XQuery

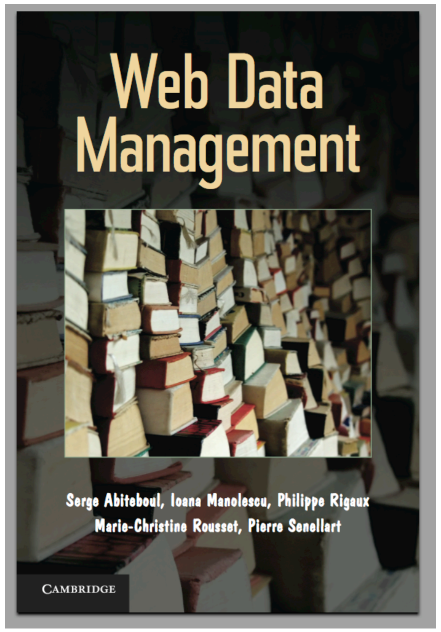
Federico Ulliana
UM, LIRMM, INRIA GraphIK Team

Slides collected from J. Cheney, S. Abiteboul, I. Manolescu, P. Senellart, P. Genevès, D. Florescu, and the W3C

Readings

[WDM-Query] Web Data Management – XPath / XQuery

http://webdam.inria.fr/Jorge/files/wdm-xpath.pdf


Web Data Management
Serge Abiteboul, Ioana Manolescu, Philippe Rigaux
Marie-Christine Rousset, Pierre Senellart
CAMBRIDGE

Recap of last class

XML Queries

- XML documents are hierarchical structures (trees; opposed to relational tables that are “flat”)

- XML Queries should ***navigate*** hierarchical structures (XPath)

- XML Queries should ***transform*** hierarchical structures (XQuery)

W3C Standardization Roadmap

1999
XPath 1.0

2007
XPath 2.0 first edition
XQuery 1.0 first edition

2010
XPath 2.0 second edition
XQuery 1.0 second edition

2014
XQuery 3.0

2017 (support for JSON)
XPath 3.0
XQuery 3.1

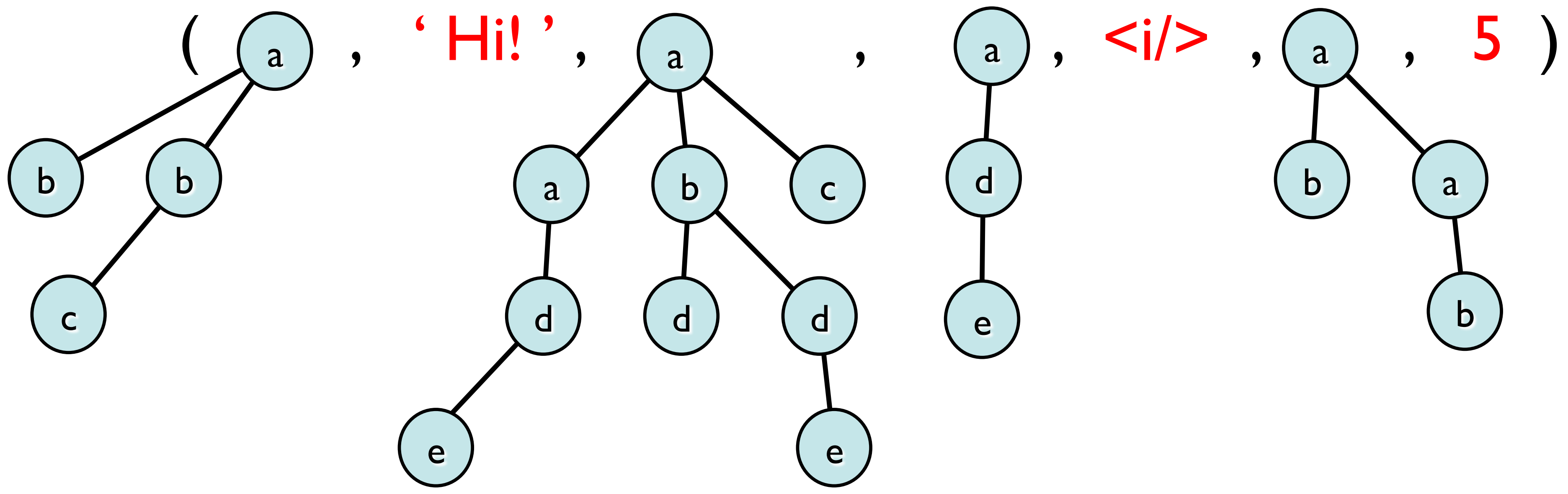
XML Working Group
ended on 31.08.2017

THE XQUERY LANGUAGE

XQuery : Goals

- “SQL-like” query language for XML
- Extend XPath 2.0
- Optimization in mind
    - reason for the verbose syntax
    - reason for a functional language

XQuery : Evaluation



Every expression evaluates to a sequence of trees and/or primitive values (integers, strings)

XQuery

Really many features and high expressive power.

We focus on constructs original for a DB language.

