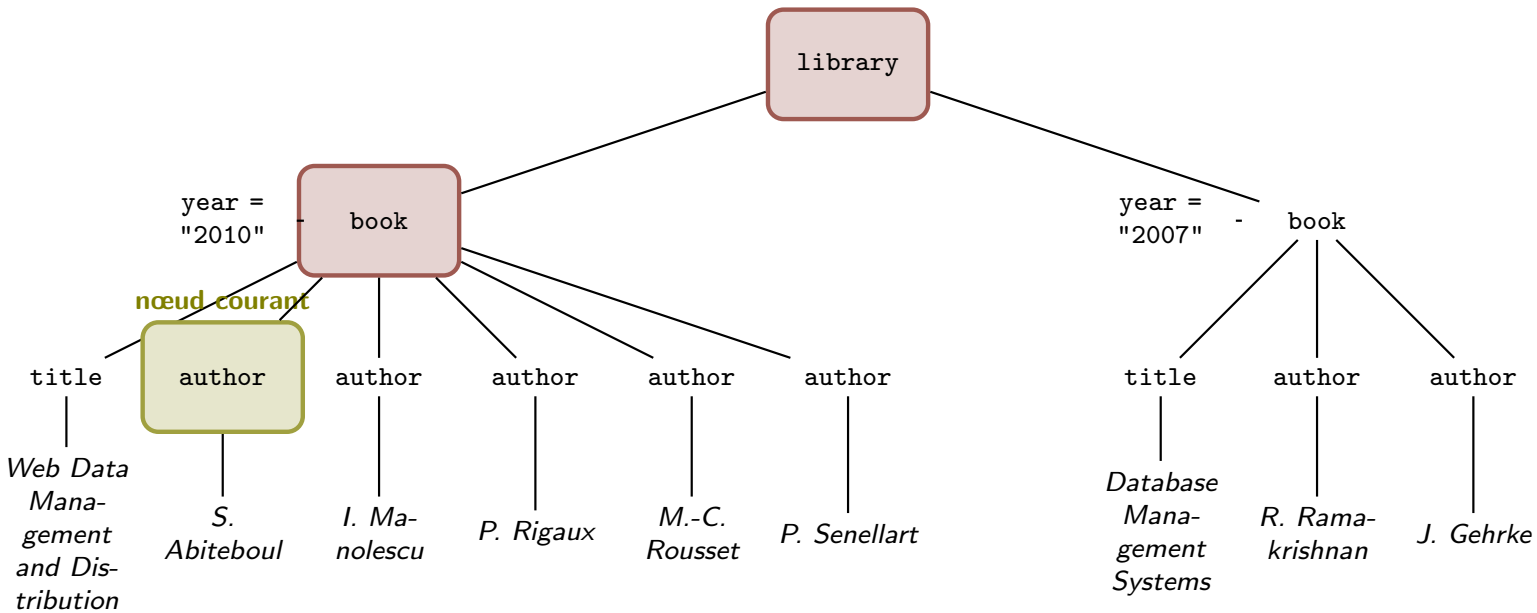
Quelques axes en exemple : Ancestor

`ancestor` : Selects all ancestors (parent, grandparent, etc.) of the current node



Quelques axes en exemple : Ancestor

`ancestor` : Selects all ancestors (parent, grandparent, etc.) of the current node

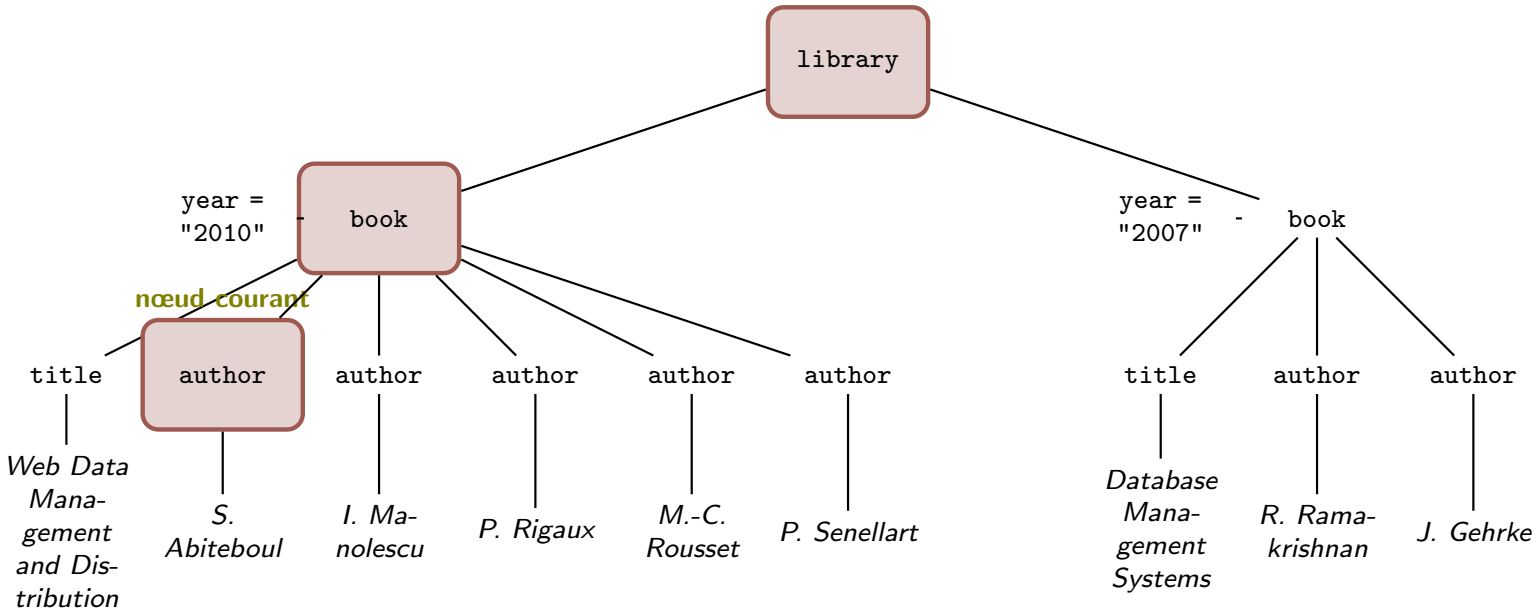


Plan du cours

1. XPath
   - Les chemins de localisation
   - Child
   - Parent
   - Ancestor
   - Ancestor-or-self
   - Descendant
   - Descendant-or-self
   - Following
   - Following-sibling
   - Preceding
   - Preceding-sibling
   - Attribute
   - Test de noeud
   - Les prédicats

Quelques axes en exemple : Ancestor-or-self

`ancestor-or-self` : Selects all ancestors (parent, grandparent, etc.) of the current node and the current node itself

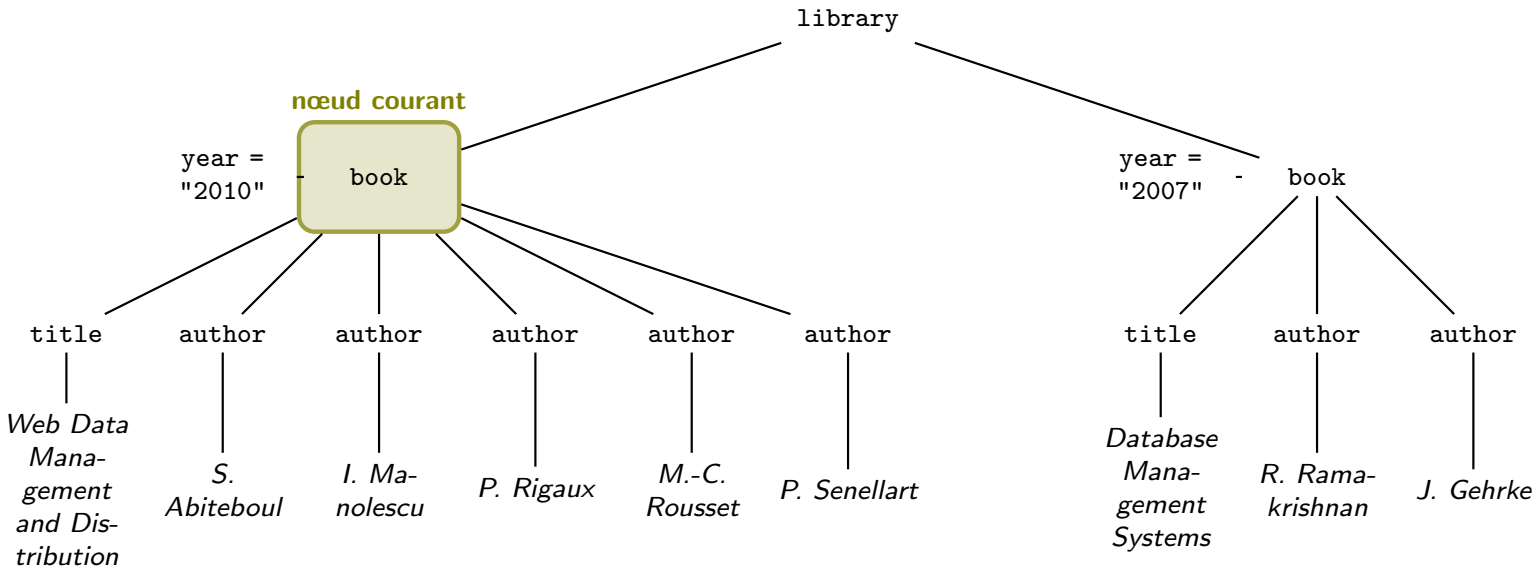


Plan du cours

1. XPath
   - Les chemins de localisation
   - Child
   - Parent
   - Ancestor
   - Ancestor-or-self
   - Descendant
   - Descendant-or-self
   - Following
   - Following-sibling
   - Preceding
   - Preceding-sibling
   - Attribute
   - Test de noeud
   - Les prédicats

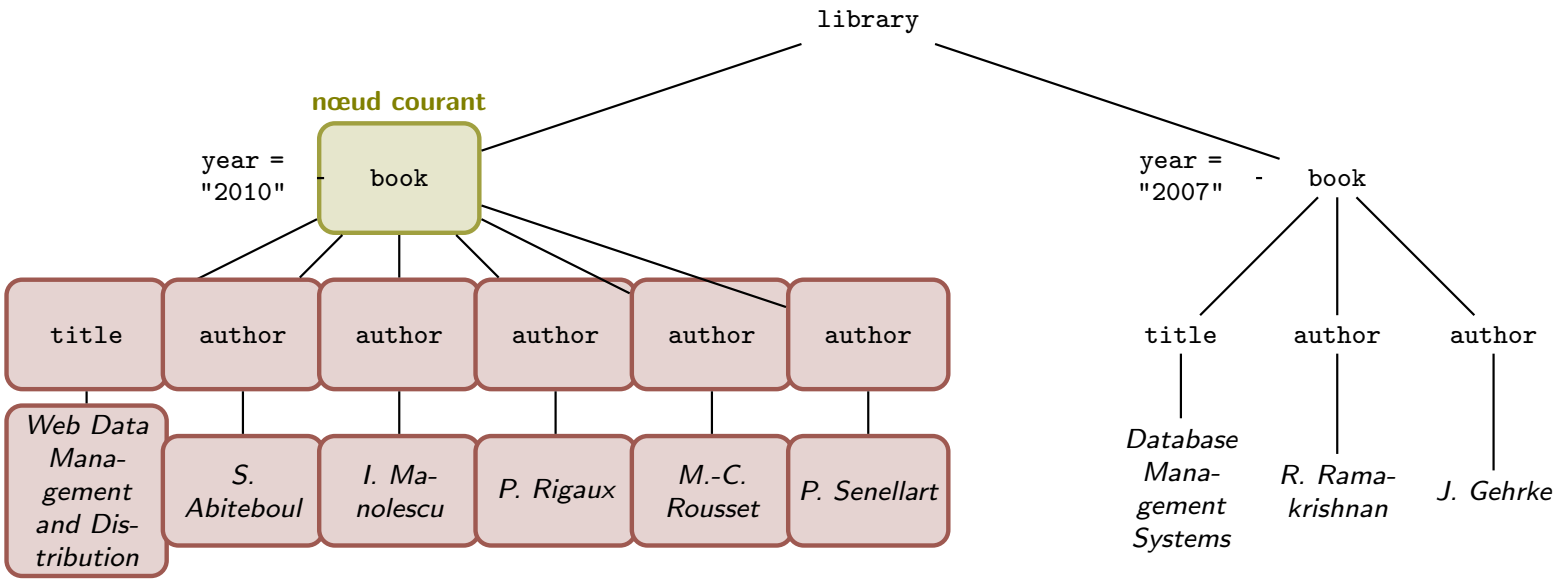
Quelques axes en exemple : Descendant

`descendant` : Selects all descendants (children, grandchildren, etc.) of the current node



