NPRG036

XML Technologies

Lecture 9

Advanced XQuery and XSLT

20. 4. 2020

Author: **Irena Holubová**
Lecturer: **Martin Svoboda**

http://www.ksi.mff.cuni.cz/~svoboda/courses/192-NPRG036/

Lecture Outline

- **XQuery Update Facility**
- **XSLT**
  - Advanced constructs
- **XQuery vs. XSLT**

XQuery Update Facility 1.0

XQuery Update Facility 1.0

- Extension of XQuery 1.0
  - Node insertion
  - Node deletion
  - Node modification (preserving identity)
  - Creating a modified copy of a node
- Assumes that the data are persistently stored in a database, file system, …
  - We change the stored data

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

XQuery Update Facility 1.0
Node Insertion

- Construct **insert**

```
insert node SourceExpr into TargetExpr
insert node SourceExpr as first into TargetExpr
insert node SourceExpr as last into TargetExpr
insert node SourceExpr after TargetExpr
insert node SourceExpr before TargetExpr
```

- Inserts copies of zero or more source nodes at a position specified with regards to a target node
  - Source nodes: SourceExpr
  - Target node: TargetExpr
- Instead of node we can use nodes (it does not matter)

XQuery Update Facility 1.0
Node Insertion

- Source nodes are inserted before / after the target node

```
insert node SourceExpr after TargetExpr
insert node SourceExpr before TargetExpr
```

- Source nodes are inserted at the end / beginning of child nodes of target node

```
insert node SourceExpr as first into TargetExpr
insert node SourceExpr as last into TargetExpr
```

- Source nodes are inserted among child nodes of the target node (position is implementation dependent)

```
insert node SourceExpr into TargetExpr
```

XQuery Update Facility 1.0
Node Insertion

- Conditions:
  - SourceExpr and TargetExpr cannot be update expressions
  - For the into versions TargetExpr must return exactly one element / document node
  - For other versions TargetExpr must return exactly one element / text / comment / processing instruction node

XQuery Update Facility 1.0

Node Insertion - Example

```
insert node  <phone>111111111</phone>
       after //customer/email

insert node  <phone>111111111</phone>
       into  //customer

for $p in //item
return
 insert node  <total-price>
                {$p/price * $p/amount}
              </total-price>
        as last into $p
```

XQuery Update Facility 1.0
Node Deletion

- Construct **delete**

```
delete node TargetExpr
```

- Deletes target nodes
  - Specified using TargetExpr
- Instead of node we can use nodes (it does not matter)
- Conditions:
  - TargetExpr cannot be an update expression
  - TargetExpr must return zero or more nodes
- If any of the target nodes does not have a parent, it depends on the implementation whether an error is raised

XQuery Update Facility 1.0
Node Deletion - Example

```
delete node //customer/email

delete node //order[@status = "dispatched"]

delete node for $p in //item
            where fn:doc("stock.xml")//product[code = $p/code]/items
                  = 0
            return $p

for $p in //item
where fn:doc("stock.xml")//product[code = $p/code]/items = 0
return delete node $p
```

XQuery Update Facility 1.0
Node Replacing

- Construct **replace**

```
replace node TargetExpr with Expr
```

- Replaces the target node with a sequence of zero or more nodes
  - Target node: TargetExpr
- Conditions:
  - TargetExpr cannot be update expressions
  - TargetExpr must return a single node and must have a parent

XQuery Update Facility 1.0
Value Replacing

- Construct **replace value of**

```
replace value of node TargetExpr with Expr
```

- Modifies the value of the target node
  - Target node: TargetExpr
- Conditions:
  - TargetExpr cannot be update expressions
  - TargetExpr must return a single node

XQuery Update Facility 1.0
Node/Value Replacing - Example

```
replace node (//order)[1]/customer
        with (//order)[2]/customer

for $v in doc("catalogue.xml")//product
return
 replace value of node $v/price
         with $v/price * 1.1
```

