8 Querying XML Data and Documents

- **XQuery, W3C XML Query Language**
  - “work in progress”, Working Draft, 15/11/2002
  - joint work by XML Query and XSL Working Groups
  - influenced by many research groups and query languages
    - QL: XPath, XSL, XML-QL, SQL, QL, QLrel, ...
  - Goal: a query language applicable to any XML-represented data, both documents and databases

Functional Requirements (1)

- Support operations (selection, projection, aggregation, sorting, etc.) on all data types:
  - Choose parts of data based on content or structure
  - Also operations on document hierarchy and order
- Structural preservation and transformation:
  - Preserve the relative hierarchy and sequence of input document structures in the query results
  - Transform XML structures and create new XML structures
- Combining and joining:
  - Combine related information from different parts of a given document or from multiple documents

Functional Requirements (2)

- Cross-references:
  - Ability to traverse intra- and inter-document references
- Closure property:
  - The result of an XML query is also XML (usually not a valid document, but a well-formed document fragment)
  - > results can be used as an input to another query
- Extensibility:
  - The query language should support the use of externally defined functions on all data types of the data model

XQuery in a Nutshell

- Functional expression language
- **Strongly-typed**: (XML Schema) types may be assigned to expressions statically
  - Includes XPath 2.0 (day 1 Draft, but not all XPath axes included!)
  - XQuery 1.0 and XPath 2.0 share extensive functionality:
    - XQuery 1.0 and XPath 2.0 Functions and Operators, WD 15/11/2002
- Roughly: XQuery = XPath' + XSLT' + SQL'

XQuery: Basics

- A query is represented as an expression
- Expressions operate on, and return sequences of
  - atomic values (of XML Schema simple types) and
  - nodes (of the 7 node types of XPath/XSLT)
  - an item = a singleton sequence
  - sequences are flat: no sequences as items
  - sequences are ordered, and can contain duplicates
  - Unlimited combination of expressions, often with automatic type conversions (e.g. for arithmetics)

- Query applied to an input sequence, accessed through functions ...
  - `input()` → implementation-dependent input sequence
  - `document("URl")` → root of the XML document available at URI
  - `same as in XSLT`
  - `collection("URl")` → sequence of nodes from URI
Central XQuery Expressions

- Path expressions
- Sequence expressions
- Element constructors (= XSLT templates)
- FLWOR expressions
  - (forall, for-each, where, order by, return)
- Quantified expressions (some/every ... satisfies ...)
- Expressions for typing and validation

Path Expressions

- Syntax of XPath:
  - arbitrary (node-sequence-valued) expressions as steps
  - only 6 of 13 XPath axes: child, descendant, attribute, self, descendant-or-self, parent
  - with document-order operators (<<, >>) sufficient for expressing queries
  - produce ordered sequence of nodes in document order, without duplicate nodes

Sequence Expressions

- Constant sequences constructed by listing values
  - comma (,) used as concatenation operator
  - (1, 2, 3), (1, 1) = (1, 2, 3, 1)
- Also dynamically generated:
  - (<start />, Temp/children, <end/>)
- Shorthands for numeric sequences:
  - 1 to 4 -> 1, 2, 3, 4
  - 6 to 3 -> 6, 5, 4, 3

Combining (Node) Sequences

- Assume variable bindings:
  s1 = 1, 2, 3, 4, 5, 6
  s2 = 1, 2, 3
- Then:
  $s1 \cup s2 = 1, 2, 3, 4, 5, 6, 7, 8, 9, 2
  $s1 \cap s2 = 1, 2, 3
  $s1 \setminus s2 = 4, 5, 6, 7, 8, 9

Element Constructors

- Similar to XSLT templates:
  - start and end tag enclosing the content
  - literal fragments written directly, expressions enclosed in braces { and }
- Often used inside another expression that binds variables used in the element constructor
- See next

Example

- Generate an emp element containing an emp-id attribute and nested name and job elements;
  values of the attribute and nested elements given in variables $id, $n, and $j:
  
  `<emp emp-id="\$id">` 
  `<name>`$n`</name>`
  `<job>`$j`</job>`
  `</emp>`
FLWOR ("flower") Expressions

- Constructed from for, let, where, order by and return clauses (~SQL select-from-where)
- Form: $(\text{ForClause} \mid \text{LetClause})^+ \\text{WhereClause}^? \\text{OrderByClause}^? \\text{return Expr}$
- FLWOR binds variables to values, and uses these bindings to construct a result (an ordered sequence of nodes)