Quelques axes en exemple : Descendant

`descendant` : Selects all descendants (children, grandchildren, etc.) of the current node

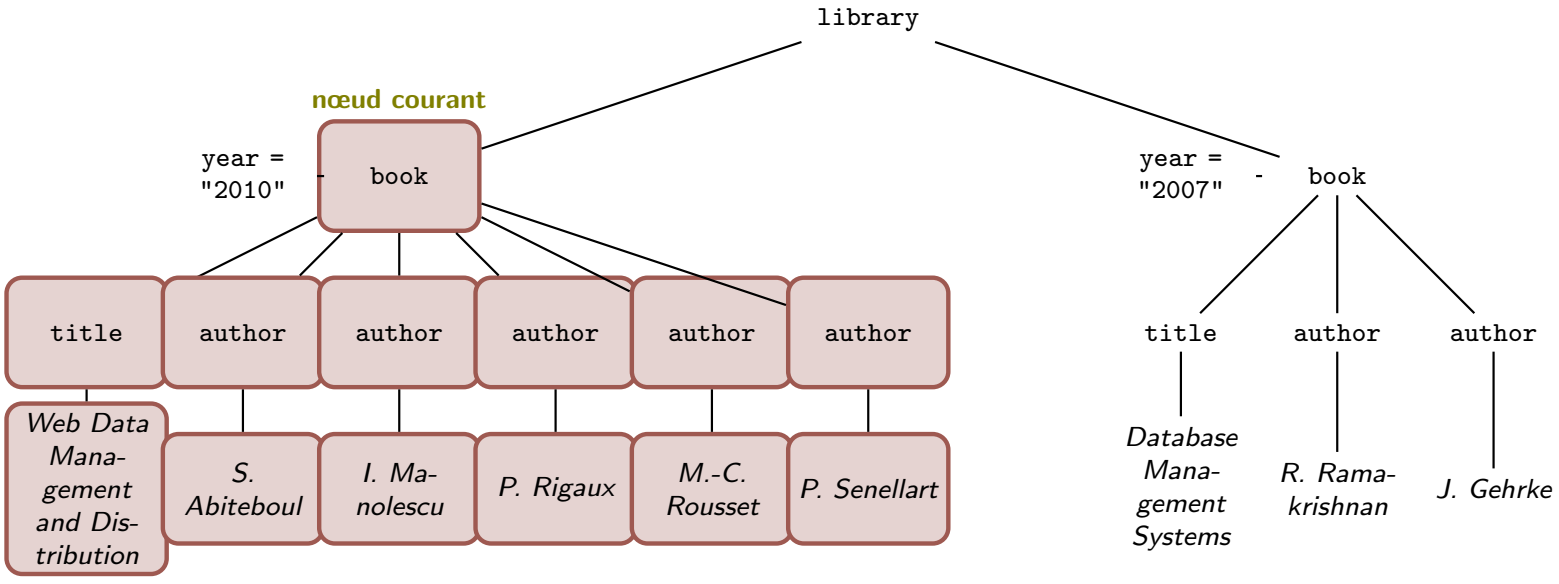


Plan du cours

1. XPath
   - Les chemins de localisation
   - Child
   - Parent
   - Ancestor
   - Ancestor-or-self
   - Descendant
   - Descendant-or-self
   - Following
   - Following-sibling
   - Preceding
   - Preceding-sibling
   - Attribute
   - Test de noeud
   - Les prédicats

Quelques axes en exemple : Descendant-or-self

`descendant-or-self` : Selects all descendants (children, grandchildren, etc.) of the current node and the current node itself

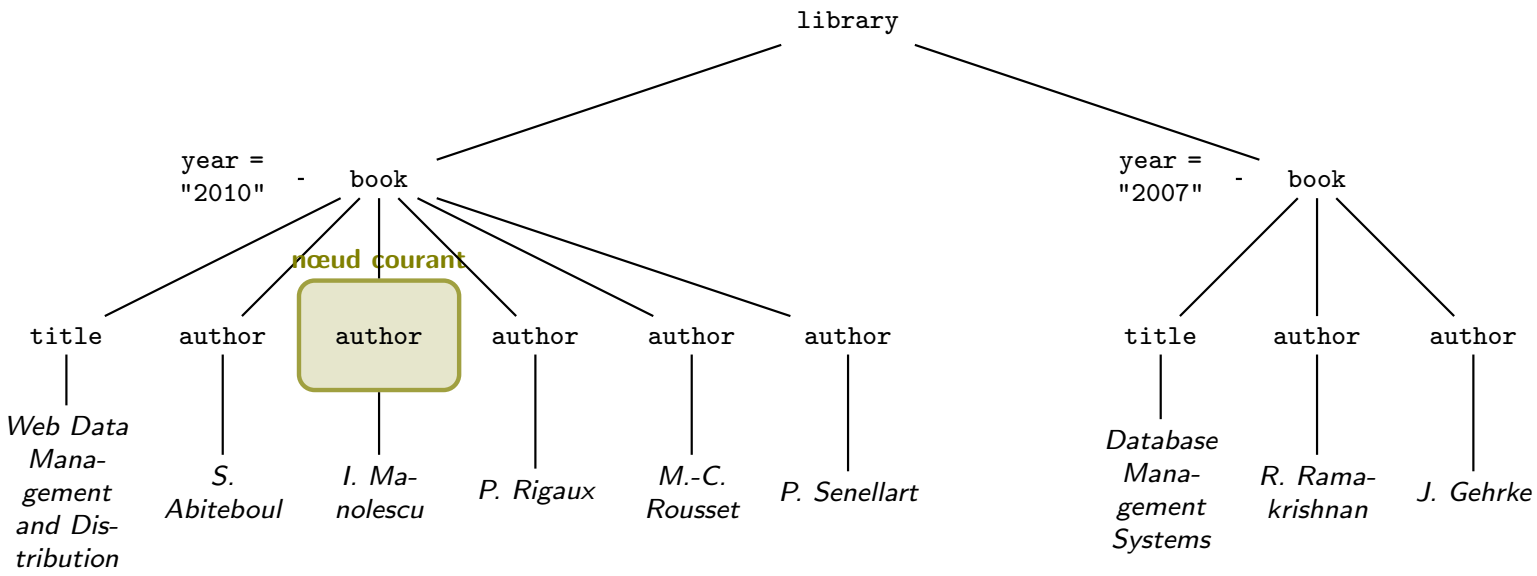


Plan du cours

1. XPath
   - Les chemins de localisation
   - Child
   - Parent
   - Ancestor
   - Ancestor-or-self
   - Descendant
   - Descendant-or-self
   - Following
   - Following-sibling
   - Preceding
   - Preceding-sibling
   - Attribute
   - Test de noeud
   - Les prédicats

Quelques axes en exemple : Following

`following` : Selects everything in the document defined after the closing tag of the current node



Quelques axes en exemple : Following

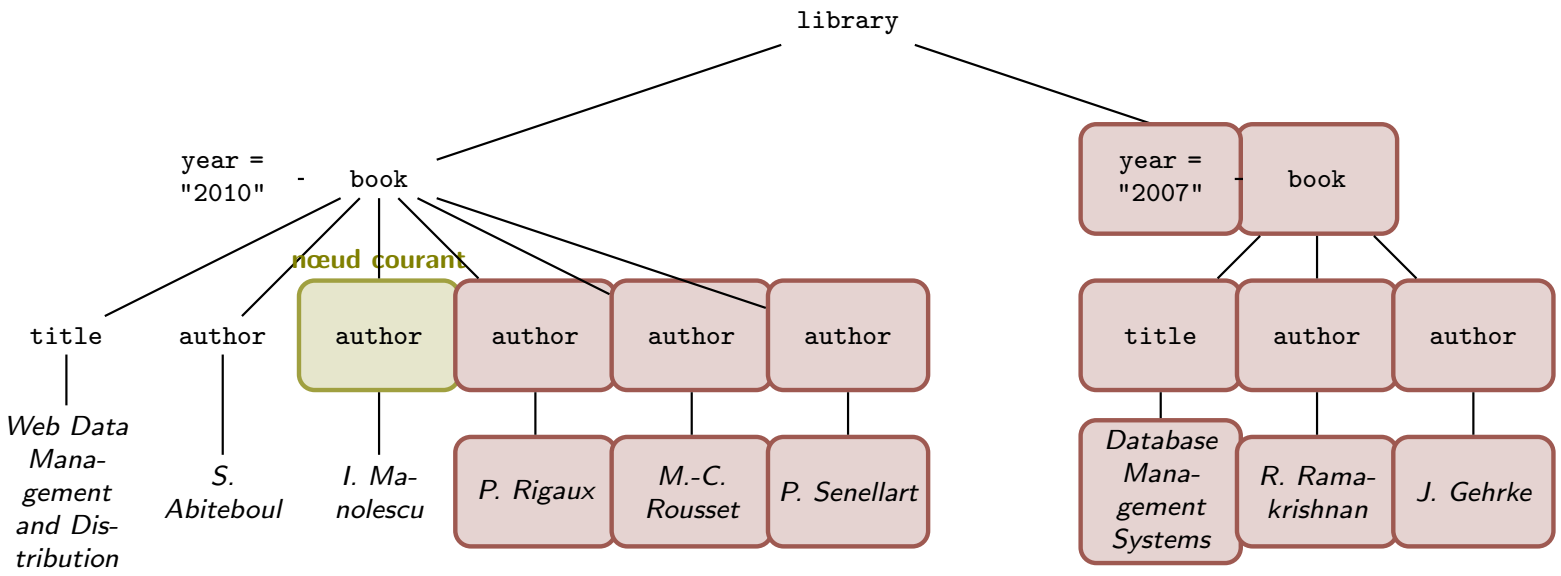
`following` : Selects everything in the document defined after the closing tag of the current node