XQuery Update Facility 1.0
Other Functions

- Renaming

```
rename node TargetExpr as NewNameExpr
```

- Transformation
  - Creating a modified copy of a node (having a new identity)

```
copy $VarName := ExprSource
modify ExprUpdate
return Expr
```

XSLT:
Advanced Constructs

XSLT 1.0 – Sorting

- Element xsl:sort
  - Within xsl:apply-templates or xsl:for-each
    - Influences the order of further processing
  - Attribute select
    - According to what we sort
  - Attribute order
    - ascending / descending
      - Default: ascending

XSLT 1.0 – Sorting

```
<xsl:for-each select="//item">
 <xsl:sort select="./name" />
 ...
</xsl:for-each>
```

```
<xsl:for-each select="book">
 <xsl:sort select="author/surname"/>
 <xsl:sort select="author/firstname"/>
 <p>
   <xsl:value-of select="author/surname"/>
   <xsl:text> - </xsl:text>
   <xsl:value-of select="title"/>
 </p>
</xsl:for-each>
```

XSLT 1.0 – Keys

- Element xsl:key
  - Attribute name
    - Name of key
  - Attribute match
    - XPath expression identifying elements for which we define the key
  - Attribute use
    - XPath expression identifying parts of the key
- Function key(key-name, key-value)
  - Finds the node with key having key-name and value key-value

XSLT 1.0 – Keys

```
<xsl:key name="product-key"
         match="product"
         use="./product-code" />

<xsl:for-each select="//item">
 <xsl:variable name="prod"
               select="key('product-key', ./@code)" />
 <xsl:value-of select="$prod/name" />
 <xsl:value-of select="$prod/vendor" />
</xsl:for-each>
```

XSLT 1.0 – Modes

- Processing of the same nodes in different ways = modes
- Attribute mode of element xsl:template and xsl:apply-template
  - Only for unnamed templates

XSLT 1.0 – Modes

```
<xsl:template match="/">
 <xsl:apply-templates mode="overview" />
 <xsl:apply-templates mode="full-list" />
</xsl:template>

<xsl:template match="item" mode="overview">
 ...
</xsl:template>

<xsl:template match="item" mode="full-list">
 ...
</xsl:template>
```

XSLT 1.0 – Combinations of Scripts

- Referencing to another XSLT script
  - Element xsl:include
    - Attribute href refers to an included script
    - The templates are "included" (copied) to the current script
  - Element xsl:import
    - Attribute href refers to an imported script
    - In addition, the rules from the current script have higher priority than the imported ones
    - xsl:apply-imports – we want to use the imported templates (with the lower priority)

XSLT 1.0 – Combinations of Scripts

```
<!-- stylesheet A -->
<xsl:stylesheet ...>

 <xsl:import  href="C.xsl" />
 <xsl:include href="B.xsl" />

</xsl:stylesheet>
```

XSLT 1.0 – Copies of Nodes

- Element xsl:copy-of
  - Attribute select refers to the data we want to copy
  - Creates a copy of the node including all child nodes
- Element xsl:copy
  - Creates a copy of the current node, but not its attributes or child nodes

XSLT 1.0 – Copies of Nodes

```
<xsl:template match="/">
  <xsl:copy-of select="."/>
</xsl:template>
```

```
<xsl:template match="/|@*|*|text()">
  <xsl:copy>
    <xsl:apply-templates select="@*|*|text()"/>
  </xsl:copy>
</xsl:template>
```

- Both create a copy of the input document, but in a different way

XSLT 2.0

- Uses XPath 2.0
  - XSLT 1.0 uses XPath 1.0
- Adds new constructs (elements)
  - The output (input) can be into (from) multiple documents
  - User-defined functions
    - Can be called from XPath expressions
  - Element xsl:for-each-group for grouping of nodes
- …and many other extensions
  - see http://www.w3.org/TR/xslt20/

XSLT 2.0 – Output and Input

- Element xsl:result-document
  - Attribute href
    - URL of output document
  - Attribute format
    - Format of the output document
    - Reference to an xsl:output element
- Element xsl:output
  - New attribute name
    - To enable referencing

