XML Databases

II. XML Query languages
II.3 XSL and XQuery

Weeks 4 - 6

Outline

II.1 Introduction
II.2 Selecting
II.3 Transforming
A. XSLT 1.0
B. XQuery 1.0

Course based on:
- The W3C specifications (XSL: november 1999, XQuery 1.0: april 2005)
- “XQuery from the experts” (E. Katz editor)

A. XSLT (eXtensible Stylesheet Language Transformations)

- W3C Recommandation (16 november 1999)
- Transforms an XML
- Usage
  - 80% XML to HTML
  - 10% XML to SVG, PDF, WMF
  - 10% XML to XML
- Subset of XSL (XSL-FO, XSLT and XPath)

A. XSLT

Main characteristics

- **XML-based syntax**
- **Declarative**, functional programming model

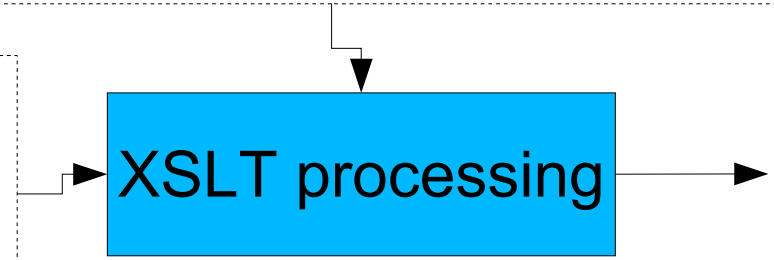
  Result tree = function(Input tree)
- **Rule based**
  - Input: pattern to be matched / Output: template-based
- **Tree to tree transformation**
- **Two language model**; XPath is the sublanguage used for: selecting nodes for processing, specifying conditions for different ways of processing a node; generating text to be inserted in the result tree.
- **Data model** ≈ XPath data model

A. XSLT

Example

```
...
<xslt:template
match="appendix">
   <h2>
      Appendix <xslt:number
format="A">
      <xslt:text>&amp;nbsp;</xslt:text>
      <a name="{@id}"/>
      <xslt:value-of
select="@title"/>
   </h2>
   <xslt:apply-templates/>
</xslt:template>
....
```

```
<appendix id="bibl"
title="Bibliography">
<para>A reference</para>
</appendix>
```



A.1. Writing XSLT: basics

Basics

- Every XSLT instruction is an XML element in the **xsl(t)** namespace (http://www.w3.org/1999/XSL/Transform)
- The **match** attribute of **xsl:template** specifies a pattern in XPath
- Elements in XML document **matching the pattern** are processed by the actions within the xsl:template element
- For elements that do not match any template
  - Attributes and text contents are output as is
  - Element copied
  - PI and comments discarded
  - Templates are recursively applied on sub-nodes

A.1. Writing XSLT: basics

Basic structure

- xsl:stylesheet
  - **Output format**: xsl:output
  - **Including other XSL files**: xsl:import, xsl:include
  - **Variables and parameters:** xsl:param, xsl:variable
  - **Space handling:** xsl:preserve-space, xsl:strip-space
  - **Namespace handling:** xsl:namespace-alias
  - **Templates:** xsl:template

A.1. Writing XSLT: basics

A template

- **XSL instruction**: xsl:template
  - Pattern based (@pattern) **or**
    - The context node **=** the matched node
  - Named based (@name)
  - In case of conflicting template rules, uses rules of priority
  - Parameters (xsl:value, xsl:param)
- **Examples**: <xsl:template match="doc">, <xsl:template match="doc/title">, <xsl:template match="doc/p[1]">

A.2. Generating content

Element and attributes

- Direct
  <h2 id=”appendix”> Appendix </h2>
- Direct with attribute value template
  <h2 id=”{@id}”> Appendix </h2>
- Indirect
  - <xsl:element **name=”...”** namespace=”...”> ... </>
  - <xsl:attribute **name=”...”** namespace=”...”> ... </>

A.2. Generating content

Other

- **Attribute set:** xsl:attribute-set
- **Text:** xsl:text
- **Processing-instruction:** xsl:processing-instruction
- **Comments:** xsl:comment
- **Copying:** xsl:copy, xsl:copy-of

A.3. Flow control and conditions
(within a template)

- **Flow control**
  - **(Recursive) processing:** xsl:apply-templates (@select), xsl:for-each (**@select**). It is possible to sort the result with xsl:sort (@select, @data-type, @order, ...)
  - **Getting text-content:** xsl:value-of
- **Conditions**
  - **IF ... THEN ...** xsl:if (**@test**)
  - **Case:** xsl:choose, xsl:when **(@test),** xsl:otherwise