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <library>
3    <book year="2010">
4      <title>Web Data Management and Distribution</title>
5      <author>S. Abiteboul</author>
6      <author>I. Manolescu</author>
7      <author>P. Rigaux</author>
8      <author>M.-C. Rousset</author>
9      <author>P. Senellart</author>
10   </book>
11   <book year="2007">
12     <title>Database Management Systems</title>
13     <author>R. Ramakrishnan</author>
14     <author>J. Gehrke</author>
15   </book>
16 </library>
```

Listing 2 – Fichier twobooks.xml

Quelques axes en exemple : Following

`following` : Selects everything in the document defined after the closing tag of the current node

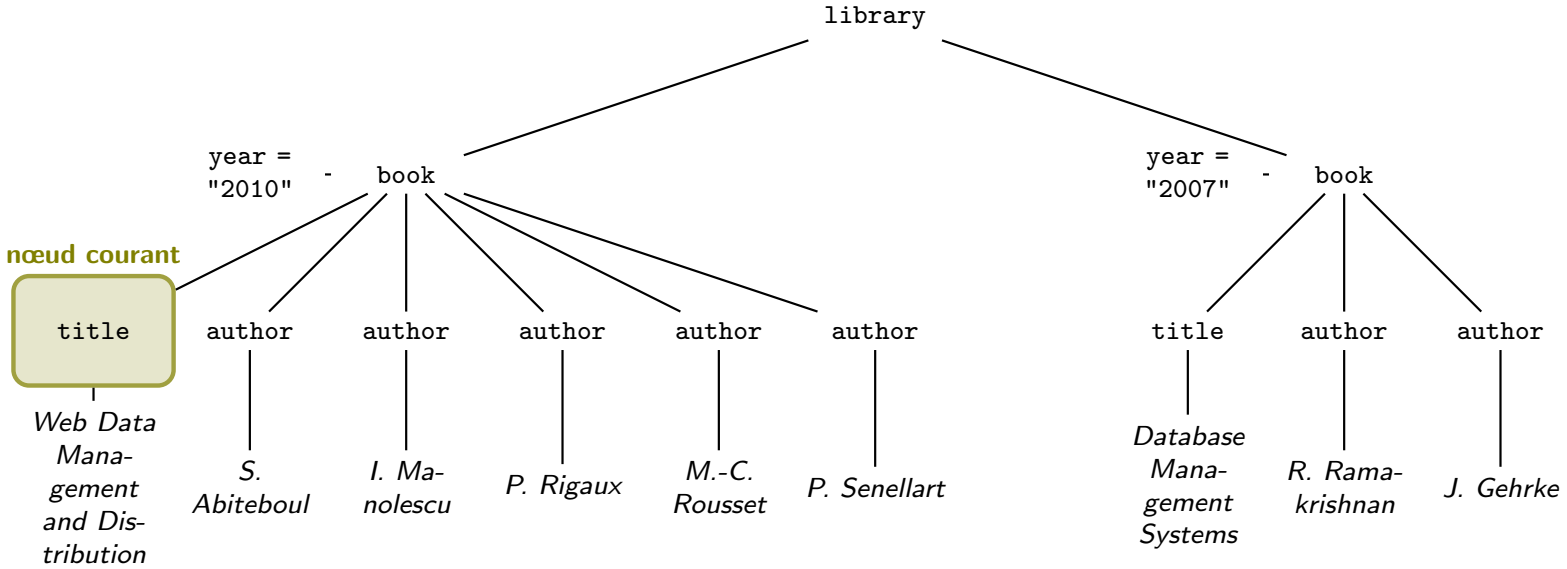


Plan du cours

1. XPath
   - Les chemins de localisation
   - Child
   - Parent
   - Ancestor
   - Ancestor-or-self
   - Descendant
   - Descendant-or-self
   - Following
   - Following-sibling
   - Preceding
   - Preceding-sibling
   - Attribute
   - Test de noeud
   - Les prédicats

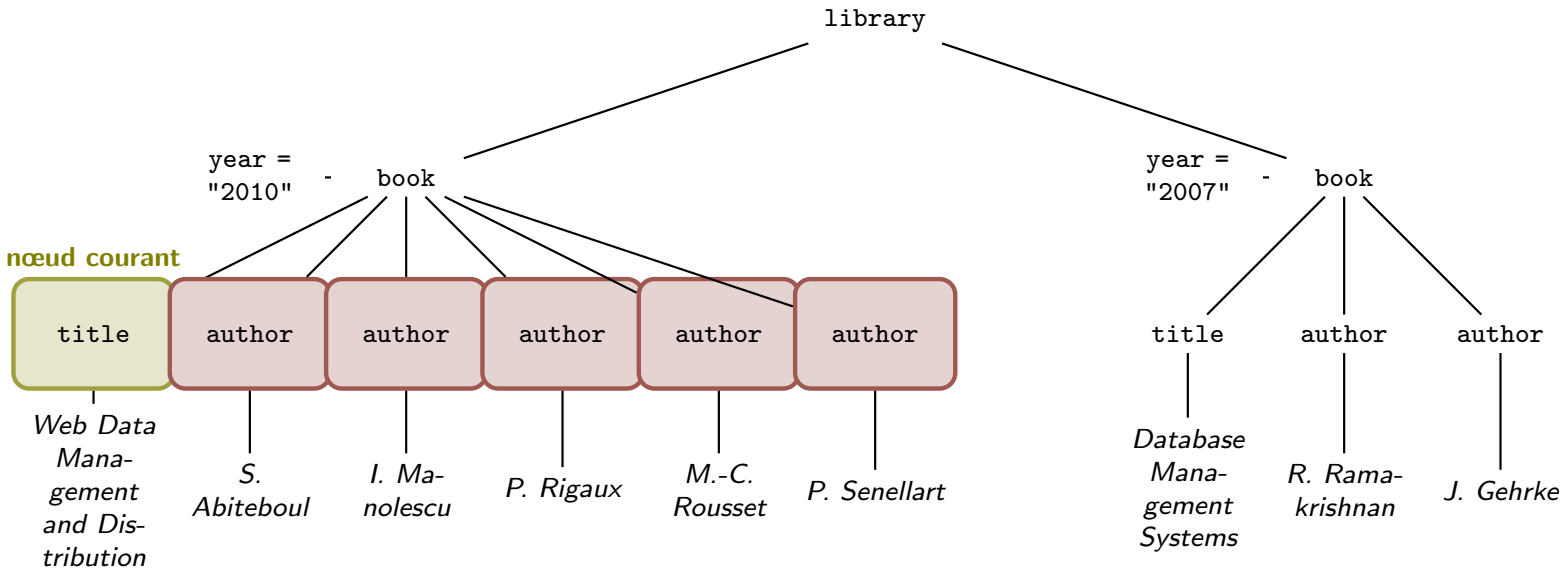
Quelques axes en exemple : Following-sibling

`following-sibling` : Selects all siblings after the current node.



Quelques axes en exemple : Following-sibling

`following-sibling` : Selects all siblings after the current node

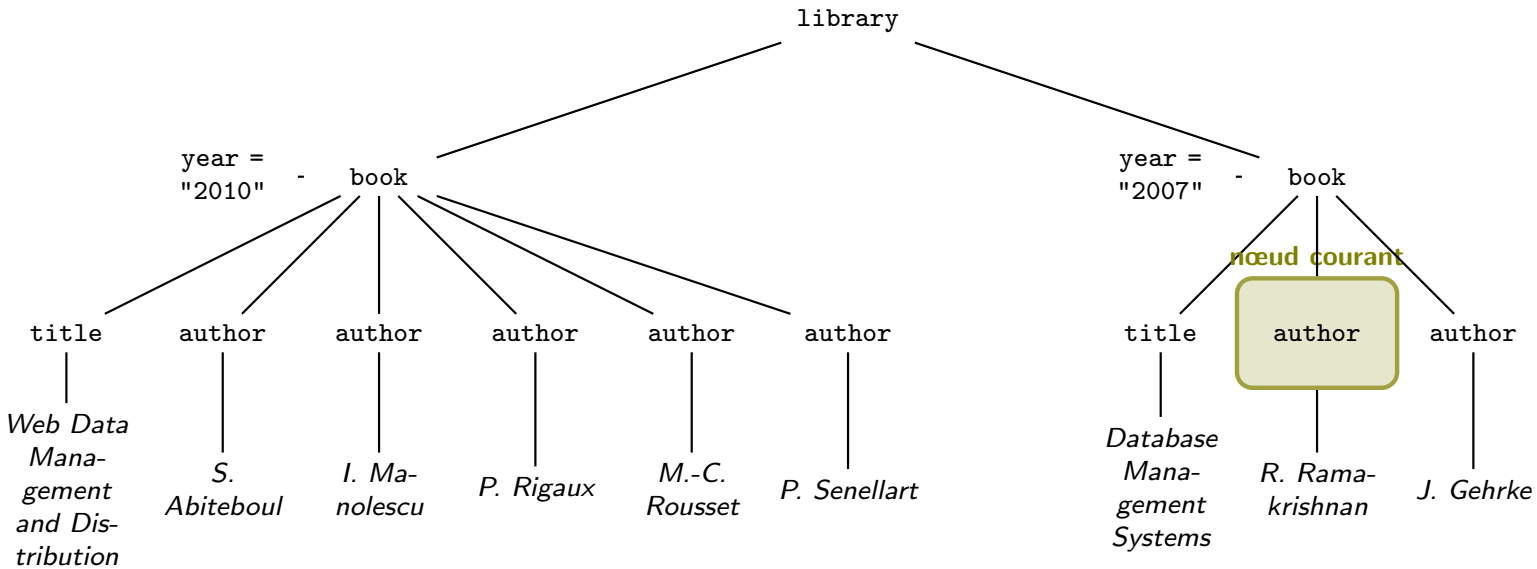


Plan du cours

1. XPath
   - Les chemins de localisation
   - Child
   - Parent
   - Ancestor
   - Ancestor-or-self
   - Descendant
   - Descendant-or-self
   - Following
   - Following-sibling
   - Preceding
   - Preceding-sibling
   - Attribute
   - Test de noeud
   - Les prédicats

Quelques axes en exemple : Following

`preceding` : Selects everything in the document that is defined before the start tag of the current node



Quelques axes en exemple : Following

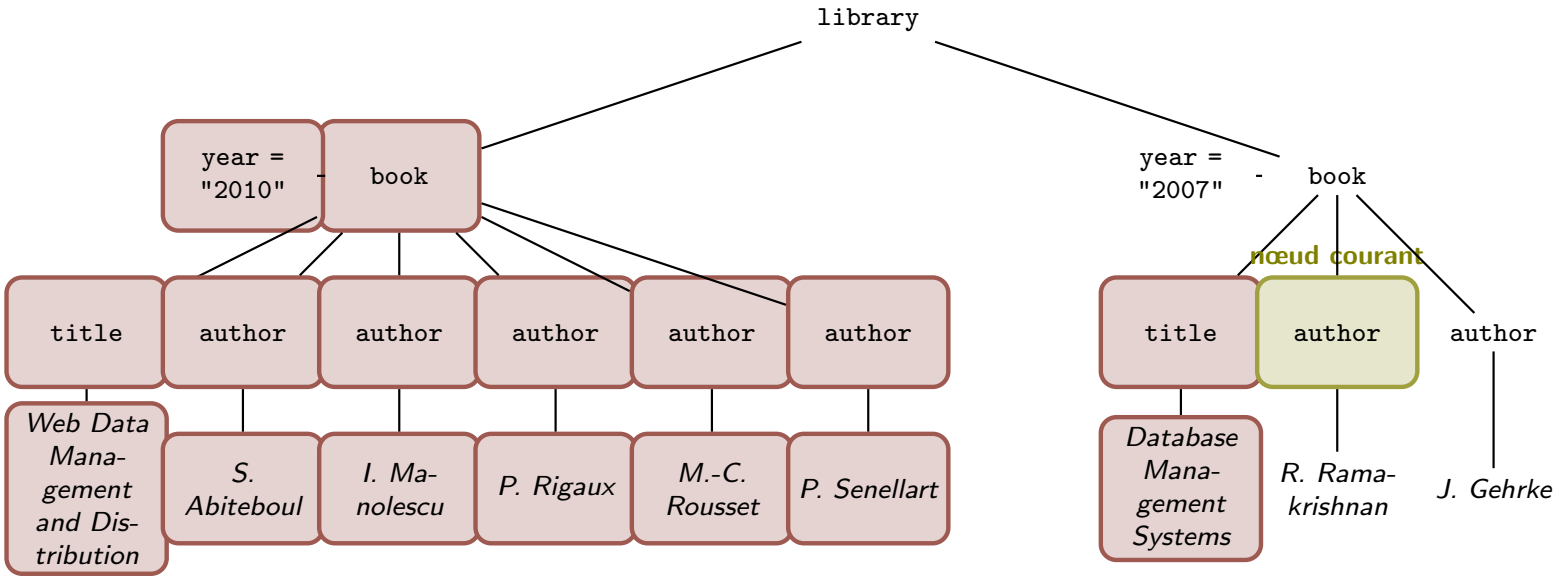
`preceding` : Selects everything in the document that is defined before the start tag of the current node