1. Iteration / let-binding

2. Element construction (static/dynamic)

3. Functions

For / Let

From XPath to XQuery

```
/a/b/c/parent::*/d
```

↓

```
for $x1 in /a return
 for $x2 in $x1/b return
  for $x3 in $x2/c return
   for $x4 in $x3/parent::* return
    for $x5 in $x4/d return $x5
```

From XPath to XQuery

```
/a/b/c/parent::*/d
```

↓

```
for $x in /a/b/c
 return $x/parent::*/d
```

For Iteration

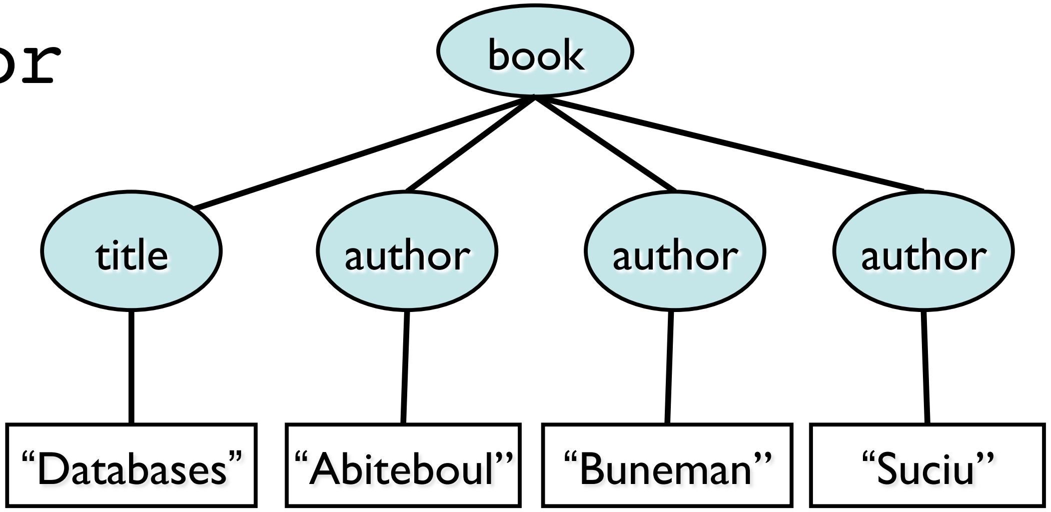
```
for $x in Q₁ return Q₂
```

Semantics

1. Evaluates `Q1` to a sequence
2. Generates one binding of `$x` for each element
3. Evaluates `Q2` for each `$x`-binding
4. Concatenate results in order

Iteration

```
for $x in //author
return $x/text()
```



Result

(“Abiteboul” , “Buneman”, “Suciu”)

Let binding

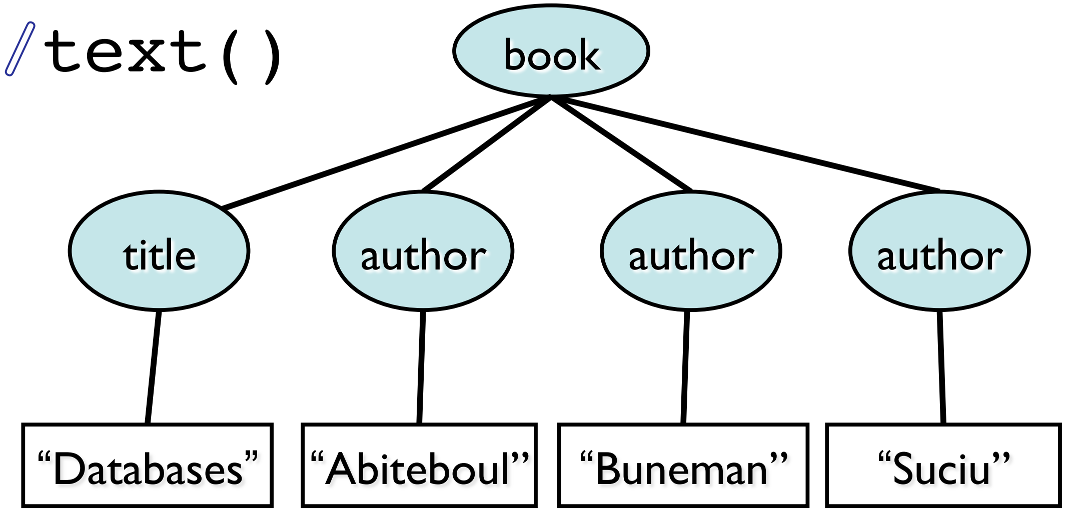
```
let $x := Q₁ return Q₂
```

Semantics

1. Evaluates expression Q_1 to a single value (sequence)
2. Binds `$x` to that value
3. Evaluates Q_2 with new binding

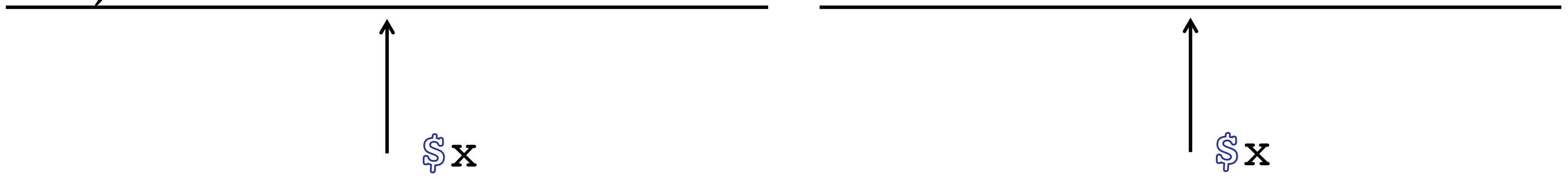
Let binding

```
let $x := //author/text()
return
( $x , $x )
```