XSLT 2.0 – Output and Input

```
<xsl:output name="orders-report-format" method="xhtml" .../>
<xsl:output name="order-format"         method="xml" ... />

<xsl:template match="/">
 <xsl:result-document href="orders-report.html"
                      format="orders-report-format">
  <html>
   <body><xsl:apply-templates /></body>
  </html>
 </xsl:result-document>

 <xsl:for-each select="document('orders.xml')//order">
  <xsl:result-document href="order{./@number}.html"
                       format="order-format">
   <xsl:apply-templates select="." />
  </xsl:result-document>
 </xsl:for-each>
</xsl:template>
```

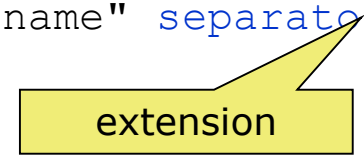
other input

XSLT 2.0 – Grouping of Nodes

- Grouping of nodes according to specific conditions
- Element xsl:for-each-group
  - Attribute select
    - Like for xsl:for-each
  - Attribute group-by
    - XPath expression specifying values according to which we group
  - … and other attributes for other types of grouping
  - Function current-group() returns items in the current group

XSLT 2.0 – Grouping of Nodes

```
<xsl:template match="/">
 <xsl:for-each-group select="document('products.xml')//product"
                     group-by="./category">
  <h1><xsl:value-of select="current-grouping-key()" /></h1>
  <p>
   <xsl:value-of select="current-group()/name" separator=", " />
  </p>
 </xsl:for-each-group>
</xsl:template>
```



XSLT 2.0 – User-defined Functions

- Element xsl:function
  - Attribute name
    - Name of function
  - Attribute as
    - Return value of function
  - Subelement xsl:param
    - Parameter of function
- Similar mechanism as named templates
- But we can use the functions in XPath expressions

XSLT 2.0 – User-defined Functions

```
<xsl:function name="mf:value-added-price" as="xs:anyAtomicType">
 <xsl:param name="price" as="xs:anyAtomicType"/>
 <xsl:value-of select="$price * 1.19" />
</xsl:function>

<xsl:template match="/">
 <html>
  <body>
   <xsl:value-of select="mf:value-added-price
     (sum(for $p in //item return $p/price * $p/amount))" />
  </body>
 </html>
</xsl:template>
```