```
1  <?xml version="1.0" encoding="ISO-8859-1"?>
2  <library>
3    <book year="2010">
4      <title>Web Data Management and Distribution</title>
5      <author>S. Abiteboul</author>
6      <author>I. Manolescu</author>
7      <author>P. Rigaux</author>
8      <author>M.-C. Rousset</author>
9      <author>P. Senellart</author>
10   </book>
11   <book year="2007">
12     <title>Database Management Systems</title>
13     <author>R. Ramakrishnan</author>
14     <author>J. Gehrke</author>
15   </book>
16 </library>
```

Listing 3 – Fichier twobooks.xml

Quelques axes en exemple : Following

`preceding` : Selects everything in the document that is defined before the start tag of the current node

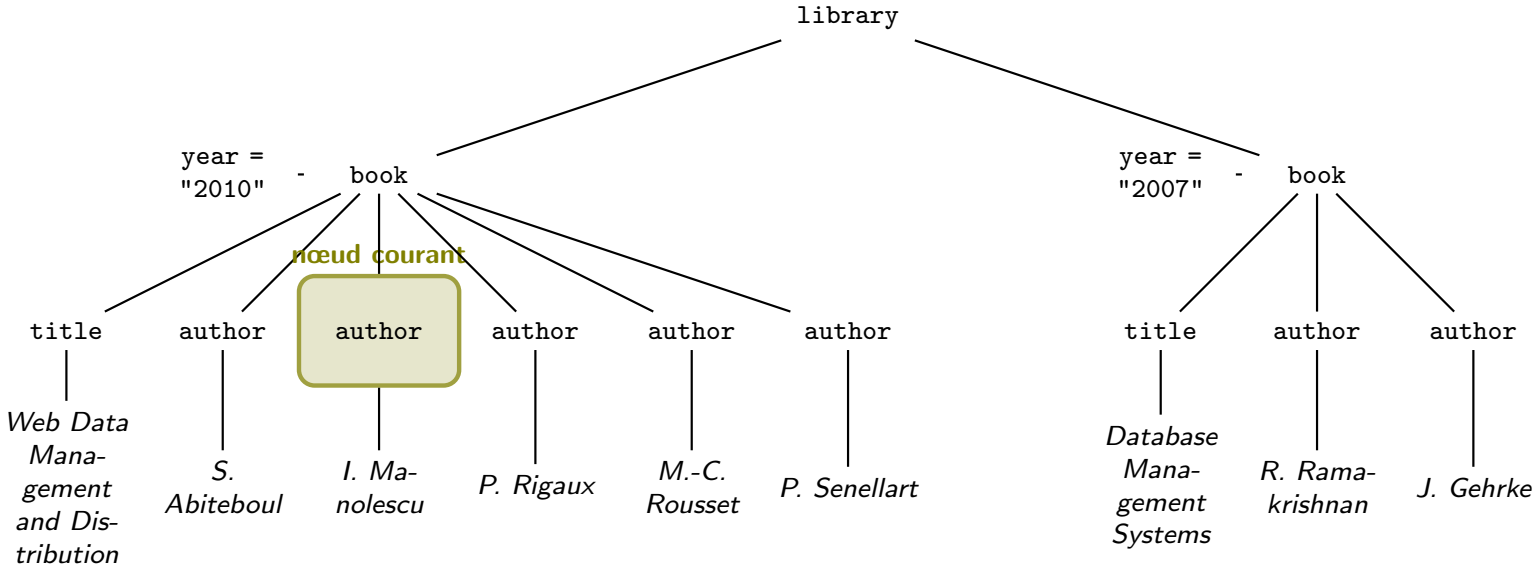


Plan du cours

1. XPath
   - Les chemins de localisation
   - Child
   - Parent
   - Ancestor
   - Ancestor-or-self
   - Descendant
   - Descendant-or-self
   - Following
   - Following-sibling
   - Preceding
   - Preceding-sibling
   - Attribute
   - Test de noeud
   - Les prédicats

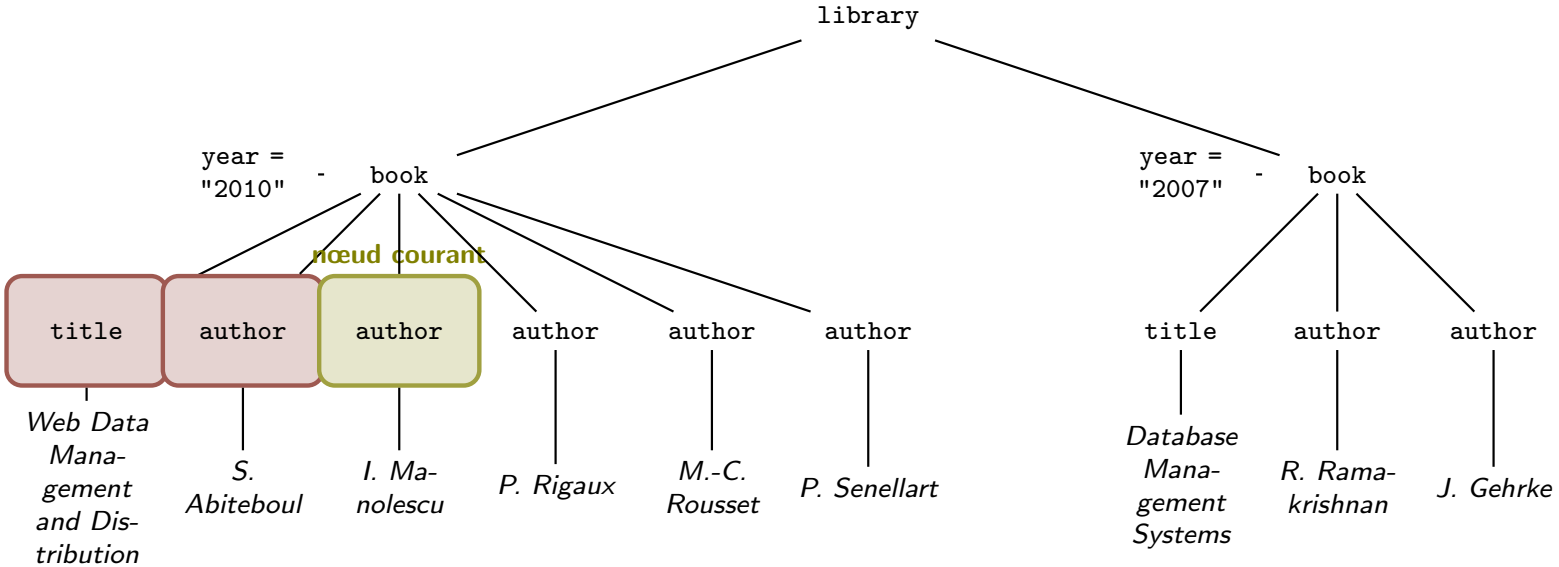
Quelques axes en exemple : Following-sibling

`preceding-sibling` : Selects all siblings before the current node.



Quelques axes en exemple : Following-sibling

`preceding-sibling` : Selects all siblings before the current node.

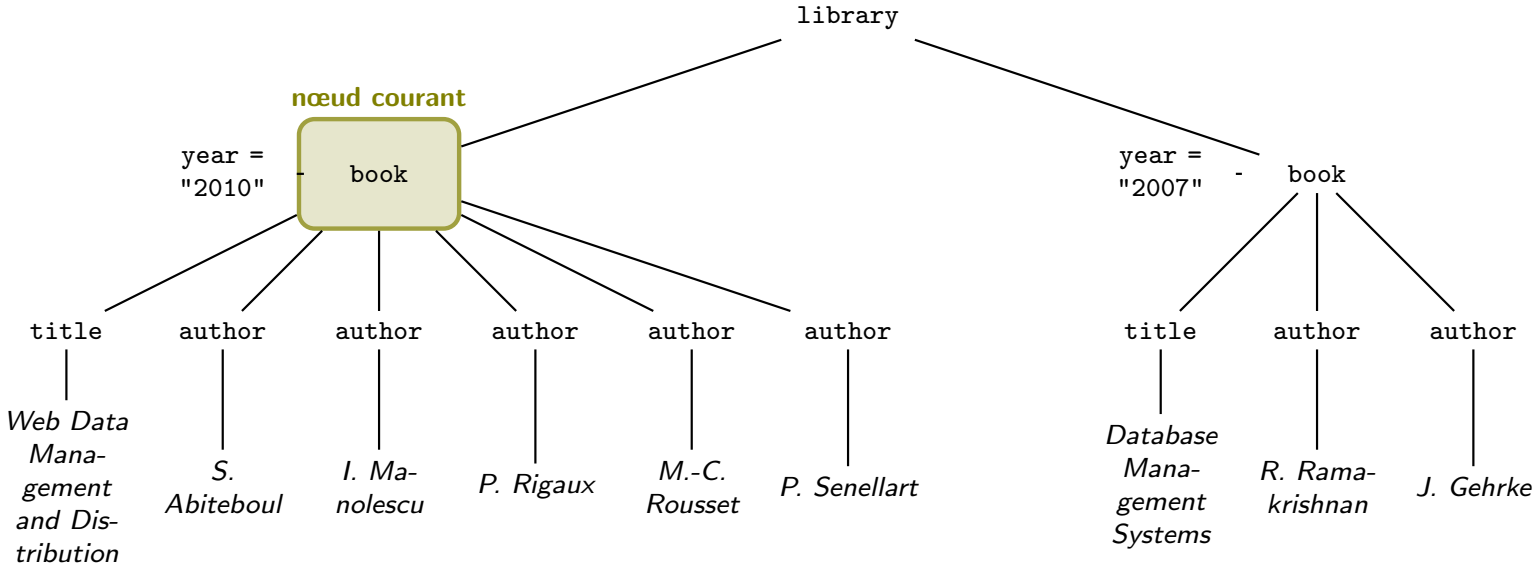


Plan du cours

1. XPath
   - Les chemins de localisation
   - Child
   - Parent
   - Ancestor
   - Ancestor-or-self
   - Descendant
   - Descendant-or-self
   - Following
   - Following-sibling
   - Preceding
   - Preceding-sibling
   - Attribute
   - Test de noeud
   - Les prédicats

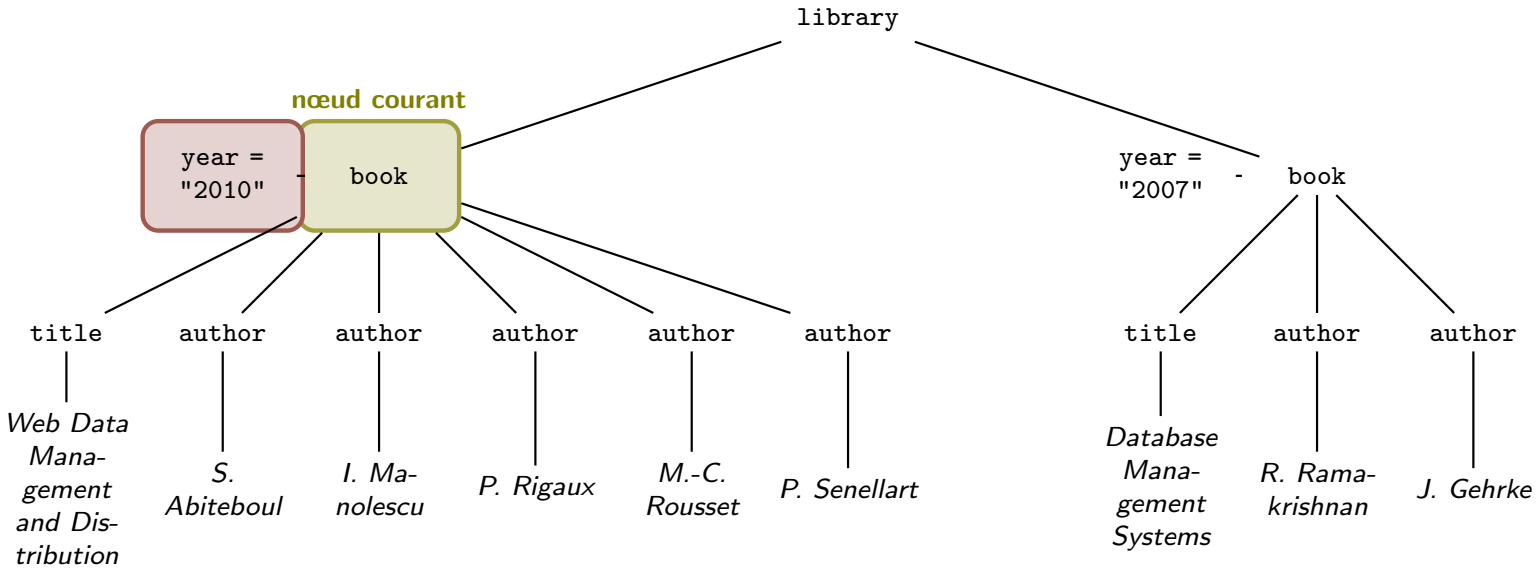
Quelques axes en exemple : Attribute

`attribute` : Selects all attributes of the current node



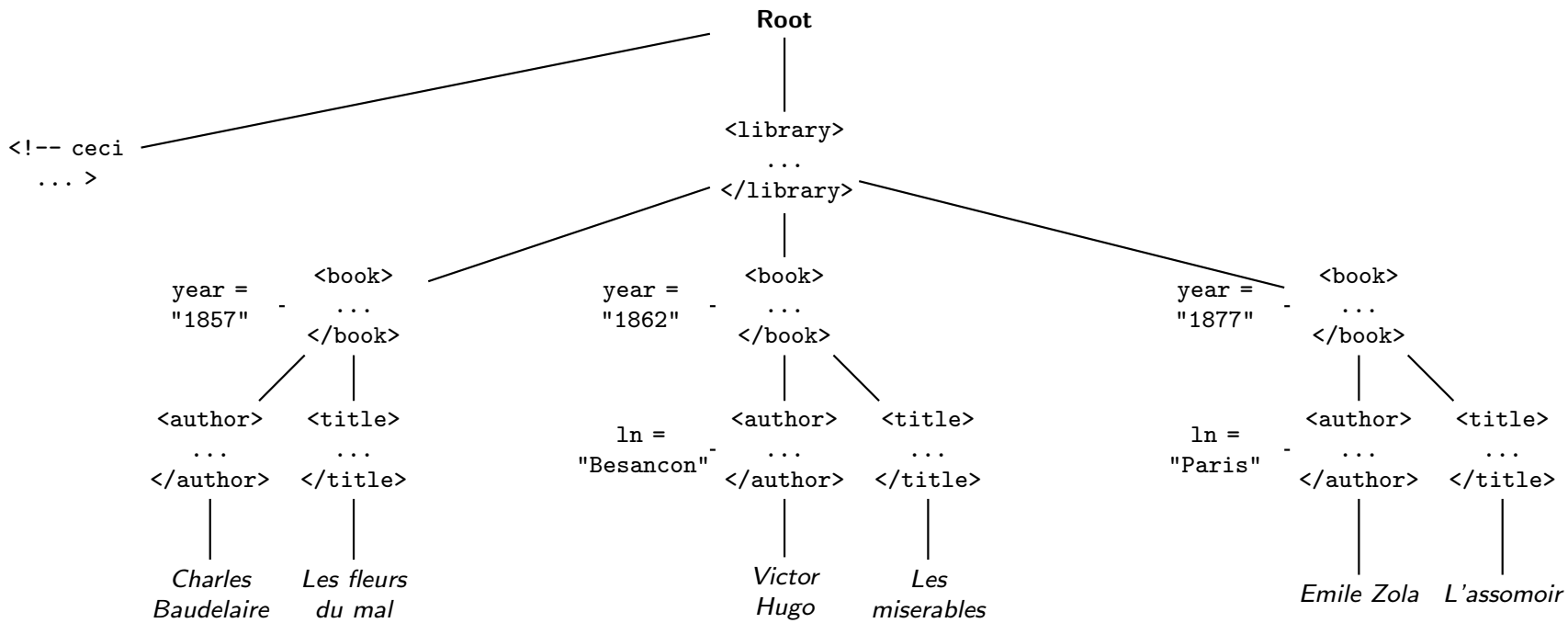
Quelques axes en exemple : Attribute

`attribute` : Selects all attributes of the current node



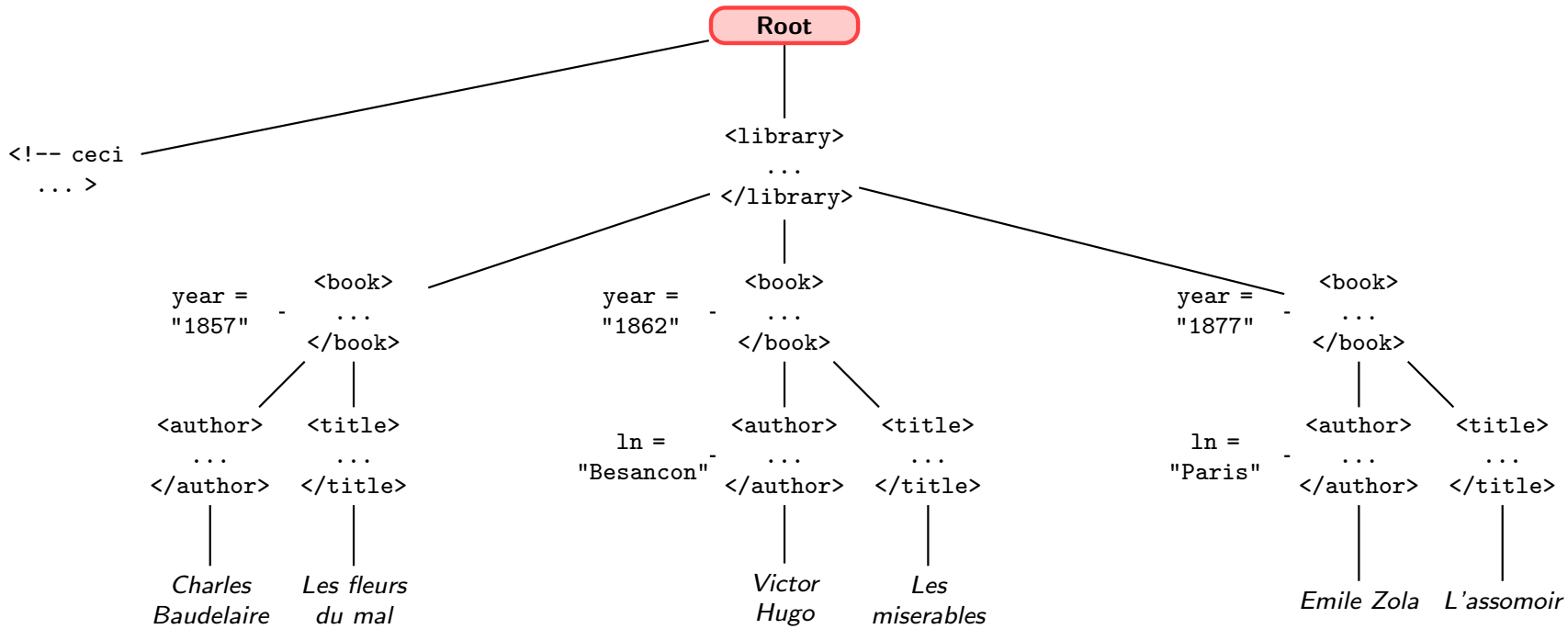
Exemple