Flow of data in a FLWOR expression

for clauses

- for $SV_i \in \text{Exp}_1, (\text{Exp}_2, ...)$
  - associates each variable $V_i$ with expression $\text{Exp}_i$ (e.g. a path expression)
- Result: list of tuples, each containing a binding for each of the variables
- can be thought of as loops iterating over the items returned by respective expressions

Example: for clause

```
for $i$ in (1,2),
   $j$ in (1 to $i$)
return <tuple>
   <i>{$i}</i><j>{$j}</j></tuple>
```

Result:
```
<tuple><1><1></tuple>
<tuple><1><2></tuple>
<tuple><2><2></tuple>
```

let clauses

- let also binds variables to expressions
  - each variable gets the entire sequence as its value (without iterating over the items of the sequence)
  - results in binding a single sequence for each variable
- Compare:
  - for $x$ in /library/book
  - let $x := /library/book$
  - single binding (to a sequence of books)

for/let clauses

- A FLWOR expr may contain several fors and lets
  - each of these clauses may refer to variables bound in previous clauses
- the result of the for/let sequence:
  - an ordered list of tuples of bound variables
  - number of tuples = product of the cardinalities of the node-lists returned by the expressions in the for clauses
Example: let clauses

```xml
let $s := <one/>, <two/>, <three/>
return <out>{$s}</out>
```

Result:
```
<out/>
<one/>
<two/>
<three/>
</out>
```

where clause

- Variables bound by a for clause represent a single item
- scalar predicates, e.g. `$color = "Red"`
- Variables bound by a let clause may represent lists of nodes
- list-oriented predicates, e.g. `$avg[$prices] > 100`
- a number of aggregation functions available: `avg[()], sum[()], count[()], max[()], min[()]`
  (also in XPath 1.0)

return clause

- The return clause generates the output of the FLWOR expression
- instantiated once for each binding tuple
- often contains element constructors, references to bound variables, and nested subexpressions

Examples (from "XML Query Use Cases")

- Assume: a document named "bib.xml" containing a list of books:
  ```xml
  <book>
    <title/>
    <author/>
    <year/>
    <price/>
  </book>
  ```

List the titles of books published by Morgan Kaufmann after 1998

```xml
<recent-MK-books> {
  for $b in document("bib.xml")/book
  where $b/publisher = "Morgan Kaufmann"
    and $b/year > 1998
  return <book year="#[year]">[b/title]"</book>
} </recent-MK-books>
```
Result could be...

```xml
<recent-MX-books>
  <book year="1999">
    <title>TCP/IP Illustrated</title>
  </book>
  <book year="2000">
    <title>Advanced Programming in the Unix environment</title>
  </book>
</recent-MX-books>
```

Invert the book-list structure

```xml
<author_list>| [ -- group books by authors --> ]
  for $a in distinct-values(document("bib.xml")/author)
    return <author>
      <name>$a/text()$</name>,
      for $b in document("bib.xml")/book[
        author = $a
      ]
        return $b/title
    </author>
| </author_list>
```

Queries on Document Order

- Operators `<<` and `>>`:
  - `x << y` = true if node `x` precedes node `y` in document order, `y >> x` similarly.
- Consider a surgical report with elements
  - procedure, incision and anesthesia
- Return a "critical sequence" consisting of contents between the 1st and 2nd incisions of the 1st procedure

List publishers with the average price of their books

```xml
for $p in distinct-values(document("bib.xml")/publisher)
let $a := avg(document("bib.xml")/book[
  publisher = $p]/price)
return <publisher>
  <name>$p/text()$</name>,<avgprice>($a)</avgprice>
</publisher>
```

Make an alphabetic list of publishers, for each publisher, list books (title & price) in descending order of price

```xml
for $p in distinct-values(document("bib.xml")/publisher)
order by $p
return <publisher>
  (<name>$p/text()$</name>)
  for $b in document("bib.xml")/book[
    order by price descending
  ]
    return <book>$b/title</book>
</publisher>
```

Computing a "critical sequence"

```xml
<critical_sequence>
  let $p := //procedure[1]
  for $n in $p/node()
    where $n >> ($p/inclusion)[1] and
    $n << ($p/inclusion)[2]
      return $n
| </critical_sequence>
```

- NB: if incisions are not children of the procedure, then some ancestors of the second incision are included in the result. How to avoid this?
User defined functions: Example

```
define function
  precedes($a as node, $b as node) as boolean
  ($a << $b and $b not ancestor of $a) or not($b/ancestor-or-self::*).is($a))
```

Now, "critical sequence" without ancestors if incision:
```
<critical_sequence>
  let $p := //procedure[1]
  for $n in $p/node()
    where $n << ($p/incision)[1] and
    precedes($n, ($p/incision)[2])
  return $n
</critical_sequence>
```

Recursive transformations

```
Example: Exclude all but sections and titles (with content) from a document:

define function
  sectAndTitles($n as node) as node?
  if (name($n)="sect")
    then <sect> {
      for $c in $n/* return sectAndTitles($c)
    </sect>
  else if (name($n)="title")
    return $n
  else -- check children, if any: --
    for $c in $n/* return sectAndTitles($c) }
```

Querying relational data

- Lots of data is stored in relational databases
- Should be able to access also them
- Example: Tables for parts and suppliers
  - P(pno, descr) part numbers and descriptions
  - S(pno, sname) supplier numbers and names
  - SP(pno, pno, price)
    who supplies which parts for what price?