XPath 2.0

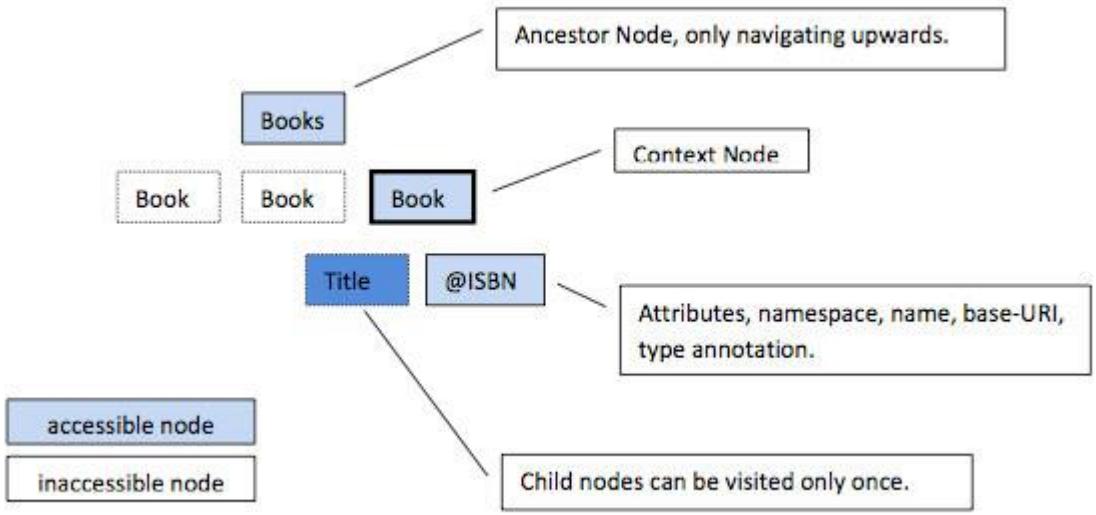
XSLT 3.0

- Currently just W3C candidate recommendation
  - To be used in conjunction with XPath 3.0
- Main extensions:
  - Streaming mode of transformations
    - Neither the source document nor the result document is ever held in memory in its entirety
    - Motivation: we do not want to load the entire document in memory
  - Higher order functions
  - Extended text processing
  - Improves modularity of large stylesheets
  - …

http://www.w3.org/TR/xslt-30/

XSLT 3.0 and Streaming

- Restrictions to be aware of:
  - We have access only to the current element attributes and namespace declaration
  - Sibling nodes and ancestor siblings are not reachable
  - We can visit child nodes only once



A processor that claims conformance with the streaming option offers a guarantee that an algorithm will be adopted allowing documents to be processed that are orders-of-magnitude larger than the physical memory available.

XSLT 3.0 and Streaming

```
<?xml version="1.0"?>
<xsl:stylesheet version="3.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

 <xsl:template match="/">
  <xsl:stream href="books.xml">
   <xsl:iterate select="/books/book">
    <xsl:result-document
         href="{concat('book', position(),'.xml')}">
      <xsl:copy-of select="."/>
    </xsl:result-document>
    <xsl:next-iteration/>
   </xsl:iterate>
  </xsl:stream>
 </xsl:template>
</xsl:stylesheet>
```

We explicitly indicate to stream the execution of its instruction body

XSLT 3.0 and Higher-Order Functions

- Higher order functions = functions that either take functions as parameters or return a function
- XSLT 3.0 introduces the ability to define anonymous functions
  - Enables meta-programming using lambda expressions
- Example:
  - (x, y) $\rightarrow$ x*x + y*y … lambda expression that calculates the square of two numbers and sums them
  - x $\rightarrow$ (y $\rightarrow$ x*x + y*y) … equivalent expression that accepts a single input, and as output returns another function, that in turn accepts a single input

XSLT 3.0 and Higher-Order Functions

```
<?xml version='1.0'?>
<xsl:stylesheet
     version="3.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xsl:template match="/">
    <xsl:variable name="f1" select="
      function($x as xs:integer) as (function(xs:integer) as
xs:integer){
        function ($y as xs:integer) as xs:integer{
          $x * $x + $y * $y
          }
      } "/>
    <xsl:value-of select="$f1(2)(3)"/>
  </xsl:template>
</xsl:stylesheet>
```

Variable f1 is assigned to an anonymous function that takes an integer and returns a function that takes an integer and returns an integer

XSLT 3.0 and Higher-Order Functions

- Support for common lambda patterns (operators)
  - map – applies the given function to every item from the given sequence, returning the concatenation of the resulting sequences
  - filter – returns items from the given sequence for which the supplied function returns true
  - fold-left – processes the supplied sequence from left to right, applying the supplied function repeatedly to each item, together with an accumulated result value
  - fold-right – respectively
  - map-pairs – applies the given function to successive pairs of items taken one from sequence 1 and one from sequence 2, returning the concatenation of the resulting sequences