```
/child::library/child::*/attribute::year
```



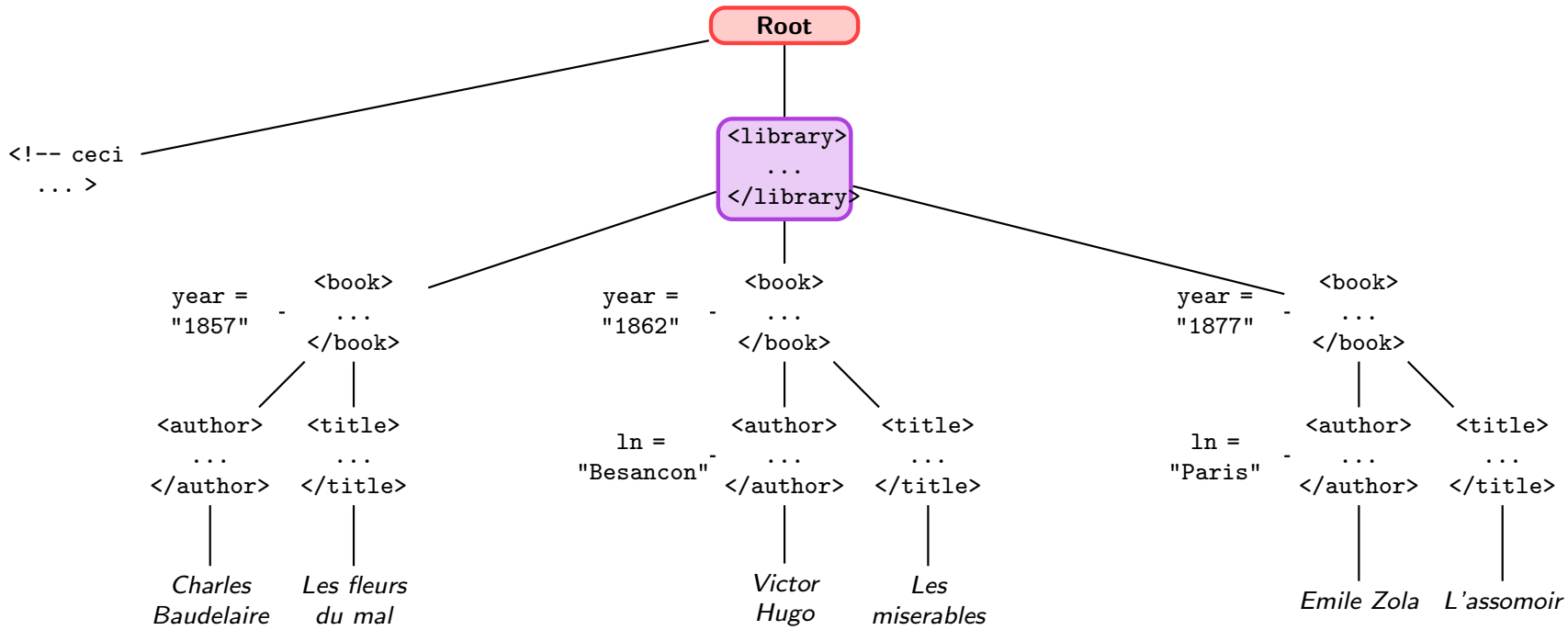
Exemple (suite)

