XML Technologies

Doc. RNDr. Irena Holubova, Ph.D.

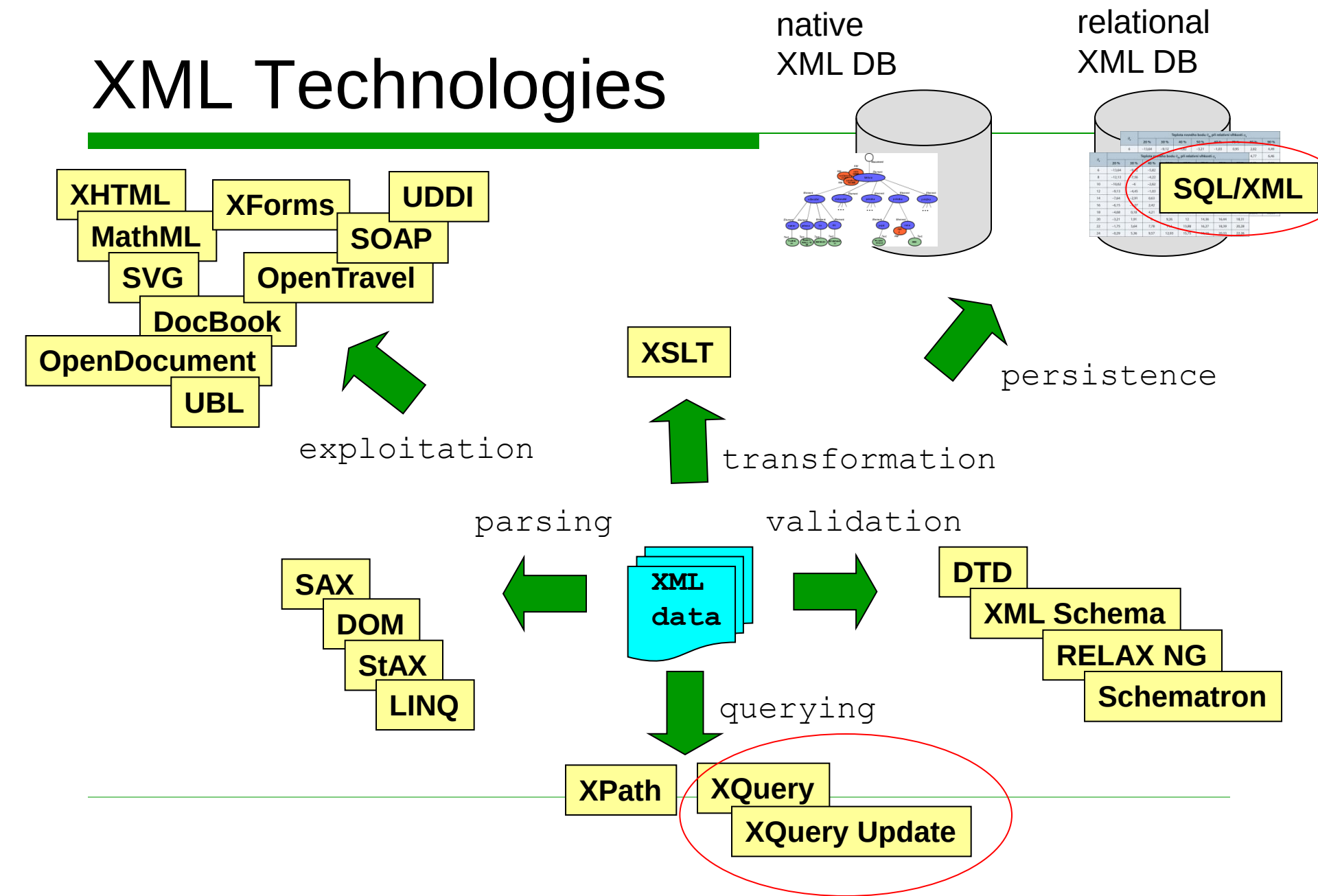
holubova@ksi.mff.cuni.cz

Web pages:
MFF: http://www.ksi.mff.cuni.cz/~holubova/NPRG036/
FEL: http://www.ksi.mff.cuni.cz/~holubova/A7B36XML/

Outline

- Introduction to XML format, overview of XML technologies
- DTD
- XML data models
- Interfaces for XML data
- XPath
- XSLT
- XQuery, XQuery Update
- XML schema languages
- SQL/XML
- An overview of standard XML formats
- XML data persistence