http://www.w3.org/TR/xpath-functions-30/

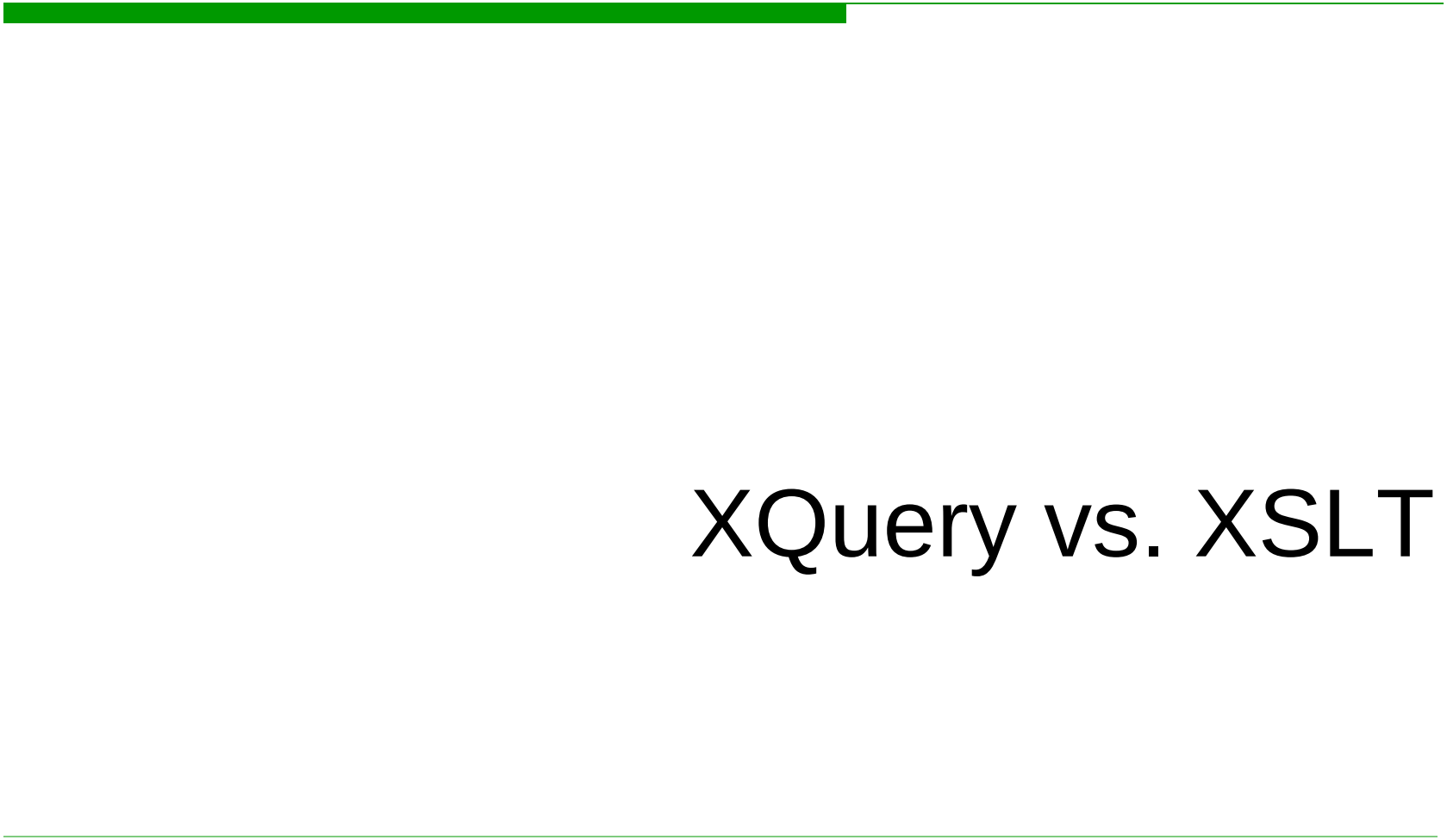
XSLT 3.0 and Higher-Order Functions

Folding that sums only positive numbers from a list

```
<?xml version="1.0"?>
<xsl:stylesheet  version="3.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:variable name="list" select="(10,-20,30,-40)"/>

  <xsl:template match="/">
    <xsl:variable name="f1" select="
      function($accumulator as item()*, $nextItem as item()) as item()*
        {
        if ($nextItem &gt; 0) then
           $accumulator + $nextItem
        else
           $accumulator
        }"/>
    <xsl:value-of select="fold-left($f1, 0, $list)"/>
  </xsl:template>
</xsl:stylesheet>
```


XQuery vs. XSLT

XQuery vs. XSLT

- XSLT = language for XML data transformation
  - Input: XML document + XSLT script
  - Output: XML document
    - Not necessarily
- XQuery = language for XML data querying
  - Input: XML document + XQuery query
  - Output: XML document
    - Not necessarily
- Seem to be two distinct languages
  - Observation: Many commonalities
  - Problem: Which of the languages should be used?

