7 Querying XML

- **How to access various XML data sources?**
  - XQuery, XML Query Lang, W3C Rec, Jan '07
  - joint work by XML Query and XSLT WGs
    - with XPath 2.0 and XSLT 2.0
  - influenced by many research groups and query languages
    - Quilt, XPath, SQL, OQL, Lorel, ...
  - A query language for any XML-represented data: both documents and databases

**Functional Requirements**

- **Closure property:**
  - Results of XML queries are also XML
    - well-formed document fragments
  - > queries can be combined without limit

- **Extensibility:**
  - should support externally defined functions on all data types of the data model

**XQuery in a Nutshell**

- **Functional expression language (lausekekieli)**
  - Strongly-typed: (optional) type-checking of expressions, and validation of results (We’ll concentrate to processing)
    - predeclared prefix for type names: `xs=http://www.w3.org/2001/XMLSchema`
  - Extends XPath 2.0
    - XQuery 1.0 and XPath 2.0 Functions and Operators, Rec. Jan. 2007
  - XQuery = XPath 2.0 + XSLT' + SQL' (roughly)

Example Query

```xquery
<xquery version="1.0">
  <cheapBooks>
    <Title>Cheap Books</Title>
    [ for $b in fn:doc("bib.xml")/book[price < 50] order by $b/title return $b ]
  </cheapBooks>
</xquery>
```

- **Syntax "conceivable and easily understood"**
- **XML-based syntax (XQueryX) has also been specified**

**A possible result**

```xml
<?xml version="1.0" encoding="UTF-8"?>
<cheapBooks>
  <Title>Cheap Books</Title>
  <book price="25.50">
    <author>David Mayer</author>
    <publisher>Benjamin Cummings</publisher>
    <year>1998</year>
  </book>
  <book price="40.00">
    <title>Designing Internet applications</title>
    <publisher>Prentice Hall</publisher>
    <year>1998</year>
  </book>
</cheapBooks>
```

**XQuery and XPath**

- **XQuery is an extension of XPath (2.0)**
  - Common data model, functions and operators
  - > review some XPath first
- **XPath used in several other contexts, too:**
  - For uniqueness constraints in XML Schema
  - For in validation rules of Schematron
  - For pattern matching and selection in XSLT
  - For addressing in XLink and XPointer

**XPath in a Nutshell**

- **XPath 1.0 (W3C Rec. 11/99)**
  - a compact non-XQuery syntax for addressing parts of XML documents (as node-sets in XPath 1.0)
  - also operations on strings, numbers and truth values
- **XPath 2.0 (2.0 Rec. 1/07) extends and generalizes:**
  - data manipulated as sequences of items
    - item = a node or an atomic value of a simple XML Schema datatype
### XQuery/XPath 2.0 Sequences

- Expressions operate on, and return sequences of:
  - atomic values (of simple XML Schema types) and
  - nodes
  - an item = a singleton sequence
  - sequences are flat: no sequences as items
    - \( (1, 2, 3), (1, 1) = (1, 2, 3, 1) \)
  - sequences are ordered, and can contain duplicates
- Unlimited combination of expressions, often with automatic type conversions (e.g. for arithmetics)

### Sequence Expressions

- Constant sequences constructed by listing values
  - comma (\( , \)) is a catenation operator
  - \( (1, 2, 3), (1, 1) = (1, 2, 3, 1) \)
- Shorthands for numeric sequences:
  - \( 1 \text{ to } 4 \) \( \rightarrow (1, 2, 3, 4) \)
  - \( 4 \text{ to } 1 \) \( \rightarrow () \)
  - \( \text{fn:reverse}(1 \text{ to } 4) \) \( \rightarrow (4, 3, 2, 1) \)

### Accessing Documents

- XQuery operates on nodes accessible by input functions
  - \( \text{fn:doc("URI")} \)
    - document-node of the XML document available at URI
  - \( \text{fn:same as document("URI")} \) in XSLT 1.0
  - \( \text{fn:collection("URI")} \)
    - sequence of nodes from URI
  - predefined prefix for the default function namespace: \( \text{fn=\{http://www.w3.org/2005/04/xpath-functions\}} \)

### XQuery/XPath 2.0 Data Model

- Documents are viewed as trees made of six types of nodes:
  - document (additional root above document element)
  - element nodes
  - attribute nodes
  - text nodes
  - Comments and processing instructions
- Obs 1: No entity nodes
- Obs 2: Namespace nodes have been deprecated (XPath/XSLT 1.0 contains them)