Possible XML representation of relations

```
<Relational data>
  <S SNO DNAME>
    <pno descr>...</pno descr>
  </S>
  <P PNO DESCRIPT>
    <sno descr>...</sno descr>
  </P>
  <SP SNO PNO PRICE>
    <sno descr>...</sno descr>
  </SP>
</Relational data>

SQL vs. XQuery

- SQL:
  ```
  SELECT pno
  FROM P
  WHERE descr LIKE 'Gear'
  ORDER BY pno;
  ```

- XQuery:
  ```
  for $p in document("p.xml")/p_tuple
  where starts-with($p/descr, "Gear")
  order by $p/pno
  return $p/pno
  ```

Grouping

- Many relational queries involve forming data into groups and applying some aggregation function such as count or avg to each group
- in SQL GROUP BY and HAVING clauses
- Example: Find the part number and average price for parts that have at least 3 suppliers
Grouping: SQL

```sql
SELECT pno, avg(price) AS avgprice
FROM sp
GROUP BY pno
HAVING count(*) >= 3
ORDER BY pno;
```

Grouping: XQuery

```xquery
for $pm in distinct-values:
  document("sp.xml")//sp_tuple
let $sp=doc("sp.xml")//sp_tuple[pno=$pm]
where count($sp) >= 3
order by $pm
return <well_supplied_item>
  <sp>
    <avgprice(avg=avg(price))</avgprice>
  </sp>
</well_supplied_item>
```

Joins

- Example: Return a 'flat' list of supplier names and their part descriptions, in alphabetical order

```xquery
for $p in document("sp.xml")//sp_tuple,
   $p in document("p.xml")//p_tuple|
   pno = $sp/pno,
   $s in document("s.xml")//s_tuple|
   sno = $sp/sno|
order by $p/desc, $s/name
return <sp_part>
  <s/name/>
  <p/desc/>
</sp_part>
```

XQuery vs. XSLT

- Could we express XQuery queries with XSLT?
  - In principle yes, always, (but could be tedious)
- Ex: Partial XSLT simulation of FLWOR expressions

```
XQuery: for $x in Expr...
   can be expressed in XSLT as:

   <xsl:for-each select="Expr">
     <xsl:variable name="x" select="."/>
     ...
   </xsl:for-each>
```

XQuery vs. XSLT

```
XQuery: let $y := Expr...
   corresponds directly to:
   <xsl:variable name="y" select="Expr" />

and where Condition... can be expressed as:
   <xsl:if test="Condition">
     ...
   </xsl:if>
```

XQuery vs. XSLT

```
XQuery: return ElemConstructor
   can be simulated with a corresponding XSLT template:
   - static fragments as such
   - enclosed expressions in element content, e.g.
     ($x/name) become
     <xsl:copy-of select="$/name"/>
```
XQuery vs. XSLT: Example

XQuery: for $b in document("bib.xml")//book where $b/publisher = "NI" and $b/year > 1950
return <book year="($b/year)"
     ($b/title) </book>

XSLT: template match=""{
  <xsl:for-each select="document("bib.xml")//book"
     <xsl:variable name="B" select="." />
     <xsl:if test="$b/publisher="NI" and $b/year > 1950"
       <book year="($b/year)
           ($b/title) </book>
     </xsl:if>
   </xsl:for-each> </xsl:template>

XSLT for XQuery FLWOR Expressions

- NB: The sketched simulation is not complete:
  - Only two things can be done in XSLT for result tree fragments produced by templates:
    - insertion in result tree
      - with <xsl:copy-of select="$X" />
    - with <xsl:value-of select="$X" />
  - Not possible to apply other operations to results (like, e.g., sorting in XQuery):
    for $y in <a><xsl:copy-of select="$y"></a>
    order by <y>

XQuery: Summary

- XQuery
  - W3C working draft of an XML query language
  - vendor support?
    - http://www.w3.org/XML/Query
      - mentions about 20 prototype implementations or products that support (or will support) some version of XQuery
  - future?
  - promising confluence of document and database research