```
/child::library/child::*/attribute::year
```



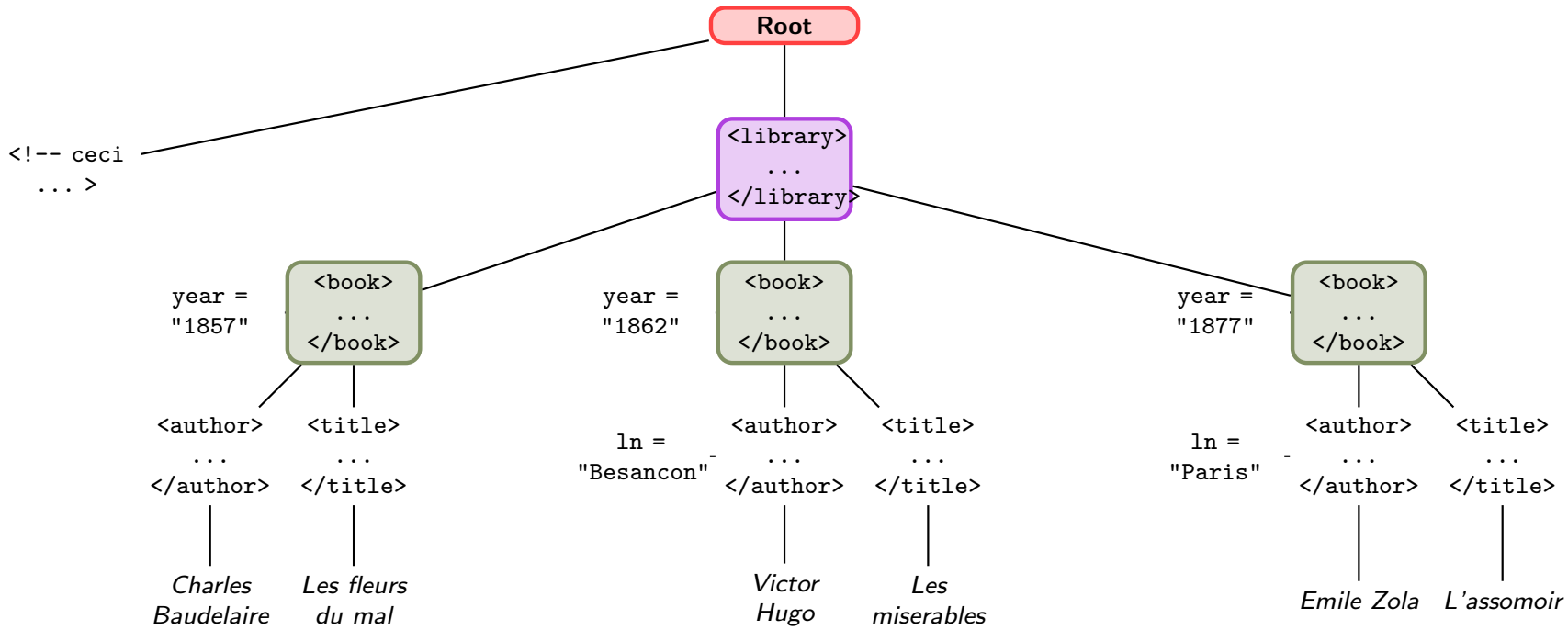
Exemple (suite)

/child::library/child::*/attribute::year



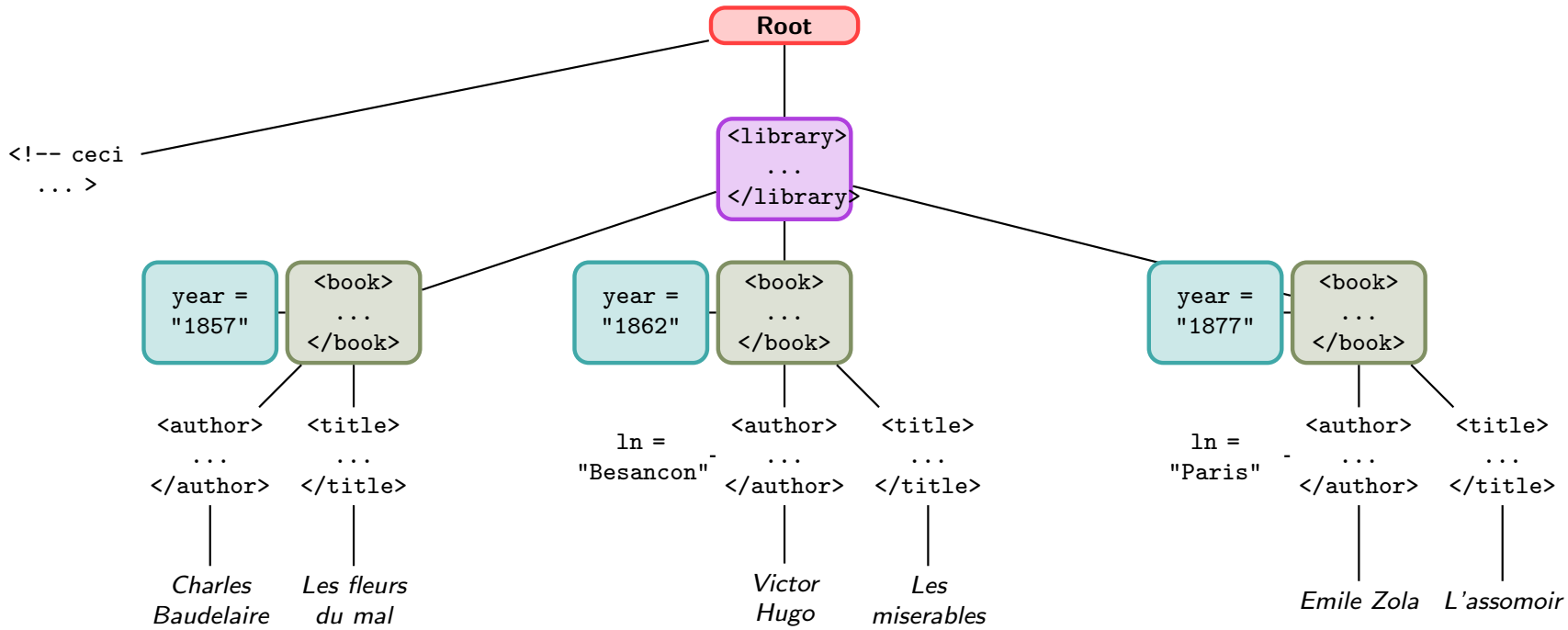
Exemple (suite)

/child::library/child::*/attribute::year



Exemple (suite)

/child::library/child::*/attribute::year



Plan du cours

1. XPath
   - Les chemins de localisation
   - Child
   - Parent
   - Ancestor
   - Ancestor-or-self
   - Descendant
   - Descendant-or-self
   - Following
   - Following-sibling
   - Preceding
   - Preceding-sibling
   - Attribute
   - Test de noeud
   - Les prédicats

Les tests de nœuds

Test du type du nœud :

- `node()` sélectionne les nœuds quelque soit leur "type" (élément, nœud texte, processing-instruction)
- `text()` sélectionne les nœuds de "type" texte
- `comment()` sélectionne les nœuds de "type" commentaire
- `processing-instruction()` sélectionne les nœuds de "type" proc-instruction

Test de nom :

- `name` sélectionne les éléments ou attributs de noms `name`
- `*` A node test * is true for any node of the *principal node type*. For example, child : :* will select all element children of the context node, and attribute : :* will select all attributes of the context node.

Quelques axes en exemple : Child

Revenons sur `Child`.

`child::para`	selects the para element children of the context node
`child::*`	selects all element children of the context node
`child::text()`	selects all text node children of the context node
`child::node()`	selects all the children of the context node, whatever their node type

Mise en jambe

Test

Que "cherchent" à calculer les expressions suivantes ? Donner les résultats de leurs évaluations sur le fichier *twobooks.xml*.

```
/child::library
/child::*/child::book/child::title/child::text()
/child::library/child::book()/attribute::*
/child::library/child::book/attribute::year
/descendant::author
/child::*/descendant::library
/descendant::book/child::title
```

Plan du cours

1. XPath
   - Les chemins de localisation
   - Child
   - Parent
   - Ancestor
   - Ancestor-or-self
   - Descendant
   - Descendant-or-self
   - Following
   - Following-sibling
   - Preceding
   - Preceding-sibling
   - Attribute
   - Test de noeud
   - Les prédicats

Les prédicats

Un prédicat est une condition filtrant un ensemble contexte (ne garde que les nœuds (sous-arbres) satisfaisant la condition).

Contraintes booléennes (combinaison logique (and, or, not) de contraintes plus simples) :

- Comparaisons
  `child::biblio/child::livre[attribute::ISBN='0262-01210-3']/child::authors`
- Existence d'un attribut ou d'un élément
  `document[child::date]`
  `personne[attribute::date_naissance]`

Exemples (W3C) :

```
child::employee[child::secretary or child::assistant]
```

```
descendant::toy[not(attribute::color = "red") and
attribute::country= "China"]
```

Evaluation des prédicats

Une petite subtilité...

- Si l'expression est une "véritable" expression booléenne, pas de soucis
  `/inventory/drink/lemonade[child::amount>15]`
- Si le résultat du prédicat ou si le prédicat est un nombre, le résultat du prédicat est converti à true si le nombre est égal à la position dans le contexte d'évaluation (sinon false)
  Ainsi, `para[3]` est équivalent à `para[position()=3]`.

Zoom sur les prédicats utilisant la fonction `position()`

```
<library>
  <book year="2010">
    <title>Web Data Management and Distribution</title>
    <author>S. Abiteboul</author>
    <author>I. Manolescu</author>
    <author>P. Rigaux</author>
    <author>M.-C. Rousset</author>
    <author>P. Senellart</author>
  </book>
  <book year="2007">
    <title>Database Management Systems</title>
    <author>R. Ramakrishnan</author>
    <author>J. Gehrke</author>
  </book>
</library>
```

Listing 4 – Extrait de twobooks.xml

Example

```
/child::library/child::book/child::author[position()=1]
```

Test

Quel est le résultat de l'évaluation de cette requête que le fichier *twobooks.xml* ?

Zoom sur les prédicats utilisant la fonction `position()`

```
<library>
  <book year="2010">
    <title>Web Data Management and Distribution</title>
    <author>S. Abiteboul</author>
    <author>I. Manolescu</author>
    <author>P. Rigaux</author>
    <author>M.-C. Rousset</author>
    <author>P. Senellart</author>
  </book>
  <book year="2007">
    <title>Database Management Systems</title>
    <author>R. Ramakrishnan</author>
    <author>J. Gehrke</author>
  </book>
</library>
```

Listing 5 – Extrait de twobooks.xml

Example

```
/child::library/child::book/child::author[position()=last()]
```

Test

Quel est le résultat de l'évaluation de cette requête que le fichier *twobooks.xml* ?

Zoom sur les prédicats utilisant la fonction `position()`

```
<library>
  <book year="2010">
    <title>Web Data Management and Distribution</title>
    <author>S. Abiteboul</author>
    <author>I. Manolescu</author>
    <author>P. Rigaux</author>
    <author>M.-C. Rousset</author>
    <author>P. Senellart</author>
  </book>
  <book year="2007">
    <title>Database Management Systems</title>
    <author>R. Ramakrishnan</author>
    <author>J. Gehrke</author>
  </book>
</library>
```

Listing 6 – Extrait de twobooks.xml

Example

```
/child::library/child::book/child::author[position()=3]
```

Test

Quel est le résultat de l'évaluation de cette requête que le fichier *twobooks.xml* ?

Zoom sur les prédicats utilisant la fonction `position()`

```
<library>
  <book year="2010">
    <title>Web Data Management and Distribution</title>
    <author>S. Abiteboul</author>
    <author>I. Manolescu</author>
    <author>P. Rigaux</author>
    <author>M.-C. Rousset</author>
    <author>P. Senellart</author>
  </book>
  <book year="2007">
    <title>Database Management Systems</title>
    <author>R. Ramakrishnan</author>
    <author>J. Gehrke</author>
  </book>
</library>
```

Listing 7 – Extrait de twobooks.xml

Example

```
/child::library/child::book[position()=2]/child::author
```

Test

Quel est le résultat de l'évaluation de cette requête que le fichier *twobooks.xml* ?

Zoom sur les prédicats utilisant la fonction `position()`

```
<library>
  <book year="2010">
    <title>Web Data Management and Distribution</title>
    <author>S. Abiteboul</author>
    <author>I. Manolescu</author>
    <author>P. Rigaux</author>
    <author>M.-C. Rousset</author>
    <author>P. Senellart</author>
  </book>
  <book year="2007">
    <title>Database Management Systems</title>
    <author>R. Ramakrishnan</author>
    <author>J. Gehrke</author>
  </book>
</library>
```

Listing 8 – Extrait de twobooks.xml

Example

```
/child::library/child::book/child::*[position()=1]
```

Test

Quel est le résultat de l'évaluation de cette requête que le fichier *twobooks.xml* ?

Abréviations

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<inventory>
  <drink>
    <lemonade supplier="mother" id="1">
      <price>$2.50</price>
      <amount>20</amount>
    </lemonade>
    <pop supplier="store" id="2">
      <price>$1.50</price>
      <amount>10</amount>
    </pop>
  </drink>
  <snack>
    <chips supplier="store" id="3">
      <price>$4.50</price>
      <amount>60</amount>
      <calories>180</calories>
    </chips>
  </snack>
</inventory>
```