Result

("Abiteboul","Buneman","Suciu","Abiteboul","Buneman","Suciu")



Let vs. For

```
let $x := (1,2,3)
 let $y := ("a","b")
   return ($x,$y)
```

```
Result
                (1,2,3,"a","b")
```

Let vs. For

```
for $x in (1,2,3)
 let $y := ("a","b")
   return ($x,$y)
```

```
Result
    (1,"a","b",2,"a","b",3,"a","b")
```

Let vs. For

```
let $x := (1,2,3)
 for $y in ("a","b")
   return ($x,$y)
```

```
Result
            (1,2,3,"a",1,2,3,"b")
```

Let vs. For

```
for $x in (1,2,3)
 for $y in ("a","b")
   return ($x,$y)
```

```
Result
 (1,"a",1,"b",2,"a",2,"b",3,"a",3,"b")
```

Let vs. For

```
for $x in (1,2,3)
 for $y in ("a","b")
   where $x = 4
     return ($x,$y)
```

Result

```
()
```

Where

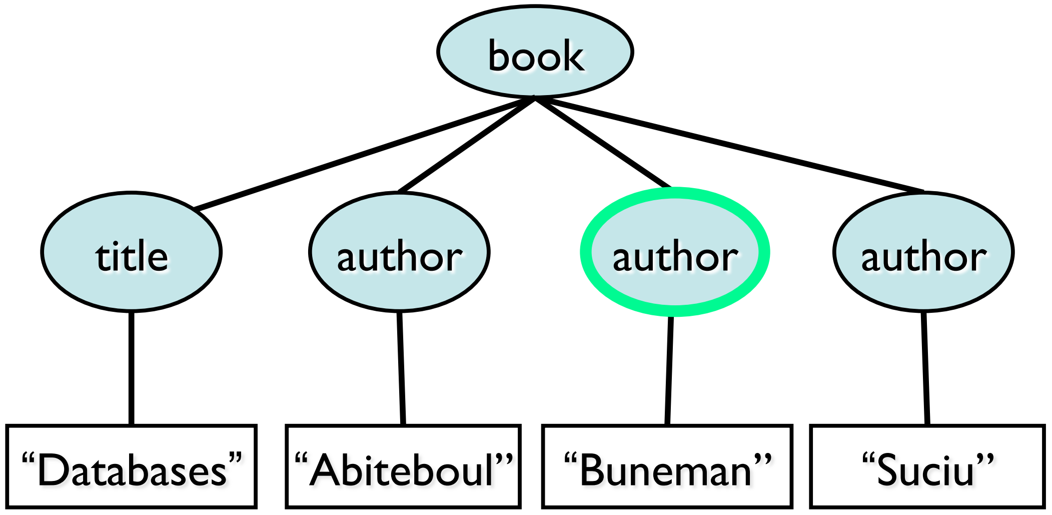
```
for $x in Q₁
 where Q'
   return Q₂
```

Semantics

1. Evaluates Q_1 to a sequence
2. Generates one binding of $x for each element
3. Evaluates Q' for each binding of $x
   1. If Q' is empty return ( )
   2. Otherwise, evaluates Q_2 with the binding

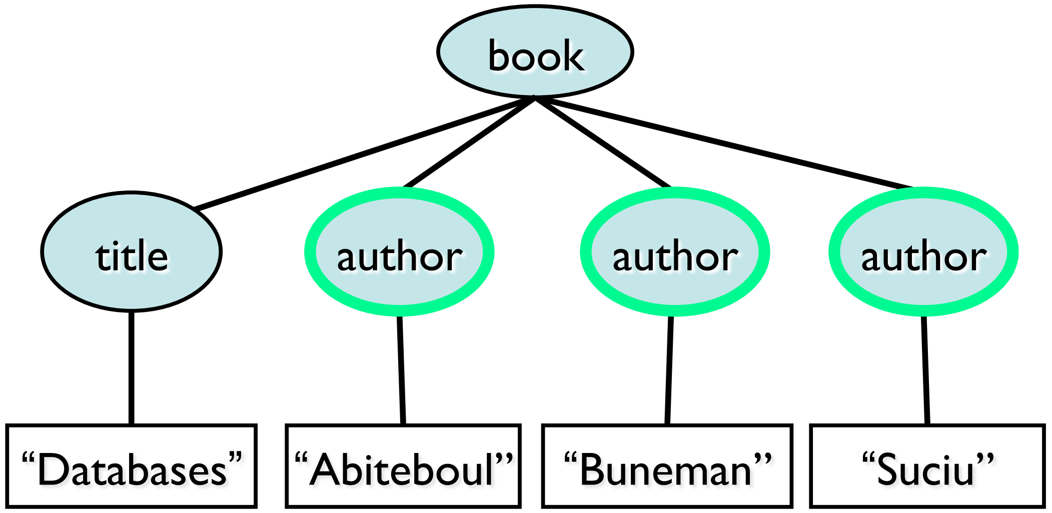
Where

```
for $x in //author
where $x/text()= “Buneman”
    return $x
```