Example: variables and constructors

```
<emp empid="{$id}">
  <name>{$n}</name>
  <job>{$j}</job>
</emp>
```

```
<emp empid="{$id}">
  <name><xsl:copy-of select="$n"/></name>
  <job><xsl:copy-of select="$j"/></job>
</emp>
```

```
element {$tagname} {
   element description {$d} ,
   element price {$p}
}
```

```
<xsl:element name="{$tagname}">
  <description><xsl:copy-of select="$d"/></description>
  <price><xsl:copy-of select="$p"/></price>
</xsl:element>
```

Example: FLWOR

```
for $b in fn:doc("bib.xml")//book
where $b/publisher = "Morgan Kaufmann" and $b/year = "1998"
return $b/title
```

```
<xsl:template match="/">
  <xsl:for-each select="//book">
    <xsl:if test="publisher='Morgan Kaufmann' and year='1998' ">
      <xsl:copy-of select="title"/>
    </xsl:if>
  </xsl:for-each>
</xsl:template>
```

Example: join + variable

```
<big_publishers>
  { for $p in distinct-values(fn:doc("bib.xml")//publisher)
    let $b := fn:doc("bib.xml")/book[publisher = $p]
    where count($b) > 100
    return $p }
</big_publishers>
```

```
<xsl:for-each select="//publisher[not(.=preceding::publisher)]">
  <xsl:variable name="b" select="/book[publisher=current()]"/>
  <xsl:if test="count($b) > 100">
    <xsl:copy-of select="."/>
  </xsl:if>
</xsl:for-each>
```

Example: evaluations

```
<result>
 { let $a := fn:avg(//book/price)
   for $b in /book
   where $b/price > $a
   return
     <expensive_book>
       { $b/title }
       <price_difference>
         { $b/price - $a }
       </price_difference>
     </expensive_book> }
</result>
```

```
<xsl:variable name="avgPrice"
 select="sum(//book/price) div count(//book/price)"/>
<xsl:for-each
 select="/bib/book[price > $avgPrice]">
 <expensive_book>
   <xsl:copy-of select="title"/>
   <price_difference>
     <xsl:value-of select="price - $avgPrice"/>
   </price_difference>
 </expensive_book>
</xsl:for-each>
```

Example: if-then-else, order

```
for $h in //holding
order by (title)
return
   <holding>
      { $h/title ,
        if ($h/@type = "Journal")
        then $h/editor
        else $h/author }
   </holding>
```

```
<xsl:template match="/">
  <xsl:for-each select="//holding">
    <xsl:sort select="title"/>
    <holding>
      <xsl:copy-of select="title"/>
      <xsl:choose>
        <xsl:when test="@type='Journal'">
          <xsl:copy-of select="editor"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:copy-of select="author"/>
        </xsl:otherwise>
      </xsl:choose>
    </holding>
  </xsl:for-each>
</xsl:template>
```

Example: quantifiers

```
for $b in //book
where some $p in $b/para
satisfies fn:contains($p, "sailing") and fn:contains($p, "windsurfing")
return $b/title
```

```
<xsl:template match="/">
  <xsl:for-each select="//book">
    <xsl:if test="./para[contains(., 'sailing') and contains(., 'windsurfing')]">
      <xsl:copy-of select="title"/>
    </xsl:if>
  </xsl:for-each>
</xsl:template>
```

Example: quantifiers

```
for $b in //book
where every $p in $b/para
satisfies fn:contains($p, "sailing")
return $b/title
```

```
<xsl:template match="/">
  <xsl:for-each select="//book">
    <xsl:if test="count(./para)=count(./para[contains(., 'sailing')])">
      <xsl:copy-of select="title"/>
    </xsl:if>
  </xsl:for-each>
</xsl:template>
```