A.4. Example (input)

```
<nutrition>
<daily-values>
<total-fat units="g">65</total-fat>
<saturated-fat units="g">20</saturated-fat>
<cholesterol units="mg">300</cholesterol>
<sodium units="mg">2400</sodium>
<carb units="g">300</carb>
<fiber units="g">25</fiber>
<protein units="g">50</protein>
</daily-values>

<food>
  <name>Truffles, Dark Chocolate</name>
  <mfr>Lyndon's</mfr>
  <serving units="g">39</serving>
  <calories total="220" fat="170"/>
  <total-fat>19</total-fat>
  <saturated-fat>14</saturated-fat>
  <cholesterol>25</cholesterol>
  <sodium>10</sodium>
  <carb>16</carb>
  <fiber>1</fiber>
<protein>1</protein>
  <vitamins>
    <a>0</a>
    <c>0</c>
  </vitamins>
  <minerals>
  <ca>0</ca>
  <fe>0</fe>
  </minerals>
</food>
<food>...</food>
</nutrition>
```

A.4. Example (output)

food list

- With 410 calories: Chicken Pot Pie from Swanson.
- With 370 calories: Beef Frankfurter, Quarter Pound from Armour.
- With 300 calories: Bagels, New York Style from Thomas'.
- With 220 calories: Truffles, Dark Chocolate from Lindor.
- With 200 calories: Nutella from Ferrero.
- With 160 calories: Soy Patties, Grilled from Gardenburger.
- With 150 calories: Potato Chips from Lays.
- With 110 calories: Avocado Dip from Sunnyside.
- With 70 calories: Eggs from Safeway.
- With 20 calories: Cole Slaw from Fresh Express.

B. XQuery 1.0 (and XPath 2.0)

- General purpose query language for XML data
- Currently being standardized by World Wide Web Consortium (W3C)
- **Derived** from the Quilt query language, itself based on features from XPath, XML-QL, SQL, OQL, Lorel, XQL, and YATL.
- **Defined** by several W3C drafts:
  - XQuery 1.0: An XML Query Language
  - XQuery 1.0 and XPath 2.0 Functions and Operators
  - XQuery 1.0 and XPath 2.0 Data Model
  - XQuery 1.0 and XPath 2.0 Formal Semantics

B. XQuery

Principles and Requirements

- Compositionality
  - Any result can be used as an operand
  - No side-effect
- Closure (Query data model)
- Schema conformance (primitives, definition and inheritance)
- Completeness (expressive power)
- Generality (with[out] schemas)
- Conciseness (implicit operations)
- Static analysis (“compilation” + “execution”)

B. Watershed issues

- **Handling of untyped data:** kinds of operations that can be applied to data whose type is not known.
- **Unknown and inapplicable data:** how unknown values can be encoded in XML and how various operators should be defined on these values.
- **What is a type?** how a datatype is specified in XQuery.
- **Element constructors:** how to construct a new XML element by using a query expression
- **Static typing:** kinds of type-checking that can be performed on a query independently of any input data.
- **Function resolution:** This issue deals with how functions are defined and how function calls are processed.
- **Error handling:** This issue deals with the kinds of errors that can be encountered in queries, and whether query expressions are deterministic (that is, whether they always return the same result for a given input).
- **Ordering operators:** This issue deals with the kinds of operators that are provided in the language for controlling the order of results.

B. XQuery

Examples

- find all accounts with balance > 400, with each result enclosed in an <account-number> tag

```
for      $x in /bank/account
 let      $acctno := $x/@account-number
where $x/balance > 400
return <account-number>{$acctno}</account-number>
```

- Note: can be written as:

```
for $x in /bank/account[balance>400]
return <account-number>{$x/@account-number}
   </account-number>
```

B.1. Guided Tour

XQuery Data Model

- Sequence = series of items
  - Concating sequences with **,**
  - Creating integer sequences with **to**
- An item
  - Nodes as XPath 1.0
  - Atomic values from XML Schema
- Same document order as Xpath

B.1. Guided Tour

Literals and comments

- Comments

(: Thanks, Jeni! :)

- Numbers

1, 2.3, 1e+5

- Strings

“quotation mark delimited string”
'quote mark delimited string'

B.1. Guided Tour

Locating nodes

- Input functions

fn:doc(“mydoc.xml”)
fn:collection(“urn:Shakespeare”)

- Path expressions ≈ extended version of Xpath 1.0

fn:doc("books.xml")/bib/book/(author | editor)

B.1. Guided Tour

Creating nodes

- Direct

<a>...

- Using a constructor

document { expr }
element name { expr }
element {expr} { expr }
attribute name { expr }
attribute {expr} { expr }
text { expr }
...