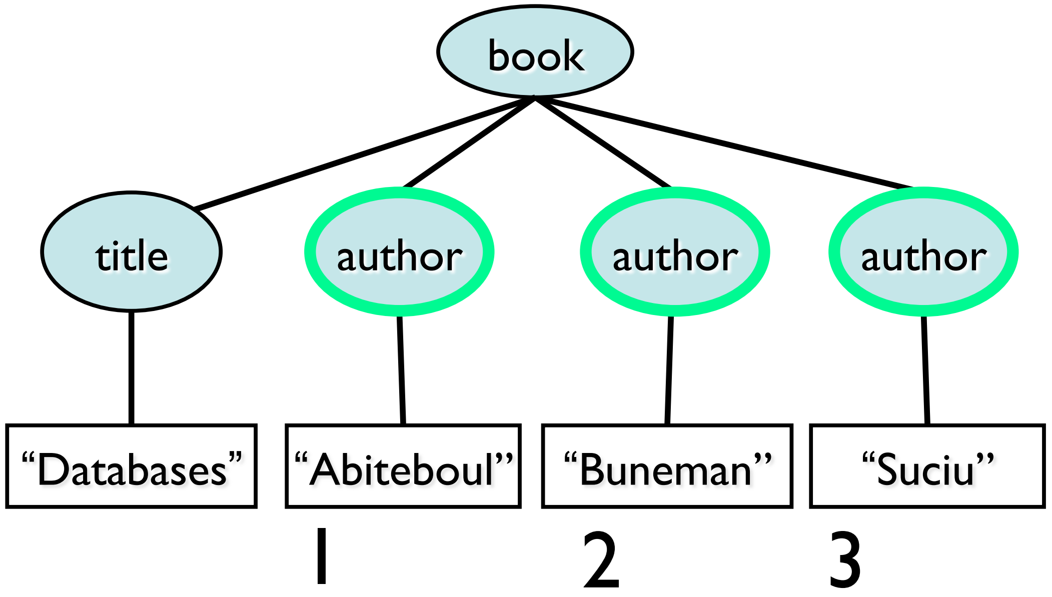
Where / Let

```
let $x := //author
where $x/text()= "Buneman"
    return $x
```



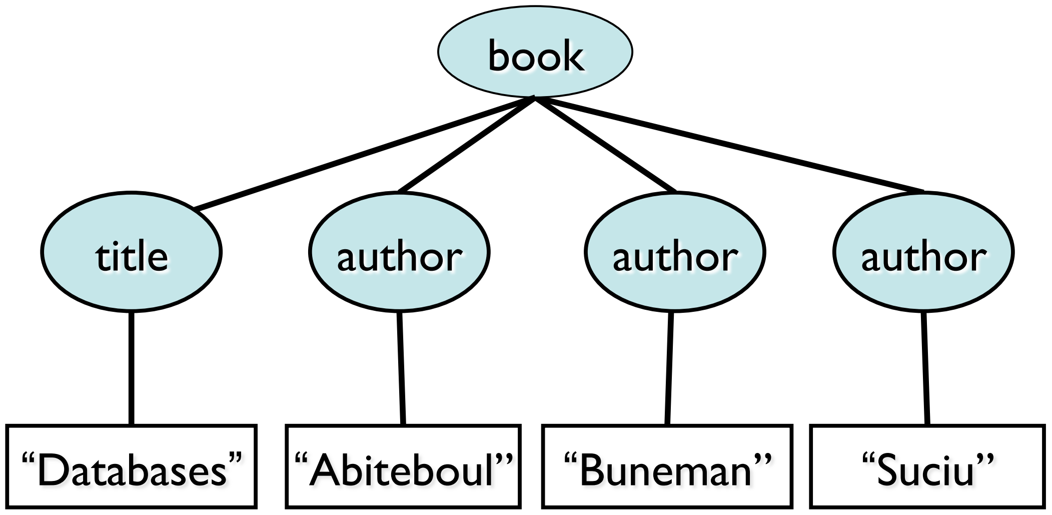
Order by

```
for $x in //author
return $x
```



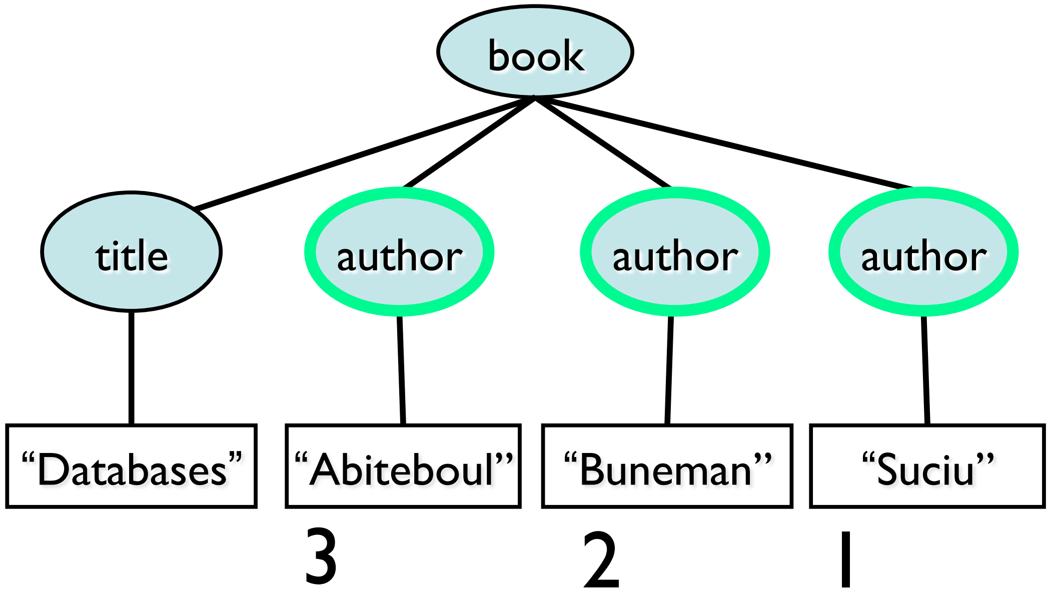
Order by

```
for $x in //author
order by $x/text() desc
return $x
```