XML Technologies



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
```

SQL/XML

What is SQL/XML?

- Extension of SQL which enables to work with XML data
  - New built-in data type: XML
  - Publication of relational data in XML
  - XML data querying
- Node: SQL/XML ≠ SQLXML
  - Technology of Microsoft used in SQL Server
  - Not a standard from the SQL family
- Key aspect: XML value
  - Intuitively: XML element or a set of XML elements

Functions for Data Publication

- SQL expressions → XML values
  - XMLELEMENT – creating XML elements
  - XMLATTRIBUTES – creating XML attributes
  - XMLFOREST – creating XML elements for particular tuples
  - XMLCONCAT – from a list of expressions creates a single XML value
  - XMLAGG – XML aggregation

Employees

id	first	surname	dept	start
1001	Brad	Pitt	HR	2000-05-24

XMLELEMENT

- Creates an XML value for:
  - Element name
  - Optional list of namespace declarations
  - Optional list of attributes
  - Optional list of expressions denoting element content

```
SELECT E.id,
  XMLELEMENT (NAME "emp",
    E.first || ' ' || E.surname) AS xvalue
FROM Employees E WHERE ...
```

id	xvalue
1001	<emp>Brad Pitt</emp>
…	…

XMLELEMENT – Subelements

```
SELECT E.id,
  XMLELEMENT (NAME "emp",
    XMLELEMENT (NAME "name",
      E.first || ' ' || E.surname),
    XMLELEMENT (NAME "start_date", E.start)
  ) AS xvalue
FROM Employees E WHERE ...
```

id	xvalue
1001	<emp> <name>Brad Pitt</name> <start_date>2000-05-24</start_date> </emp>
…	…

XMLELEMENT – Mixed Content

```
SELECT E.id,
  XMLELEMENT (NAME "emp", 'Employee ',
    XMLELEMENT (NAME "name",
      E.first || ' ' || E.surname ), ' started on ',
    XMLELEMENT (NAME "start_date", E.start)
  ) AS xvalue
FROM Employees E WHERE ...
```

id	xvalue
1001	<emp>Employee <name>Brad Pitt</name> started on <start_date>2000-05-24</start_date></emp>
…	…

Children

id	parent
37	1001

XMLELEMENT – Subqueries

```
SELECT E.id,
  XMLELEMENT (NAME "emp",
    XMLELEMENT (NAME "name",
      E.first || ' ' || E.surname ),
    XMLELEMENT (NAME "children",
      (SELECT COUNT (*) FROM Children D WHERE D.parent = E.id ))
  ) AS xvalue
FROM Employees E WHERE ...
```

id	xvalue
1001	<emp> <name>Brad Pitt</name> <children>3</children> </emp>
…	…

XMLATTRIBUTES

- Specification of attributes can occur as
  - 3rd argument if namespace declarations are present
  - 2nd argument otherwise

```
SELECT E.id,
  XMLELEMENT (NAME "emp",
    XMLATTRIBUTES (E.id AS "empid"),
    E.first || ' ' || E.surname) AS xvalue
FROM Employees E WHERE ...
```

id	xvalue
1001	<emp empid="1001">Brad Pitt</emp>
…	…

XMLELEMENT – Namespaces

```
SELECT E.id,
  XMLELEMENT (NAME "IBM:emp",
    XMLNAMESPACES ('http://a.b.c' AS "IBM"),
    XMLATTRIBUTES (E.id AS "empid"),
    E.first || ' ' || E.surname
  ) AS xvalue
FROM Employees E WHERE ...
```

id	xvalue
1001	<IBM:emp xmlns:IBM="http://a.b.c" empid="1001">Brad Pitt</IBM:emp>
…	…

XMLFOREST

- Constructs a sequence of XML elements for
  - Optional declaration of namespaces
  - List of named expressions as arguments
    - If any of the expressions returns NULL, it is ignored
    - If all the expressions return NULL, the result is XML value NULL
- Each element in the result can be named implicitly or explicitly

XMLFOREST

```
SELECT E.id,
  XMLELEMENT (NAME "emp",
    XMLFOREST (
      E.first || ' ' || E.surname AS "name",
      E.start AS "start") ) AS xvalue
FROM Employees E
WHERE ...
```

id	xvalue
1001	<emp> <name>Brad Pitt</name> <start>2000-05-24</start> </emp>
…	…

XMLCONCAT

- Creates an XML value as a concatenation of results of multiple expressions