B.1. Guided tours

FLOWR expression

- FLWOR ("flower") expression is constructed from

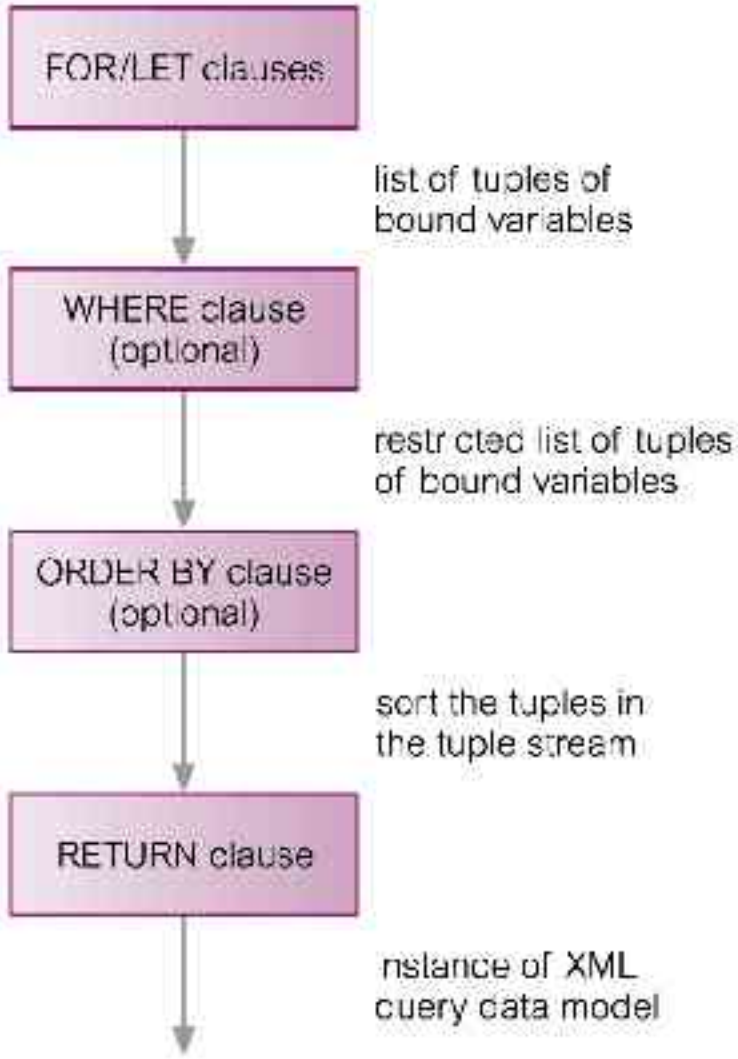
  FOR/LET,

  WHERE,

  ORDER BY,

  RETURN

  clauses.



B.1. Guided tours

FLOWR expression

- FOR and LET bind values to one or more variables using expressions (e.g., path expressions).
  - FOR iterates over items returned by its respective expression
  - LET binds a variable to one or more expressions without iteration (single binding for each variable).
- ORDER BY clause determines order of the tuple bindings.
- WHERE specifies one or more conditions to restrict nodes generated by FOR and LET.
- RETURN evaluated once for each bindings and results concatenated to form result.

B.1. Guided tours

Operators

- Two central operations
  - Typed-value extraction
  - Atomization
- Operators
  - Arithmetic (+, -, *, div, idiv, and mod)
  - Comparison
    - General comparison operator (=, !=, <, <=, >, >=)
    - Value comparison (eq, ne, lt, le, gt, le)
    - Node comparison (<<, >>, is, is not)
    - Sequence operators (“,”, to, union, intersect, except)

B.1. Guided tours

Functions, modules

- Built-in functions (fn namespace)
  - min, max, count, sum, etc.
  - distinct-values(), empty(), exists()
- User defined

```
declare function local:books-by-author($last, $first)  as
element()*
{
  for $b in doc("books.xml")/bib/book
  where some $ba in $b/author satisfies
        ($ba/last=$last and $ba/first=$first)
  order by $b/title
  return $b/title
};
```

B.1. Guided tours

Functions, modules (II)

- Global variables

```
define variable $titles { doc("books.xml")//title }
```

- Library modules

```
module "http://example.com/xquery/library/book"
define function toc($book-or-section as element())  as element()*  {
  for $section in $book-or-section/section
  return
    <section>{ $section/@* , $section/title , toc($section) } </section>
}
```

- External function and variables

B.1. Guided tour

Type

- Types based on XML Schema
  - XQuery basic types: xdt:untyped, xdt:untypedAtomic, xdt:anyAtomicType
  - Built-in schema types (xs:string, xs:integer, ...)
  - A type = a sequence of expanded Qname
- Some functions: “instance of”, “cast as”, “castable as”, “treat as”
- Examples
  - for $i as xs:integer in ...
  - declare function local:a as xs:string ...

B.1. Guided tour

Type (II)

- Step I: Item type
  - Kind test
  - Atomic type
- Step 2: Sequence
  - ?, * or +
  - xs:integer?
  - element(*,xs:string)*

B.1. Guided tour

Some structures

- **Conditions**

if (Expr) then ExprSingle else ExprSingle

- **Switch on type**

typeswitch (Expr)
  case $varname as Type return ExprSingle
  ...
  default $varname return ExprSingle

- **Quantified expression**

some/every $varname in Expr [, $varname in Expr]
satisfies ExprSingle

B.2. Formal Semantics

- Clarifies the intended meaning of the English specification
- Ensures that no corner cases are left out
- Provides a reference for implementation
- Three components
  - Dynamic semantics
  - Static semantics
  - Normalization rules

B.2. Formal Semantics

Notations

- Grammar production
  (ForClause | LetClause) "return" ExprSingle
- Judgements

Painting **is** beautiful, Expr => Value, Expr : Type,
Expr **raises** Error, Expr **matches** Type

- Inference Rule

$$\frac{\text{premise}_1 \ \ldots \ \text{premise}_n}{\text{conclusion}}$$

- Environment
  env |- judgment