Order by

```
for $x in //author
order by $x/text() desc
return $x
```



Conditionals

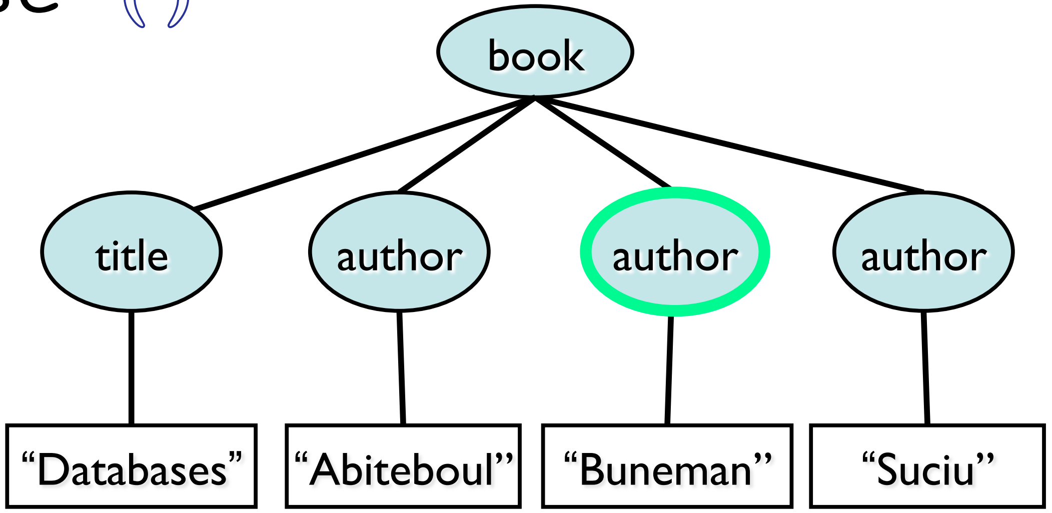
`if` Q_0 `then` Q_1 `else` Q_2

Evaluate Q_0

- if Q_0 is not empty, evaluate Q_1 (then-branch)
- if Q_0 is empty, evaluate Q_2 (else-branch)

If

```
for $x in //author return
if $x/text()= “Buneman”
   then $x else ()
```



XQuery-FLWOR

`for` : each item in an XQuery sequence

`let` : a new variable have a specified value

`where` : a condition expressed in XQueryis true

`order by` : the value of an XQuery expression

`return` : a sequence of items

+ `if then else`

Element / Attribute Creation

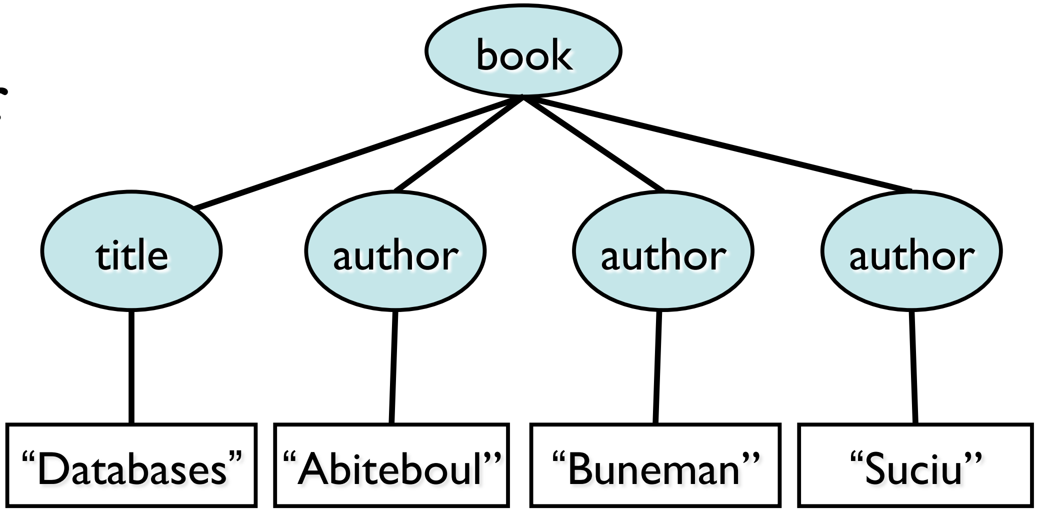
Element Creation

XQuery
code starts

```
<myNewElement> {

      (:Content of my new element:)

}
</myNewElement>
```

Nested Loop

```
for $x in //book,
for $y in $x/author
  return
  ( $x/title , $y )
```