Listing 9 – Fichier limonade.xml

Abréviations

`child` peut être omis dans le chemin de localisation

```
/inventory/drink/lemonade[child::amount>15]
```

`//a` est une forme raccourcie de `/descendant-or-self::node()/child::a`

```
//lemonade[child::amount>15]
/inventory//lemonade[child::amount>15]
```

`a//b` est un raccourci pour `a/descendant-or-self::node()/child::b`

`.` est un raccourci pour `self::node()`

```
./inventory//lemonade[child::amount>15]
```

`..` est un raccourci pour `parent::node()`

```
/inventory/drink/lemonade[child::amount>15]/../pop
```

`@attr_name` est un raccourci pour `attribute::attr_name`

```
/inventory/drink/lemonade[@supplier='store']
```

`[1]` est un raccourci pour `[position()=1]`

```
/inventory/drink[2]
```

Opérations et fonctions

- XPath permet aussi d'effectuer des calculs des nombres, des booléens et des chaînes de caractères.
- Opérations sur les nombres : `+`, `-` , `*`, `div`, `mod`
  Renvoie `NaN` si l'un des arguments n'est pas un nombre.
- Opérations sur les booléeans : `and`, `or`
- Égalité et différence (chaînes de caractères, nombres et booléens) `=`, `!=`
- Autres comparaisons (nombres seulement) : `<`, `<=`, `>`, `>=`

Opérations et fonctions (suite)

Fonctions sur les ensembles de nœuds

- `count(nset)` nombre d'éléments dans l'ensemble de nœuds.
- `name(nset)` nom du premier élément avec le préfixe de l'espace de noms
- `localname(nset)` nom du premier élément sans le préfixe de l'espace de noms
- `namespace-uri(nset)` préfixe de l'espace de noms du premier élément
- `sum(nset)` somme des arguments de l'ensemble de nœuds (après conversion des chaînes de caractères en nombres).
- `position()` position dans le context
- `last()` nombre correspondant à la dernière position dans le context
- `id(mon_id)` renvoi l'élément du document ayant pour identifiant `mon_id` (cherché parmis tous les attributs déclarés de type ID).

```
//book[count(auteur)>=1]
/biblio/book[position()=last()]
```

le cnam

Opérations et fonctions (suite)

Fonctions sur les chaînes de caractères

- `concat(ch1, ..., chn)`, `starts-with(ch1,ch2)`, `contains(ch1,ch2)`, `substring-before(ch1,ch2)`, `substring-after(ch1,ch2)`, `substring(ch,i,l)`, `string-length(ch)`, `normalize-space(ch)`, `translate(ch1,ch2,ch3)`

```
//titre[contains(text(),"Star")]
```

Voir `http://www.w3.org/TR/xpath#section-Function-Calls`

Opérations et fonctions (suite)

Fonctions sur les booléens

- `not(bool)`

```
not(@attr = 1)
not(not(@attr = 1) or false()) and true()
//book[not(count(auteur)>=1)]
```

Opérations et fonctions (suite)

Fonctions sur les nombres

- `floor(n)` arrondi à l'entier supérieur de `n`
- `ceiling(n)` arrondi à l'entier inférieur de `n`
- `round(n)` arrondi entier de `n` (plus proche de `floor(n)` ou `ceiling(n)`)

Voir `http://www.w3.org/TR/xpath#section-Function-Calls`

Zoom sur les espaces de noms

```
<html xmlns="http://www.w3.org/1999/xhtml"> </html>
```

Deux fonctions :
local-name
namespace-uri

Utilisation (exemples) :

```
//*[local-name()='html']
//*[namespace-uri()='http://www.w3.org/1999/xhtml'
//*[local-name()='html']/namespace-uri()
```

Zoom sur le cas des ID et IDREF(S)

Fonctions :
id
idref

Utilisation (exemple) :
`//id("0-201-53771-0")/child::author` retourne les auteurs de l'élément dont l'ID est "0-201-53771-0"

`/library/document[3]/id(@references)/titre` retourne les titres des documents référencés (references est un IDREFS) par le troisième document

`//idref("0-201-53771-0")` retourne tous les éléments qui référencent "0-201-53771-0"

XPath 2.0

- Repose toujours sur la notion de chemin de localisation afin de naviguer dans le document XML.
- Pas juste une extension de XPath 1.0, c'est une véritable refonte de XPath : une fusion entre XPath 1.0 et XQuery.
- Caractéristiques :
  - Support de 19 types prédéfinis (date, etc), avec test de type ;
  - Réponse = séquence ordonnée de valeurs (et plus un NodeSet) ;
  - Opérateur d'intersection, union et except ;
  - Opérateurs de quantification ;
  - Boucles, tri, définition interne (let) ;
  - Expressions conditionnelles (if-then-else) ;
  - Extensible (possibilité de créer des fonctions utilisateurs).

⇒ Un sous-language de XQuery que nous verrons dans la suite du cours

Plan du cours

1 XPath

2 XQuery

Ce cours XQuery

Une partie de cours XQuery est fondée sur le contenu de :

Serge Abiteboul, Ioana Manolescu, Philippe Rigaux, Marie-Christine Rousset, and Pierre Senellart. *Web Data Management and Distribution*. Cambridge University Press, 2011.

`http://webdam.inria.fr/Jorge/`

Principes de XQuery

Langage fermé / Composition XQuery repose sur un modèle de données. Toute requête (ou expression) XQuery prend en entrée une ou plusieurs instances de ce modèle de données et produit toujours une instance du modèle de données (à laquelle une autre requête XQuery peut être appliquée). De cette façon, les requêtes XQuery peuvent être composées pour créer des requêtes plus complexes.

Sensibilité au type Il est possible d'associer un schéma XSD à l'interprétation d'une requête XQuery. Mais XQuery permet également d'interroger des documents auxquels n'est associé aucun schéma.

Compatibilité XPath XQuery est une extention de XPath 2.0. Ainsi, toute expression XPath est une requête XQuery.

Au niveau syntaxique, on a cherché à créer un langage simple et concis.

Le modèle de données de XQuery

Une *instance* du modèle de données XQuery, appelée *valeur*, est une *séquence d'items* où un item est soit un nœud soit une valeur atomique.

Ce modèle est volontairement général. 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

Le modèle de données de XQuery (suite)

- un document XML
- plusieurs documents XML (une *collection*).

Le modèle de données de XQuery (suite)

Quelques détails concernant les séquences :

- Un séquence de taille 1 est équivalente à l'item seul ((47) ≡ 47).
- Une séquence ne peut pas contenir une autre séquence (pas d'imbrication possible). Si nécessaire, la séquence est applanie.
- La notion de valeur *null* n'existe pas en XQuery.
- Une séquence peut être vide.
- Une séquence peut contenir des items hétérogènes.
- Une séquence est ordonnée (donc deux séquences contenant les mêmes items dans les ordres différents sont différentes).

Un détail concernant les items :

- Les nœuds ont une *identité*, mais pas les valeurs.

Quelques aspets syntaxiques de XQuery

- XQuery est sensible à la casse (*case-sensitive*), les mots clef du langage doivent être écrits en minuscule.
- Des commentaires peuvent être ajoutés en respectant la syntaxe
  `(:ceci est mon commentaire :)`

Évaluation d'une requête XQuery

Une expression est évaluée dans un *contexte*, incluant des informations relatives au *contexte statique* (informations disponibles lors de l'analyse statique d'une requête) et au *contexte dynamique* (informations disponibles au moment de l'exécution de le requête).

Par exemple :

- (contexte statique) correspondance prefixe/URI d'espace de noms
- (contexte statique) variables dans le périmètre de l'évaluation de l'expression, et type correspondant
- (contexte statique) signature des fonctions qui peuvent être utilisées
- (contexte dynamique) date et heure
- (contexte dynamique) item du contexte en cours de traitement
- (contexte dynamique) position de l'item courant dans le contexte
- (contexte dynamique) taille du contexte
- (contexte dynamique) valeur des variables
- (contexte dynamique) collections et documents disponibles

Les expressions XQuery

Une *expression* XQuery prend en entrée une valeur (une séquence d'items) et renvoie une valeur.

Une expression peut prendre différentes formes :

- expression de chemin,
- constructeur,
- expression FLWOR,
- expression conditionnelle,
- expression quantifiée,
- fonction,
- ...

Expressions simples

Une valeur est une expression

- **Litéraux :** `"Hello"`, 47, 4.7, 4.7E+2
- **Built values :** date("2008-03-15"), `true`(), `false`()
- **Variables :** x
- **Séquences :** (1, (2, 3), (), (4, 5)), equiv. à (1, 2, 3, 4, 5), equiv. à (intentionnellement) 1 `to` 5.

Un document XML est également une expression

```
<employee empid="12345">
  <name>John Doe</name>
  <job>XML specialist</job>
  <deptno>187</deptno>
  <salary>125000</salary>
</employee>
```

Le résultats de chacune de ces expressions est l'expression elle-même.

le cnam

Récupérer un document ou une collection

Une requête 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 un singleton contenant 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 et au delà : les expressions XPath dans XQuery

```
collection("shakespeare")//TITLE
```

Expressions XPath dans XQuery

L'expression XPath est évaluée pour chaque document de la collection, dans la séquence issue de `collection("shakespeare")`.

XPath et au delà : les expressions XPath dans XQuery (suite)

Test

En supposant de la sémantique associée à un document XML contenant la description de Hamlet (DTD jointe ci-dessous), que revoient les expressions XQuery ci-dessous ?

```
1
2  <!ENTITY amp "&#38;#38;">
3  <!ELEMENT PLAY (TITLE, FM, PERSONAE, SCNDESCR, PLAYSUBT, INDUCT?,
4                                          PROLOGUE?, ACT+, EPILOGUE?)>
5  <!ELEMENT TITLE (#PCDATA)>
6  <!ELEMENT FM (P+)>
7  <!ELEMENT P (#PCDATA)>
8  <!ELEMENT PERSONAE (TITLE, (PERSONA | PGROUP)+)>
9  <!ELEMENT PGROUP (PERSONA+, GRPDESCR)>
10 <!ELEMENT PERSONA (#PCDATA)>
11 <!ELEMENT GRPDESCR (#PCDATA)>
12 <!ELEMENT SCNDESCR (#PCDATA)>
13 <!ELEMENT PLAYSUBT (#PCDATA)>
14 <!ELEMENT INDUCT (TITLE, SUBTITLE*, (SCENE+|(SPEECH|STAGEDIR|SUBHEAD)+))>
15 <!ELEMENT ACT (TITLE, SUBTITLE*, PROLOGUE?, SCENE+, EPILOGUE?)>
16 <!ELEMENT SCENE (TITLE, SUBTITLE*, (SPEECH | STAGEDIR | SUBHEAD)+)>
17 <!ELEMENT PROLOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
18 <!ELEMENT EPILOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
19 <!ELEMENT SPEECH (SPEAKER+, (LINE | STAGEDIR | SUBHEAD)+)>
```

Listing 10 – Fichier play.dtd fourni avec les samples de eXist pour la collection shakespeare

XPath et au delà : les expressions XPath dans XQuery (suite)

```
doc("hamlet.xml")//TITLE
```

```
doc("hamlet.xml")/PLAY/ACT/SCENE[1]
```

```
collection("shakespeare")/PLAY/ACT/SCENE[1]/TITLE
```

```
(: prenez garde au parenthesage ici :)
(doc("hamlet.xml")/PLAY/ACT/SCENE)[1]/TITLE
```

Créer des éléments avec des constructeurs

XQuery permet la création de nouveaux éléments à l'aide de constructeurs. Les expressions utilisant des constructeurs peuvent inclure des requêtes XQuery à évaluer.

```
document
{
  element scenes
  {
    element p {doc("hamlet.xml")/PLAY/ACT[1]/SCENE/TITLE}
  }
}
```

ou plus simplement :

```
<scenes>
   <p>{doc("hamlet.xml")/PLAY/ACT[1]/SCENE/TITLE}</p>
</scenes>
```

Notez ici l'utilisation des accolades indiquant au processeur XQuery que cette expression doit être évaluée.