### Document trees

- Defined in Sect. 5 of XPath 1.0 spec
  - for XSLT/XPath 2.0 & XQuery in their joint Data Model
- Element nodes have elements, text nodes, comments and processing instructions of their (direct) content as their children
  - NB: attribute nodes are not children (but have a parent)
  - \( \rightarrow \) they have no siblings either
  - the string value of an document/element is the concatenation of its all text-node descendants

### Document Order

- Document order of nodes:
  - \( = \) the depth-first traversal order
  - Document root first
  - Other nodes in the order of the first character of their XML markup in the document text
  - \( > \) an element precedes it's attribute nodes, which precede any content nodes of the element
  - Implementation dependent btw nodes belonging to different trees

### Location Paths

- XPath can select any parts of a document tree using ...
- Location paths
  - evaluated with respect to a context item (\( . \))
    - \( \rightarrow \) in XQuery typically starting from \( $x \) or \( \text{doc}(...) \)
  - Result: a sequence of nodes, in document order, without duplicates

### Path Expressions

- Similar to XPath 1.0: \( /[E|Expr]/…/Expr \)
  - but steps more liberal:
  - arbitrary (node-sequence valued) expressions OK
  - \( 6 \) of \( 13 \) XPath axes: child, descendant, attribute, self, descendant-or-self, parent
  - \( \) others (except namespace) available if Full Axis Feature supported
  - with document-order operators (\( << , >> \)) sufficient for expressing queries (\( \rightarrow \) Exercises)
Location paths

- Consist of location steps separated by '/'
  - each step produces a sequence of items
  - steps evaluated left-to-right, each item in turn as the context item
- Complete location step:
  - AxisName::NodeTest((PredicateExpr))
  - axis specifies the tree relationship between the context node and the selected nodes
  - node test restricts the type and and name of nodes
  - filtered further by 0 or more predicates

Notes on Location Paths (1)

- XPath 2.0 allows unrestricted expressions as steps
  - but intermediate steps must produce nodes only
- Numeric predicates support array-style access:
  - $row[$i]
- Predicates evaluated at a time. This often causes confusion with shorthand notations:
  - $doc("foo.xml")/title[3]
    - third title of each parent (likely none!). Why?
  - $doc("doc.xml")/child():title[3]
    - To get the third title in the doc use
  - $doc("doc.xml")/title[3]

Notes on Location Paths (2)

- References to attributes and subelements easy to use as predicates
  - Get divisions that are of class c or have a head:
    - $doc("doc.xml")//div[@class="C" or head]
  - Values are coerced to Booleans on demand
    - string/sequence true if non-empty
    - number true if and only if not zero
    - (but a single number as a predicate tests for equality with position())

Filter Expressions

- Location steps can be filtered by predicates:
  - $s1 except $s2 = $s1 - $s2
  - Then:
    - $s1 union $s2 = $s1 + $s2
    - $s1 intersect $s2 = $s1 ∩ $s2
    - $s1 except $s2 = $s1 - $s2

Set Operations on Node (!) Sequences

- Assume variable bindings:
  - $s1
  - $s2
  - Then:
    - $s1 union $s2 = $s1 + $s2
    - $s1 intersect $s2 = $s1 ∩ $s2
    - $s1 except $s2 = $s1 - $s2

Node Comparisons

- To compare single nodes,
  - for identity:
    - $book/chap[@id="ch1"] is ($book/chap) [1]
  - true iff the chapter with id="ch1" is indeed the first
  - for document order: < and >
    - $book/chap[@id="ch2"] >>
    - true iff the chapter with id="ch2" appears after
      <title>Intro</title>

Comparing values of sequences and items

- General comparisons btw sequences:
  - =, !=, <, <=, >, >=
  - existential semantics: true iff some pair of values from operand sequences satisfy the condition
    - (1,2) = (2,3); (2,3) = (3,4); (1,2) != (3,4)
  - Same as in XPath 1.0:
    - books[author = "Aho"]
    - books where some author is Aho
- Value comparisons btw single values:
  - $e1 eq $e2; $e1 lt $e2; $e1 gt $e2
  - $e1 le $e2; $e1 ge $e2

XQuery over XPath

- A query is an expression (*lausenke*)
- XQuery adds to XPath expressions
  - Element constructors (= XSLT templates)
  - FLWOR expressions
    - "flower", for-let-where-order-by-return

Central XQuery Expressions

- Path expressions
- Sequence expressions
- Comparison operators
- Quantified expressions
  - (some/very $var in ... satisfies ...)
- Element constructors (= XSLT templates)
- FLWOR expressions
  - "flower", for-let-where-order-by-return
  - XPath 2.0 has a simpler for-return expression, too
- some others in examples ...