Example: functions

```
declare function depth($e)
{ if (fn:empty($e/*)) then 1 else fn:max(depth($e/*)) + 1 }

depth(fn:doc("partlist.xml"))
```

```
<xsl:template name="depth">
 <xsl:param name="node"/>
 <xsl:param name="level" select="1"/>
 <xsl:choose>
  <xsl:when test="not($node/*)">
   <xsl:value-of select="$level"/>
  </xsl:when>
  <xsl:otherwise>
   <xsl:call-template name="depth">
    <xsl:with-param name="level" select="$level + 1"/>
    <xsl:with-param name="node" select="$node/*"/>
   </xsl:call-template>
  </xsl:otherwise>
 </xsl:choose>
</xsl:template>
```

```
<xsl:template match="/">
 <xsl:call-template name="depth">
  <xsl:with-param name="node"
       select="document('partlist.xml')"/>
 </xsl:call-template>
</xsl:template>
```

Which of the Languages We Should Use?

- In general: It does not matter
- More precisely:
  - It depends on the application
- Rules:
  - If the data are stored in database $\Rightarrow$ XQuery
  - If we want to copy the document with only small changes $\Rightarrow$ XSLT
  - If we want to extract only a small part of the data $\Rightarrow$ XQuery
  - XQuery is easy-to-learn and simpler for smaller tasks
  - Highly structured data $\Rightarrow$ XQuery
  - Large applications, re-usable components $\Rightarrow$ XSLT

Advanced XQuery

Namespaces, data model, data types, use cases

Standard Namespaces

- xml =
  `http://www.w3.org/XML/1998/namespace`
- xs =
  `http://www.w3.org/2001/XMLSchema`
- xsi =
  `http://www.w3.org/2001/XMLSchema-instance`
- fn =
  `http://www.w3.org/2005/04/xpath-functions`
- xdt =
  `http://www.w3.org/2005/04/xpath-datatypes`
- local =
  `http://www.w3.org/2005/04/xquery-local-functions`

Special Namespaces

- Special XQuery namespaces
  - `dm` = access via data model
  - `op` = XQuery operators
  - `fs` = functions from XQuery formal semantics
- Without a special URI
- Constructs from them are not accessible from XPath/XQuery/XSLT

XQuery Data Model

- A language is closed with regard to data model if the values of each used expression are from the data model
- XPath, XSLT and XQuery are closed with regard to XQuery 1.0 and XPath 2.0 Data Model

http://www.w3.org/TR/xpath-datamodel/

XQuery Data Model

- Based on XML Infoset
- Requires other features with regards to power of XQuery (and XSLT)
  - We do not represent only XML documents (input) but also results (output)
  - Support for typed atomic values and nodes
- Types are based on XML Schema
  - Ordered sequences
    - Of atomic values
    - Mixed, i.e. consisting of nodes (including document) and atomic values

XQuery Data Model

- Sequence is an ordered collection of items
  - Cannot contain other sequences
- Item is a node or atomic value
  - Can exist only within a sequence
  - Can occur multiple times in a single sequence
  - Must have a data type

➔ Each language based on XQuery data model is strongly typed

➔ The result of a query is a sequence

XQuery Data Model

Atomic values

- Atomic value is a value from a domain of an atomic data type and is assigned with a name of the data type
- Atomic type is
  - a simple data type
  - derived from a simple data type of XML Schema

XQuery Data Model

Atomic values

- Simple data types
  - 19 XML Schema built-in data types
  - xs:untypedAtomic
    - Denotes that the atomic value has no data type
  - xs:anyAtomicType
    - Denotes that the atomic value has an atomic type, but we do not specify which one
    - Involves all atomic types
  - xs:dayTimeDuration
  - xs:yearMonthDuration

XQuery Data Model

Nodes

- 7 types of nodes
- **`document`, `element`, `attribute`, `text`, `namespace`, `processing instruction`, `comment`**
  - Less than in XML Infoset
  - E.g. no DTD and notation nodes
- Each node has identity
  - Like in XPath 2.0
- Each node has its type of content
  - xs:untyped denotes that the node does not have any data type

