8 Querying XML

- How to access different sources (DBs, docs) as XML?
- **XQuery**, W3C XML Query Language
  - "work in progress", (last call) Working Draft, 12/11/03
  - joint work by XML Query and XSL Working Groups
    - with XPath 2.0
    - influenced by many research groups and query languages
      - Quilt, XPath, XQL, XML-QL, SQL, OQL, Lorel, ...
    - 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. (We'll mainly ignore this)
- Includes XPath 2.0 (says 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 12/11/2003
- 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
  - (1, 2) = (1, 2), (1, 2) ≠ (1, 3, 2)
  - sequences are ordered, and can contain duplicates
- Unlimited combination of expressions, often with automatic type conversions (e.g. for arithmetics)

XQuery: Accessing Documents

- XQuery operates on nodes accessible by input functions
  - doc("URI")
  - document-node of the XML document available at URI
    - same as document("URI") in XSLT
  - collection(URI)
  - sequence of nodes from URI

Central XQuery Expressions

- Path expressions
- Sequence expressions
- Element constructors (~ XSLT templates)
- FLWOR expressions
  - "flower": for-let-where-order by-return
- Quantified expressions
  - (some/every X... in ... satisfies ...)
- Expressions for typing and validation

Path Expressions

- Similar to XPath:
  - [Expr] /
    - arbitrary (node-sequence-valued) expressions as steps
    - only 6 (of 13 XPath) axes: child, descendant, attribute, self, descendant-or-self, parent
    - others (except for namespace) available if an optional Full Axis Feature supported
    - 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 catenation operator
    - (1, 2, 3, 4, 5) = (1, 2, 3, 4, 5)
- Also dynamically generated:
  - (1, 2, 3, 4, 5) = (1, 2, 3, 4, 5)
- Shorthands for numeric sequences:
  - 1 to 4 => (1, 2, 3, 4)
  - 4 to 1 => ()
  - reverse({1 to 4}) => (4, 3, 2, 1)

Combining (Node) Sequences

- Assume variable bindings:
  - $s1 = (...) (...) (...) (...) (...) $s2
- Then:
  - $s1 union $s2
  - $s1 intersect $s2
  - $s1 except $s2

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 empid attribute and nested name and job elements; values of the attribute and nested elements given in variables $id, $n, and $j:
  ```xml
  <emp empid="$id">
    <name>$n</name>
    <job>$j</job>
  </emp>
  ```

Flow of data in a FLWOR expression

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

Example: for clause

- for $i in (1,2), $j in (1 to $i)
  - return <tuple> <i>$i</i> <j>$j</j> </tuple>
- Result:
  ```xml
  <tuple><1><1></tuple>
  <tuple><2><1></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 $b$ in /library/book
    -> many bindings (books)
  - let $b1 := /library/book
    -> single binding (to a sequence of books)

Examples (modified from "XML Query Use Cases")

Example: let clauses

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

Result:

```xml
<out>
  <one/>
  <two/>
  <three/>
</out>
```

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 (monikkoi) of bound variables
  - number of tuples = product of the cardinalities of the node-lists returned by the expressions in the for clauses

where clause

- Variables bound by a for clause represent a single item
  -> value comparisons, e.g. $color eq "red"
- Variables bound by a let represent sequences
  -> general comparisons, e.g. $colors = "red"
  (is the string value of some node red?)
  - a number of aggregation functions available:
    - $avg()$, $sum()$, $count()$, $max()$, $min()$
    (also in XPath 1.0)

Examples (modified from "XML Query Use Cases")

Example: for + return

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

Result:

```xml
<tuple><i>1</i><j>1</j></tuple>
<tuple><i>2</i><j>1</j></tuple>
```

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 (modified from "XML Query Use Cases")

- Assume: a document named "bib.xml" containing a list of books:
  ```xml
  <book>*
    <title>
    <author>*
    <publisher>
    <year>
    <price>
  ```
List Morgan Kaufmann book titles since 1998