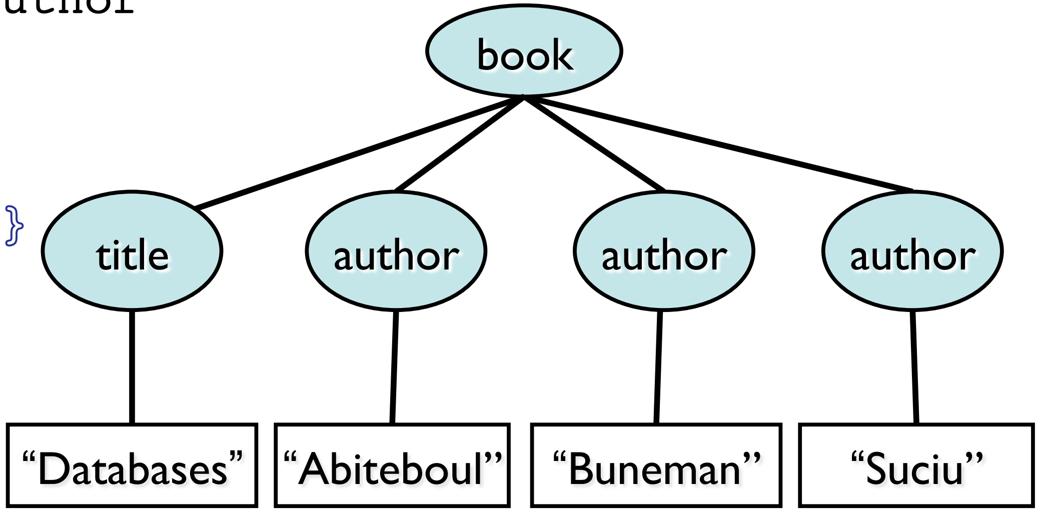


Result (“flat” sequence of 6 elements!)

```
<title>Databases</title> <author> Abiteboul </author>

<title>Databases</title> <author>  Buneman  </author>

<title>Databases</title> <author>   Suciu   </author>
```

Element Creation

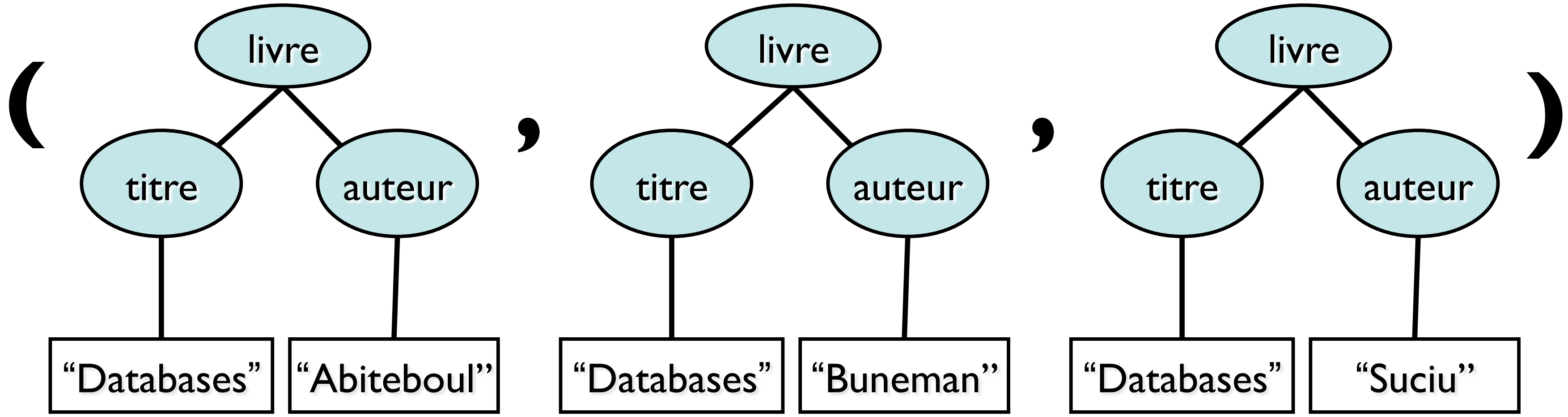
```
for $x in //book, $y in $x/author
    return
  <livre>
     <titre>
          { $x/title/text() }
     </titre>
     <auteur>
          { $y/text() }
     </auteur>
  </livre>
```



Result (3 elements!)

```
<livre> <titre>Databases</titre> <auteur> Abiteboul </auteur> </livre>

<livre> <titre>Databases</titre> <auteur>  Buneman  </auteur> </livre>

<livre> <titre>Databases</titre> <auteur>   Suciu   </auteur> </livre>
```

Element Creation



Result

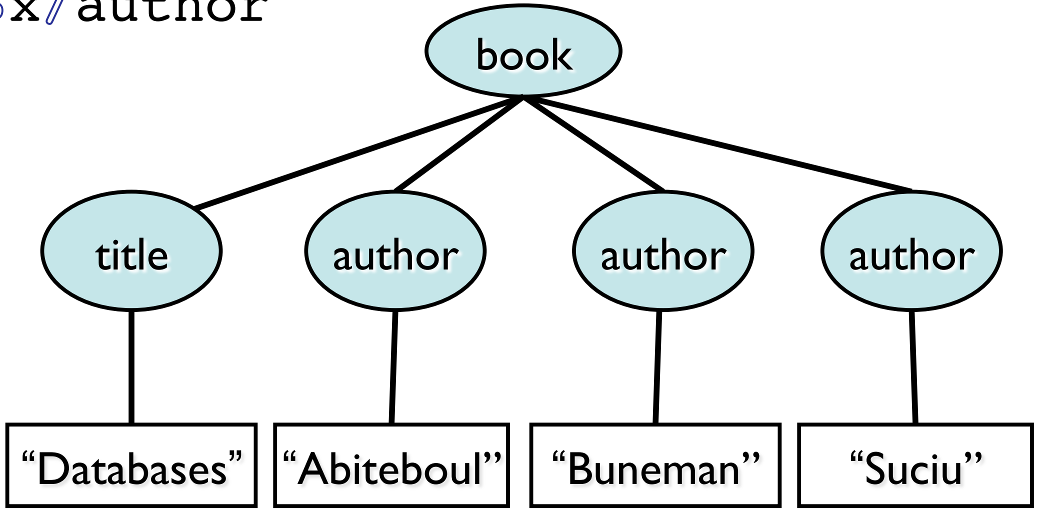
```
<livre> <titre>Databases</titre> <auteur> Abiteboul </auteur> </livre>

<livre> <titre>Databases</titre> <auteur>  Buneman  </auteur> </livre>

<livre> <titre>Databases</titre> <auteur>   Suciu   </auteur> </livre>
```