Créer des éléments avec des constructeurs (suite)

Test

Que retourne l'expression XQuery ci-dessous ?

```
<scenes>
   <query>This is a XQuery expression:
   collection("shakespeare")/PLAY/ACT[1]/SCENE/TITLE</query>
   <result>This is the result of its evaluation on the shakespeare collection:
   {collection("shakespeare")/PLAY/ACT[1]/SCENE/TITLE}</result>
</scenes>
```

Créer des éléments avec des constructeurs (suite)

Un autre exemple :

```
<hamlet nb_actes="{count(doc("hamlet.xml")//PLAY/ACT)}">
  <personnages>
       {doc("hamlet.xml")/PLAY/PERSONAE/PERSONA}
  </personnages>
</hamlet>
```

Créer des éléments avec des constructeurs (suite)

```
<hamlet nb_actes="5">
    <personages>
        <PERSONA>CLAUDIUS, king of Denmark. </PERSONA>
        <PERSONA>HAMLET, son to the late, and nephew to the present king.</PERSONA>
        <PERSONA>POLONIUS, lord chamberlain. </PERSONA>
        <PERSONA>HORATIO, friend to Hamlet.</PERSONA>
        <PERSONA>LAERTES, son to Polonius.</PERSONA>
        <PERSONA>LUCIANUS, nephew to the king.</PERSONA>
        <PERSONA>A Gentleman</PERSONA>
        <PERSONA>A Priest. </PERSONA>
        <PERSONA>FRANCISCO, a soldier.</PERSONA>
        <PERSONA>REYNALDO, servant to Polonius.</PERSONA>
        <PERSONA>Players.</PERSONA>
        <PERSONA>Two Clowns, grave−diggers.</PERSONA>
        <PERSONA>FORTINBRAS, prince of Norway. </PERSONA>
        <PERSONA>A Captain.</PERSONA>
        <PERSONA>English Ambassadors. </PERSONA>
        <PERSONA>GERTRUDE, queen of Denmark, and mother to Hamlet. </PERSONA>
        <PERSONA>OPHELIA, daughter to Polonius.</PERSONA>
        <PERSONA>Lords, Ladies, Officers, Soldiers, Sailors, Messengers, and other Attendants.</PERSONA>
        <PERSONA>Ghost of Hamlet s Father. </PERSONA>
    </personages>
</hamlet>
```

Créer des éléments avec des constructeurs (suite)

```
<hamlet nb_actes="{count(doc("hamlet.xml")//PLAY/ACT)}">
  <personnages>
      {doc("shakespeare/hamlet.xml")/PLAY/PERSONAE/PERSONA}
  </personnages>
</hamlet>
```

Syntaxe alternative :

```
document
{
  element hamlet
  {
    attribute nb_actes {count(doc("shakespeare/hamlet.xml")//PLAY/ACT)},
    element personages {doc("shakespeare/hamlet.xml")/PLAY/PERSONAE/PERSONA}
  }
}
```

Variable

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).

Expressions FLWOR

- 4 clauses :
  - For (itération sur une séquence),
  - Let (définition ou instanciation d'une variable),
  - Where (prédicat),
  - Order by (tri du résultat), and
  - Return (contructeur de résultat).
- Dans l'esprit du SELECT-FROM-WHERE de SQL

Expressions FLWOR (suite)

Les clauses `for` et `let`.

Ces deux clauses 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.

Expressions FLWOR (suite)

Instanciation d'une variable :

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

Parcours d'une séquence :

```
for $i in (1,2,3)
return
<chantons>
  { $i } kilometre(s) a pieds,
  Ca use, ca use
  { $i } kilometre(s) a pieds,
  Ca use les souliers
</chantons>
```

Expressions FLWOR (suite)

La séquence peut être obtenue par évaluation d'une expression XQuery (ci-dessous XPath)

Instanciation d'une variable :

```
let $speechs := doc("hamlet.xml")//SPEECH
return $speechs
```

Parcours d'une séquence :

```
for $speech in doc("hamlet.xml")//SPEECH
return $speech
```

Pour travailler le résultat :

```
for $speech in doc("hamlet.xml")//SPEECH
return $speech/SPEAKER
```

Expressions FLWOR (suite)

Un autre example :

```
let $born := ('born, born, born')
let $alive := (' to be alive')
for $i in (1,2,3,4,5,6)
   return
        <refrain numero="{$i}">
        You see you were
        {$born} {$alive}
        You see you were
        {$born}
        born {$alive}
        </refrain>
```

Test

Que renvoie cette expression XQuery ?

Expressions FLWOR (suite)

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>
```

Test

Que renvoie cette expression XQuery ?

Expressions FLWOR (suite)

La clause `where` est assez similaire à son synonyme SQL. Elle permet de filtrer un item en fonction d'une condition.

```
for $line in doc("hamlet.xml")//SPEECH/LINE
where contains($line,"To be, or not to be")
return $line/parent::SPEECH/parent::SCENE/TITLE
```

```
<TITLE>SCENE I. A room in the castle.</TITLE>
```

Expressions FLWOR (suite)

```
for $line in doc("hamlet.xml")//SPEECH/LINE
where contains($line,"heart")
return $line/parent::SPEECH/child::SPEAKER
```

Pour chaque élément (stocké dans $line) de
doc("hamlet.xml")//SPEECH/LINE
satisfaisant contains($line,"heart")
retourner $line/parent::SPEECH/child::SPEAKER

Expressions FLWOR (suite)

```
for $line in doc("hamlet.xml")//SPEECH/LINE
where contains($line,"heart")
return $line/parent::SPEECH/child::SPEAKER
```

```
<SPEAKER>FRANCISCO</SPEAKER>
<SPEAKER>KING CLAUDIUS</SPEAKER>
<SPEAKER>KING CLAUDIUS</SPEAKER>
<SPEAKER>KING CLAUDIUS</SPEAKER>
<SPEAKER>KING CLAUDIUS</SPEAKER>
<SPEAKER>KING CLAUDIUS</SPEAKER>
<SPEAKER>KING CLAUDIUS</SPEAKER>
<SPEAKER>HAMLET</SPEAKER>
<SPEAKER>LAERTES</SPEAKER>
<SPEAKER>OPHELIA</SPEAKER>
<SPEAKER>HAMLET</SPEAKER>
<SPEAKER>HAMLET</SPEAKER>
<SPEAKER>HAMLET</SPEAKER>
<SPEAKER>HAMLET</SPEAKER>
<SPEAKER>LORD POLONIUS</SPEAKER>
<SPEAKER>HAMLET</SPEAKER>
<SPEAKER>KING CLAUDIUS</SPEAKER>
...
```

Expressions FLWOR (suite)

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

```
//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
```

Expressions FLWOR (suite)

La clause `order by` permet de spécifier l'ordre de tri du résultat.

```
for $p in doc("hamlet.xml")//PERSONA
order by $p
return $p
```

```
<PERSONA>A Captain.</PERSONA>
<PERSONA>A Gentleman</PERSONA>
<PERSONA>A Priest. </PERSONA>
<PERSONA>BERNARDO</PERSONA>
<PERSONA>CLAUDIUS, king of Denmark. </PERSONA>
<PERSONA>CORNELIUS</PERSONA>
<PERSONA>English Ambassadors. </PERSONA>
<PERSONA>FORTINBRAS, prince of Norway. </PERSONA>
<PERSONA>FRANCISCO, a soldier.</PERSONA>
<PERSONA>GERTRUDE, queen of Denmark, and mother to Hamlet. </PERSONA>
<PERSONA>GUILDENSTERN</PERSONA>
<PERSONA>Ghost of Hamlet s Father. </PERSONA>
<PERSONA>HAMLET, son to the late, and nephew to the present king.</PERSONA>
<PERSONA>HORATIO, friend to Hamlet.</PERSONA>
<PERSONA>LAERTES, son to Polonius.</PERSONA>
...
<PERSONA>VOLTIMAND</PERSONA>
```

Les expressions conditionnelles

```
for $b in doc("hamlet.xml")//SPEECH
return
<speech>
  {$b/SPEAKER}
  <longueur>
  {
   if (count($b/LINE)>10)
   then 'long speech'
   else 'court speech'
  }
  : {(count($b/LINE))} lignes</longueur>
</speech>
```

Les expressions conditionnelles (suite)

```
...
<speech>
    <SPEAKER>BERNARDO</SPEAKER>
    <longueur>court speech
  : 4 lignes</longueur>
</speech>
<speech>
    <SPEAKER>HORATIO</SPEAKER>
    <longueur>long speech
  : 27 lignes</longueur>
</speech>
<speech>
    <SPEAKER>MARCELLUS</SPEAKER>
    <longueur>court speech
  : 1 lignes</longueur>
</speech>
...
```

Jointures

```
for $t in doc("books.xml")//title
for $e in doc("reviews.xml")//entry
where $t=$e/title
return <reviews>{$t, $e/remarks}</reviews>
```

ou

```
for $t in doc("books.xml")//title
          $e in doc("reviews.xml")//entry
where $t=$e/title
return <reviews>{$t, $e/remarks}</reviews>
```

Les expressions avec quantificateur

Quantificateur existentiel

```
for $e in doc("liste_entrepots.xml")//entrepots
where some $p in $e/product
      satisfies ($p/origine='Ecosse')
return $e/reference
```

Quantificateur universel

```
for $e in doc("liste_entrepots.xml")//entrepots
where every $p in $e/product
      satisfies ($p/origine='France')
return $e/reference
```

Les expressions avec compteur

```
for $p at $i in doc("hamlet.xml")//PERSONA
return <personne ordre="{$i}">{data($p)}</personne>
```

où data(s) prend en entrée une séquence s d'items et renvoie une séquence des valeurs atomiques associées aux items.

```
<personne ordre="1">CLAUDIUS, king of Denmark. </personne>
<personne ordre="2">HAMLET, son to the late, and nephew to the present king.</personne>
<personne ordre="3">POLONIUS, lord chamberlain. </personne>
<personne ordre="4">HORATIO, friend to Hamlet.</personne>
<personne ordre="5">LAERTES, son to Polonius.</personne>
<personne ordre="6">LUCIANUS, nephew to the king.</personne>
<personne ordre="7">VOLTIMAND</personne>
<personne ordre="8">CORNELIUS</personne>
<personne ordre="9">ROSENCRANTZ</personne>
<personne ordre="10">GUILDENSTERN</personne>
<personne ordre="11">OSRIC</personne>
...
```

Les expressions avec compteur (suite)

```
for $p at $i in doc("hamlet.xml")//PERSONA
where $i=3
return <personne ordre="{$i}">{data($p)}</personne>
```

où data(s) prend en entrée une séquence s d'items et renvoie une séquence des valeurs atomiques associées aux items.

```
<personne ordre="3">POLONIUS, lord chamberlain. </personne>
```

La clause distinct-values

```
for $speaker in data(doc("hamlet.xml")//SPEAKER)
where ends-with($speaker,"US")
return $speaker
```

```
MARCELLUS
MARCELLUS
MARCELLUS
MARCELLUS
MARCELLUS
MARCELLUS...
MARCELLUS
KING CLAUDIUS
CORNELIUS
KING CLAUDIUS
KING CLAUDIUS
LORD POLONIUS
KING CLAUDIUS
KING CLAUDIUS
KING CLAUDIUS
KING CLAUDIUS
MARCELLUS
MARCELLUS
...
```

La clause `distinct-values` (suite)

```
for $speaker in distinct-values(data(doc("hamlet.xml")//SPEAKER))
where ends-with($speaker,"US")
return $speaker
```

```
MARCELLUS
KING CLAUDIUS
CORNELIUS
LORD POLONIUS
LUCIANUS
```

XQuery vs. XSLT

- XSLT est un langage bien adapté à la transformation de documents XML.
- XQuery est un langage bien adapté à l'interrogation de (grand ensembles de) documents.