7 Querying XML

- How to access different sources (DBs, docs) as XML?
- XQuery, W3C XML Query Language
  - “work in progress”, (last call) Working Draft, Apr’05
  - joint work by XML Query and XSL Working Groups
  - influenced by many research groups and query languages
  - Quilt, 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

XQuery in a Nutshell

- Functional expression language
- Strongly-typed: (XML Schema) types may be assigned to expressions statically. (We’ll mainly ignore typing)
  - predeclared prefix for type names:
    - xs=http://www.w3.org/2001/XMLSchema
- Extends XPath 2.0 (but not all axes required!)
  - common for XQuery 1.0 and XPath 2.0
- Functions and Operators, W3C WD 4/4/2005
- Roughly: XQuery ≈ XPath’ + XSLT’ + SQL’

XQuery: Basics

- A query is 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, 3), (4, 5)) ≠ (1, 2, 3, 4)
  - 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
  - fn:doc(“URI”) (same as document(“URI”) in XSLT 1.0)
  - fn:collection(“URI”) (sequence of nodes from URI)
  - predeclared prefix for the default function namespace:
    - fn=http://www.w3.org/2005/04/xpath-functions

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]/...]/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 an ordered sequence of nodes in document order, without duplicates
Sequence Expressions

- Constant sequences constructed by listing values
  - comma (,) used as catenation operator
    - $\{1, 2, 3, 4\} = (1, 2, 3, 4)$
  - Also dynamically generated:
    - $\langle\text{start} \rangle$, $\langle\text{emp}/\text{child}\rangle$, $\langle\text{end} \rangle$

- Shorthands for numeric sequences:
  - 1 to 4 $\rightarrow (1, 2, 3, 4)$
  - 4 to 1 $\rightarrow ()$
  - $\text{fn:reverse(1 to 4)} \rightarrow (4, 3, 2, 1)$

Element Constructors

- Similar to XSLT templates:
  - start and end tag enclosing the content
  - literal fragments written directly, expressions enclosed in braces (and )
  - cf. XSLT 1.0 attribute value templates

- Shorthands for numeric sequences:
  - 1 to 4 $\rightarrow (1, 2, 3, 4)$
  - 4 to 1 $\rightarrow ()$
  - $\text{fn:reverse(1 to 4)} \rightarrow (4, 3, 2, 1)$

- Often used inside another expression that binds variables used in the element constructor
  - See next

FLWOR ("flower") Expressions

- 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)

Set Operations on Node (!) Sequences

- Assume variable bindings:
  - $s1 ---- \{ 1, 1, 2, 2, 3, 3 \} \rightarrow \{ 1, 2, 3, 1 \}$

- Then:
  - $s1 \text{ union } s2 = \{ 1, 1, 2, 2, 3, 3, 4 \}$
  - $s1 \text{ intersect } s2 = \{ 1, 2, 3 \}$
  - $s1 \text{ except } s2 = \{ 4 \}$

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 $\{id, n, j\}$:

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

Flow of data in a FLWOR expression

- **FOR/LET Clauses**
  - Ordered list of tuples of bound variables
- **WHERE Clause**
  - Pruned list of tuples of bound variables
- **RETURN Clause**
  - Instance of XML Query data model

Example: for clause

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

Result:

```
$<$tuple$>$i1</i>$</j>1$</j></tuple$>$
$<$tuple$>$i2</i>$</j>1$</j></tuple$>$
$<$tuple$>$i2</i>$</j>2$</j></tuple$>$
$<$tuple$>$i2</i>$</j>2$</j></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 $sbl := /library/book$
    -> single binding (to a sequence of books)

for/let clauses

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

where clause

- for binds variables to single items
  -> value comparisons, e.g. $\text{color} \equiv \text{"red"}$
- let to whole sequences --> general comparisons, e.g. $\text{colors} \equiv \text{"red"}$
  (~ some \text{c} in \text{colors}
   satisfies \text{c} \equiv \text{"red"})
  - a number of aggregation functions available:
    \text{avg()}, \text{sum()}, \text{count()}, \text{max()}, \text{min()}
    (also in XPath 1.0)

Example: let clauses

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

Result:
```
<out>
<\text{one}/>
<\text{two}/>
<\text{three}/>
</out>
```

where clause

- binding tuples generated by for and let
- let clauses are filtered by an optional where clause
  - tuples with a true condition are used to instantiate the return clause
- the where clause may contain several predicates
  connected by and, or, and fn:not()
  - usually refer to the bound variables
  - sequences treated as Booleans similarly to node-lists

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

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

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

Examples (modified from “XML Query Use Cases”)

- Assume: a document named "bib.xml" containing of a list of books:

```
<book>
  <title>
  <author>
  <publisher>
  <year>
  <price>
```

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

Result:
```
<tuple><i>1</i></tuple><j>1</j></tuple>
<tuple><i>2</i></tuple><j>2</j></tuple>
```
List Morgan Kaufmann book titles since 1998

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

Publishers with avg price of their books:

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

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

<author_list> { 
    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 $b/price desc
    return <book>
        {$b/title, $b/price}
    </book>
} </publisher>

Invert the book-list structure

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

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"

<incision> { 
    let $p := doc("report.xml")/procedure[1]
    for $n in $p/node()
        where $n/($p/incision)[1] and $n << ($p/incision)[2]
        return $n
} </incision>

User-defined functions: Example

declare function local:precedes($a as node(), $b as node()) as xs:boolean
{ $a << $b and (not $a is no ancestor of $b: .) empty($a/node()[. is $b]) }

- Here local: is predeclared prefix for a namespace of local function names
- Alternatively:
  declare namespace my=http://my.namespace.org;
  declare function my:precedes(... as above)
User-defined functions: Example

Now, "critical sequence" without ancestors of incision:

<critical_sequence> {
    let $p := doc("report.xml")//procedure[1]
    for $n in $p/node()
        where $n >> ($p//incision)[1] and
            local:precedes($n, ($p//incision)[2])
        return $n
} </critical_sequence>

SQL vs. XQuery

- SQL: 
  SELECT pno
  FROM p
  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

Grouping: SQL

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

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?

Recurse transformations

- Example: Table-of-contents for nested sections

    declare function local: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 {
            let $s := doc("report.xml")
                for $p in $s/node()
                    where $p//incision
                order by count($p) >= 3
                order by $p
            return $s;
        }
    }

Possible XML representation of relations

<table>
<thead>
<tr>
<th>Relational data</th>
<th>XML representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>&lt;sno sname&gt;*</td>
</tr>
<tr>
<td>P</td>
<td>&lt;pno descrip&gt;*</td>
</tr>
<tr>
<td>SP</td>
<td>&lt;sno pno price&gt;*</td>
</tr>
</tbody>
</table>

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: XQuery

for $p in distinct-values{
    doc("sp.xml")//sp_tuple
    let $sp := doc("sp.xml")//sp_tuple[pno=$p]
    where count($sp) > 3
    order by $p
} return <well_supplied_item> {
    <pno>{$p} </pno>,
    <avgprice> { avg($sp//price) } </avgprice>
} <well_supplied_item>
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],
  $x$ in doc("x.xml")/x_tuple[   
    sno = $sp/sno]
  order by $sp/descrip, $s/sname
  return <sp_pair>({   
    $s/sname , 
    $sp/descrip 
  })</sp_pair>

XQuery vs. XSLT 1.0

- 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 ... rest of query
  can be expressed in XSLT as:

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

XQuery vs. XSLT 1.0

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 1.0: Example

XQuery: for $b$ in doc("bib.xml")/book where
  $b/publisher = "HK" and $b/year > 1998
return <book year="{$b/year}">
  {$b/title}</book>

<xs:template match="/"/>
<xsl:for-each select="document('bib.xml')/book">
  <xsl:variable name="b" select="." />
  <xsl:if test="$b/publisher='MK' and $b/year > 1998">
    <book year="{$b/year}">
      <xsl:copy-of select="$b/title" /></book>
  </xsl:if>
</xsl:for-each> </xs:template>

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

- W3C XML query language (draft), also capable of general XML processing
- Vendor support??
  - http://www.w3.org/XML/Query
    mentions about 40 prototypes or products that support (or will support) some version of XQuery (30 in Apr: '04)
- Future?? Interesting confluence of document and database research, and highly potential for XML-based data integration