Attribute Creation

```
< myNewElement  myNewAttribute = "{

    (:Content of my new attribute:)

  }"
>


</myNewElement>
```

Attribute Creation

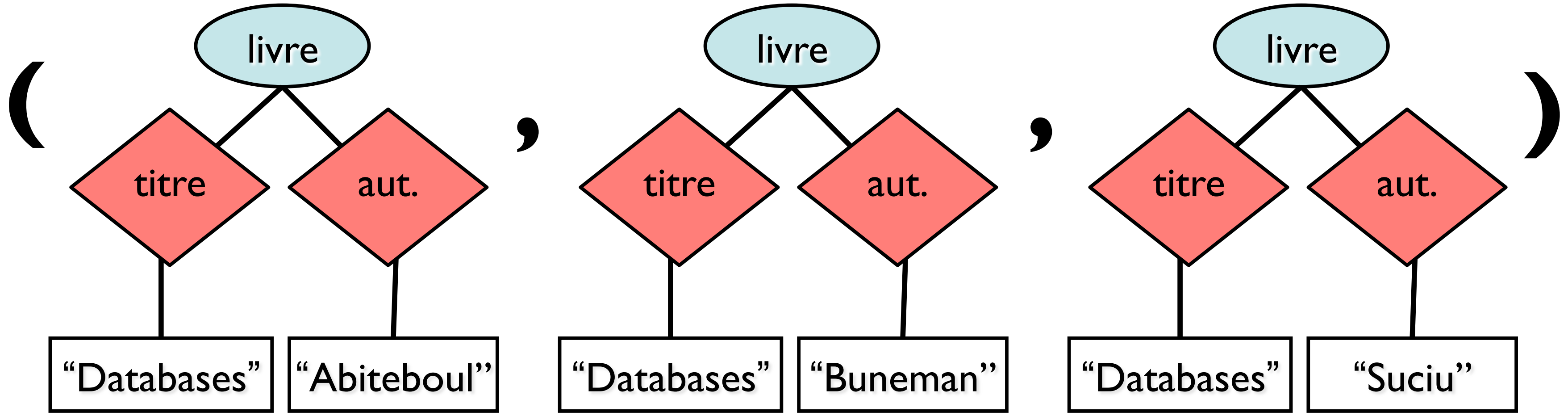
```
for $x in //book, $y in $x/author
   return

  <livre
    titre  =
     "{ $x/title/text() }"
    auteur =
     "{ $y/text() }"
  />
```



Result (3 elements!)

```
<livre titre="Databases" auteur="Abiteboul" /
>
<livre titre="Databases" auteur="Buneman" />
<livre titre="Databases" auteur="Suciu" />
```

Attribute Creation



Result

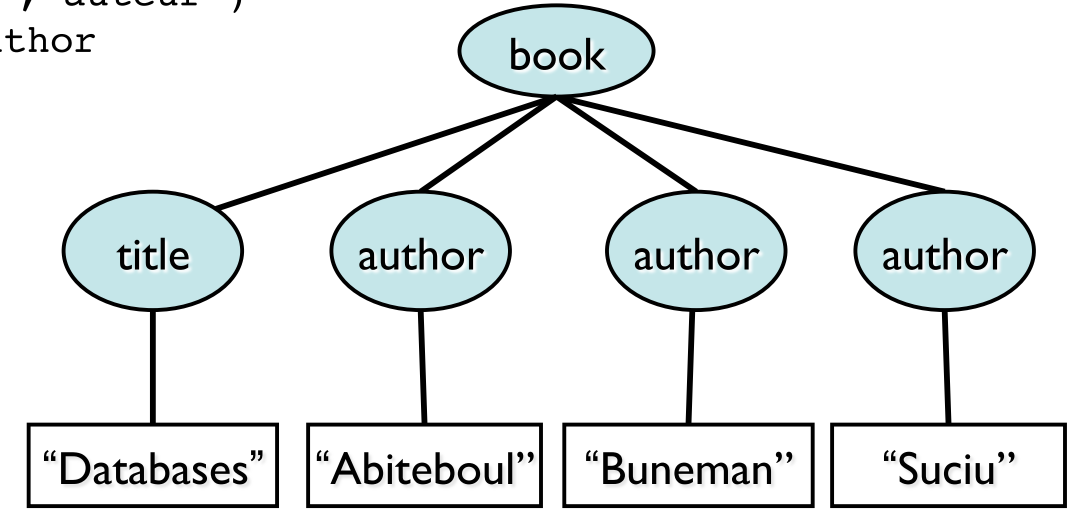
```
<livre titre="Databases" auteur="Abiteboul" /
>
<livre titre="Databases" auteur="Buneman" />
<livre titre="Databases" auteur="Suciu" />
```