```xml
<recent-MK-books> { 
  for $b in doc("bib.xml")/book 
  where $b/publisher = "Morgan Kaufmann" 
  and $b/year >= 1998 
  return <book year="{$b/year}">
    <title>{$b/title} </title>
  </book>
} </recent-MK-books>
```

Result could be...

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

Publishers with avg price of their books:

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

Invert the book-list structure

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

List of publishers alphabetically, and their books in descending order of price

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

Queries on Document Order

- Operators `<<` and `>>`:
  - `x << y = true iff node x precedes node y in document order; (y >> x similarly)`
- Consider a surgical report with
  - `procedure elements containing incision sub-elements`
- Return a "critical sequence" of contents between the first and the second incisions of the first procedure

Computing a "critical sequence"

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

User-defined functions: Example

```xml
declare function precedes($a as node(), $b as node()) as boolean
{ $a << $b and (: $a is no ancestor of $b: :) 
  empty($a/node() . i. is $b) }
```

- Now, "critical sequence" without ancestors of incision:
```xml
<critical_sequence> { 
  let $p := //procedure[1] 
  for $n in $p/node() 
  where $n >> ($p/incipion)[1] and 
  precedes($n, ($p/incipion)[2]) 
  return $n 
} </critical_sequence>
```
Recursive transformations

- Example: Table-of-contents for nested sections

```
declare function 
sectsAndTitles($n as element()) as element() 
{ 
  if (name($n)="sect") 
  then <sect> { 
    for $c in $n/* return sectsAndTitles($c) 
  } </sect> 
  else if (name($n)="title") then $n 
  else (: check children, if any: :) 
  for $c in $n/* return sectsAndTitles($c) } 
```

SDPL 2004
Notes 8: XQuery
33

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, descrip): part numbers and descriptions
  - S (sno, name): supplier numbers and names
  - SP (sno, pno, price): who supplies which parts for what price?

SDPL 2004
Notes 8: XQuery
34

Possible XML representation of relations

<table>
<thead>
<tr>
<th>Relational data</th>
<th>XML representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>$n/NAME</td>
</tr>
<tr>
<td>P</td>
<td>$n/DESCRIPTION</td>
</tr>
<tr>
<td>SP</td>
<td>$n/PRICE</td>
</tr>
</tbody>
</table>

SDPL 2004
Notes 8: XQuery
35

SQL vs. XQuery

- SQL: SELECT pno 
  FROM S 
  WHERE descrip LIKE ‘Gear’ 
  ORDER BY pno;
- XQuery: 
  for $p in doc("p.xml")/p_tuple 
  where starts-with($p/descrip, "Gear") 
  order by $p/pno 
  return $p/pno

SDPL 2004
Notes 8: XQuery
36

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

SDPL 2004
Notes 8: XQuery
37

Grouping: SQL

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

SDPL 2004
Notes 8: XQuery
38

Grouping: XQuery

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

SDPL 2004
Notes 8: XQuery
39

Joins

- Example: Return a ‘flat’ list of supplier names and their part descriptions, in alphabetic order

```
for $sp in doc("sp.xml")/sp_tuple, 
  $p in doc("p.xml")/p_tuple[
    pno = $sp/pno], 
  $s in doc("s.xml")/s_tuple[
    sno = $sp/sno] 
order by $sp/descrip, $s/name, 
$p/descrip 
return <sp_pair>(
  $s/name, 
  $p/descrip 
)</sp_pair>
```

SDPL 2004
Notes 8: XQuery
40
In principle

Only two things can be done in XSLT for Vendor support?

– In principle yes, always, (but could be tedious)

– Ex: Partial XSLT simulation of FLWOR expressions

XQuery: for $x in Expr ...rest of query

can be expressed in XSLT as:

    <xsl:for-each select="Expr">
        <xsl:variable name="x" select="." />
        ... translation of the rest of the query
    </xsl:for-each>

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="$x" />
  and conversion to a string
  » with <xsl:value-of select="$x" />

– Not possible to apply other operations to results (like, e.g., sorting in XQuery):
  for $y in <a><key>{$x/code}</key></a>
  order by $y/key

XQuery: let $y := Expr ...
corresponds directly to:

<xsl:variable name="y" select="Expr" />

and where Condition ... rest

→

<xsl:if test="Condition">
    ... translation of the rest
</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.
  {$s/sname} become

    <xsl:copy-of select="$s/sname" />

– W3C XML query language (draft), also capable of
– General XML processing
– Vendor support??
  » http://www.w3.org/XML/Query
  mentions about 30 prototype implementations or
  products that support (or will support) some version
  of XQuery
– Future?? Interesting confluence of document and
  database research, and likely potential for XML-based
  data integration