XQuery Data Model

Nodes

- Access to node value typed to `xs:string`
  - String value of a node
  - `fn:string()`
- Access to node value having the original data type
  - `fn:data()`

XQuery Data Model

Query Result

- The result of the query is an instance of the XQuery data model
  - An instance of the data model can be only a sequence
  - Items (atomic values or nodes) can exist only within sequences
- If the item is a node, it is a root of an XML tree
  - If it is **`document`**, the tree represents a whole XML document
  - If it is not **`document`**, the tree represents a fragment of XML document

XQuery Data Model

Example – XML data

```
<?xml version="1.0"?>
<catalogue>
  <book year="2002">
    <title>The Naked Chef</title>
    <author>Jamie Oliver</author>
    <isbn>80-968347-4-6</isbn>
    <category>cook book</category>
    <pages>250</pages>
    <review>
      During the past years <author>Jamie Oliver</author> has become…
    </review>
  </book>
 <book year="2007">
    <title>Blue, not Green Planet</title>
    <subtitle>What is Endangered? Climate or Freedom?</subtitle>
    <author>Václav Klaus</author>
    <isbn>978-80-7363-152-9</isbn>
    <category>society</category>
    <category>ecology</category>
    <pages>176</pages>
    <review>
      …
    </review>
  </book>
</catalogue>
```

```
element catalogue of type xs:untyped {
  element book of type xs:untyped {
    attribute year of type xs:untypedAtomic {"2002"},
    element title of type xs:untyped {
      text of type xs:untypedAtomic {"The Naked Chef"}
    },
    element author of type xs:untyped {
      text of type xs:untypedAtomic {"Jamie Oliver"}
    },
    ...
 element review of type xs:untyped {
      text of type xs:untypedAtomic {
        "During the past years "
      },
      element author of type xs:untyped {
        text  of type xs:untypedAtomic {"Jamie Oliver"}
      },
      text of type xs:untypedAtomic {
        " has become…"
      },
      ...
    },
  },
  ...
}
```

XQuery Data Model

Example – Infoset Representation

XQuery Data Model

Example – XML Schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="catalogue" type="CalalogueType" />
  <xs:complexType name="CalalogueType"
    <xs:sequence>
      <xs:element name="book" type="BookType"
                  maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="BookType">
    <xs:sequence>
      <xs:element name="title" type="xs:string" />
      <xs:element name="author" type="xs:string" />
      ...
      <xs:element name="review" type="ReviewType" />
    </xs:sequence>
    <xs:attribute name="year" type="xs:gYear" />
  </xs:complexType>
 <xs:complexType name="ReviewType" mixed="true">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="title" type="xs:string" />
      <xs:element name="author" type="xs:string" />
    </xs:choice>
  </xs:complexType>
</xs:schema>
```

```
element catalogue of type CatalogueType {
  element book of type BookType {
    attribute year of type xs:gYear {"2002"},
    element title of type xs:string {
      text of type xs:untypedAtomic {"The Naked Chef"}
    },
    element author of type xs:string {
      text of type xs:untypedAtomic {"Jamie Oliver"}
    },
    ...
 element review of type ReviewType {
      text of type xs:untypedAtomic {
        "During the past years "
      },
      element author of type xs:string {
        text  of type xs:untypedAtomic {"Jamie Oliver"}
      },
      text of type xs:untypedAtomic {
        " has become …"
      },
      ...
    },
  },
  ...
}
```

XQuery Data Model

Example – Representation (PSVI)

XQuery 3.0

- Group by clause in FLWOR expressions
- Tumbling window and sliding window in FLWOR expressions
  - Iterates over a sequence of tuples (overlapping or not)
- Expressions try / catch
- Dynamic function call
  - Function provided as a parameter
- Public / private functions
- …
- XQuery Update Facility 3.0

http://www.w3.org/TR/xquery-30/