```
SELECT E.id,
  XMLCONCAT (
    XMLELEMENT (NAME "name",
      E.first || ' ' || E.surname),
    XMLELEMENT (NAME "start_date", E.start)
  ) AS xvalue
FROM Employees E WHERE ...
```

id	xvalue
1001	<name>Brad Pitt</name><start_date>2000-05-24<start_date>
…	…

XMLAGG

- XMLAGG is an aggregation function
  - Similar to SUM, AVG from SQL
- The argument for XMLAGG must be an XML expression
- For each row in a group G, we evaluate the expression and the resulting XML values are concatenated so that they form a single XML value for the whole group G
- For sorting we can use clause ORDER BY
- All NULL values are ignored for the concatenation
  - If all the concatenated values are NULL or the group is empty, the result is NULL

XMLAGG

```
SELECT XMLELEMENT (
  NAME "department",
  XMLATTRIBUTES (E.dept AS "name"),
  XMLAGG (
    XMLELEMENT (NAME "emp", E.surname)  ) ) AS xvalue
FROM Employees E
GROUP BY E.dept
```

xvalue
<department name="HR"> <emp>Pitt</emp> <emp>Banderas</emp> </department>
<department name="PR"> ... </department>
…

XMLAGG – Sorting

```
SELECT XMLELEMENT (
  NAME "department",
  XMLATTRIBUTES (E.dept AS "name"),
  XMLAGG (
    XMLELEMENT (NAME "emp", E.surname)
    ORDER BY E.surname ) ) AS xvalue
FROM Employees E
GROUP BY E.dept
```

xvalue
<department name="HR"> <emp>Banderas</emp> <emp>Pitt</emp> </department>
…

XML Data Type and Querying

- XML data type can be used anywhere, where SQL data types (e.g. NUMBER, VARCHAR, …)
  - Type of column, parameter of a function, SQL variable, …
  - Its value is an XML value
- XML Infoset modification: XML value is
  - XML element
  - Set of XML elements

  → Not each XML value is an XML document
- Querying:
  - XMLQUERY – XQuery query, results are XML values
  - XMLTABLE – XQuery query, results are relations
  - XMLEXISTS – testing of conditions

Example: table EmployeesXML

id	EmpXML
1001	<employee> <first>Brad</first> <surname>Pitt</surname> <start>2000-05-24</start> <department>HR</department> </employee>
1006	<employee> <first>Antonio</first> <surname>Banderas</surname> <start>2001-04-23</start> <department>HR</department> </employee>
…	…

XMLQUERY

```
SELECT
  XMLQUERY (
    'for $p in $column/employee return $p/surname',
    PASSING EmpXML AS "column"
    RETURNING CONTENT NULL ON EMPTY ) AS result
FROM EmployeesXML WHERE …
```

result
<surname>Pitt</surname>
<surname>Banderas</surname>
…

XMLTABLE

```
SELECT result.*
  FROM EmployeesXML, XMLTABLE (
    'for $p in $column/employee return $p/surname',
    PASSING EmployeesXML.EmpXML AS "column"
  ) AS result
```

result
<surname>Pitt</surname>
<surname>Banderas</surname>
…

XMLTABLE

```
SELECT result.*
  FROM EmployeesXML, XMLTABLE (
    'for $p in $column/employee return $p',
    PASSING EmployeesXML.EmpXML AS "column"
    COLUMNS name VARCHAR(40) PATH 'first'
                             DEFAULT 'unknown',
            surname VARCHAR(40) PATH 'surname'
  ) AS result
```

Xpath query

Xpath query

Assumption: We do not know the first name of Craig.

name	surname
Brad	Pitt
unknown	Craig

XMLEXISTS

```
SELECT id
FROM EmployeesXML
WHERE
  XMLEXISTS('/employee/start lt "2001-04-23"'
    PASSING BY VALUE EmpXML)
```

id
1001
1006
…

- Its argument is an XQuery expression
  - Returning true / false
- Usage: WHERE clause
- It does not extend the expressive power of the language
  - The same can be done via XMLQUERY or XMLTABLE

Other Constructs

- XMLPARSE – transforms the given SQL string value to XML value
- XMLSERIALIZE – transforms the given XML value to SQL string value
- IS DOCUMENT – tests whether the given XML value has a single root element
- …

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/