Generic Syntax

```
element
  { myNewElement }
  {
    (:Content of my new element:)

    attribute
    {myNewAttribute}
    {(:Content of my new attribute:)}

  }
```

Parametric Element Creation

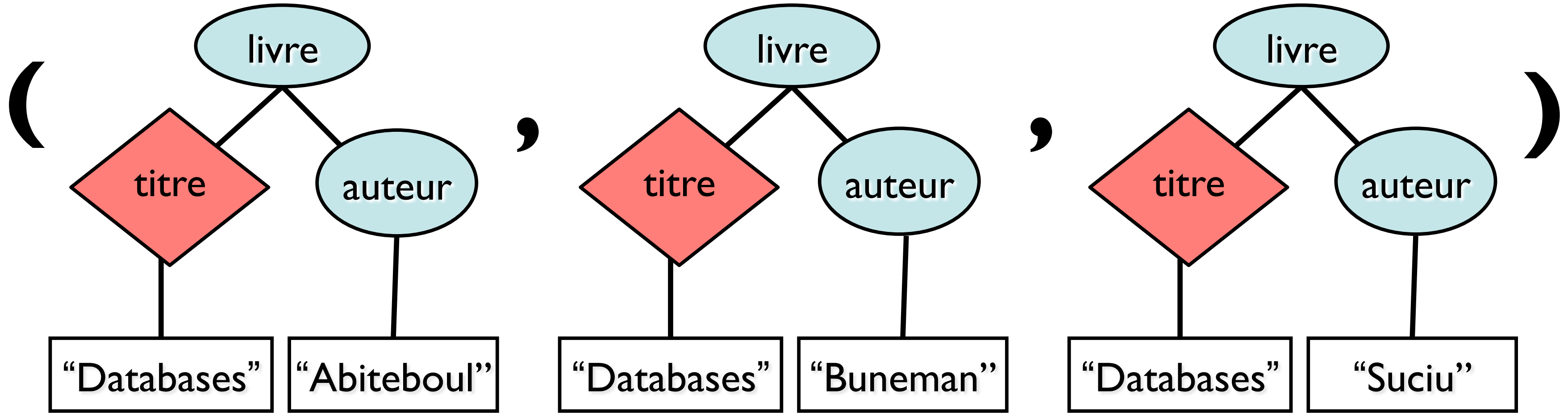
```
let $frenchTags:=('livre','titre','auteur')
  for $x in //book, $y in $x//author
    return

  element
     {$frenchTags[1]}
     {
          attribute
              {$frenchTags[2]}
              {$x/title/text()}
          ,
          element
              {$frenchTags[3]}
              {$y/text()}
      }
```



Result

```
<livre titre="Databases"> <auteur>Abiteboul</auteur> </livre>
<livre titre="Databases"> <auteur>Buneman</auteur> </livre>
<livre titre="Databases"> <auteur>Suciu</auteur> </livre>
```

Attribute Creation



Result

```
<livre titre="Databases"> <auteur>Abiteboul</auteur> </livre>
<livre titre="Databases"> <auteur>Buneman</auteur> </livre>
<livre titre="Databases"> <auteur>Suciu</auteur> </livre>
```

- Element Creation
- Attribute Creation
- Parametric/Dynamic Creation

Functions

Built-in functions

Includes all XPath primitive functions

- `last()`, `position()`, `not()`, etc.

Equality: has same (strange) semantics as in XPath

- i.e., `(1,2) = (2,3)` evaluates to `true`

`sum(), min(), max(),count()`

- like in SQL

User-definable functions

```
let $y:= //book return

( for $x1 in $y/author
    return $x1//text() ,

  for $x2 in $y/title
    return $x2//text() ,

  for $x3 in $y/year
    return $x3//text() )
```

User-definable functions

Can define functions to abbreviate parts of queries

```
declare function printText($n)
{
 for $x in $n
  return $x//text()
}
```

User-definable functions

```
let $y:= //book return

( for $x1 in $y/author
    return $x1//text()   ,

  for $x2 in $y/title
    return $x2//text() ,

  for $x3 in $y/year
    return $x3//text() )
```

```
let $y:= //book return

(
 printText($y/author) ,

 printText($y/title) ,

 printText($y/year)

)
```

Functions can be recursive!

```
declare function sumN($N as xs:int) {
   if ($N < 1) then
    return 0
   else
    let $x:= sumN($N-1)
    return  ($N + $x)
}
```

Functions can be recursive!

```
declare function d-o-s($x) {
          $x, d-o-s($x/child::node())
}
```

Built-in functions

User-defined functions

Conclusions

We’ve seen the main components of the XML family

- XML, DTD, XPath, XQuery

But there is much more

- XML Schemas, Namespaces, Integrity Constraints
- Storage/Evaluation : Native, Persistent, In-memory, Relational
- Optimisation, Parallelism, Typing
- Updates, Scripting

XML vs. OO

Encapsulation

- OO hides data
- XML makes data explicit

Type Hierarchy

- OO defines superset/subset relationship
- XML there is no equivalent for that

Data + Behavior

- OO packages them together
- XML separates data from its interpretation

XML vs. Relational

Structural Differences

- tree vs. table
- heterogeneous vs. homogeneous
- optional vs. strict typing

Some Commonalities

- logical and physical data independance
- declarative semantics
- generic data model