Element Constructors

- Similar to XSLT templates:
  - start and end tag enclosing the content
  - literal fragments written directly, expressions enclosed in braces \{ and \}
  - = XSLT 1.0 attribute value templates
- often used inside another expression that binds variables used in the element constructor
  - (There is no ‘current node’ in XQuery)
  - See next

XQuery XQuery adds to XPath expressions

- Returned by respective expressions
- Each of the variables
- Can be though of as loops iterating over the items

Example

- An *emp* element with an *empid* attribute and child elements *name* and *job*, from values in variables $i$, $n$, and $j$:

  \[
  \begin{aligned}
  & <\text{name}> \{ \text{$n$} \} \</\text{name}> \\
  & <\text{job}> \{ \text{$j$} \} \</\text{job}>
  \end{aligned}
  \]

  Also computed constructors:

  \[
  \begin{aligned}
  \text{element} & \{ \text{"emp"} \} \\
  \text{attribute} & \{ \text{"empid"} \} \{ \text{$i$} \}, \text{\"name\"} \{ \text{$n$} \} \text{\"name\"}, \text{\"job\"} \{ \text{$j$} \} \text{\"job\"}
  \end{aligned}
  \]

Example: for clause

- for $i \in (1,2)$, $j \in (1 \text{ to } i)$
  - return \{ \text{tuple} \}

  \[
  \begin{aligned}
  & <\text{tuple}> \{ \text{\"i\"} <\text{i}> \} \</\text{tuple}>
  \end{aligned}
  \]

  Result:

  \[
  \begin{aligned}
  & <\text{tuple}> <\text{tuple}> <\text{tuple}> <\text{tuple}>
  \end{aligned}
  \]

Flow of data in a FLWOR expression

- Constructed from for, let, where, order by and return clauses (~SQL select-from-where)
- Syntax:
  \[
  \begin{aligned}
  & (\text{ForClause} \ |
  \end{aligned}
  \]

  \[
  \begin{aligned}
  & \text{LetClause})+ \\
  \end{aligned}
  \]

  \[
  \begin{aligned}
  & \text{WhereClause?} \\
  \end{aligned}
  \]

  \[
  \begin{aligned}
  & \text{OrderByClause?} \\
  \end{aligned}
  \]

  \[
  \begin{aligned}
  & \text{\"return\" Expr} \\
  \end{aligned}
  \]

  \[
  \begin{aligned}
  & = \text{XPath 2.0 has a simpler for-return expression} \\
  \end{aligned}
  \]

  \[
  \begin{aligned}
  & \text{an ordered sequence of items}
  \end{aligned}
  \]

for clauses

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

Flow of data in a FLWOR expression

- Constructed from for, let, where, order by and return clauses (~SQL select-from-where)
- Syntax:
  \[
  \begin{aligned}
  & (\text{ForClause} \ |
  \end{aligned}
  \]

  \[
  \begin{aligned}
  & \text{LetClause})+ \\
  \end{aligned}
  \]

  \[
  \begin{aligned}
  & \text{WhereClause?} \\
  \end{aligned}
  \]

  \[
  \begin{aligned}
  & \text{OrderByClause?} \\
  \end{aligned}
  \]

  \[
  \begin{aligned}
  & \text{\"return\" Expr} \\
  \end{aligned}
  \]

  \[
  \begin{aligned}
  & = \text{XPath 2.0 has a simpler for-return expression} \\
  \end{aligned}
  \]

  \[
  \begin{aligned}
  & \text{an ordered sequence of items}
  \end{aligned}
  \]

for clauses

- for $i \in (1,2)$, $j \in (1 \text{ to } i)$
  - return \{ \text{tuple} \}

  \[
  \begin{aligned}
  & <\text{tuple}> <\text{tuple}> <\text{tuple}> <\text{tuple}>
  \end{aligned}
  \]

  Result:

  \[
  \begin{aligned}
  & <\text{tuple}> <\text{tuple}> <\text{tuple}> <\text{tuple}>
  \end{aligned}
  \]

Flow of data in a FLWOR expression

- Constructed from for, let, where, order by and return clauses (~SQL select-from-where)
- Syntax:
  \[
  \begin{aligned}
  & (\text{ForClause} \ |
  \end{aligned}
  \]

  \[
  \begin{aligned}
  & \text{LetClause})+ \\
  \end{aligned}
  \]

  \[
  \begin{aligned}
  & \text{WhereClause?} \\
  \end{aligned}
  \]

  \[
  \begin{aligned}
  & \text{OrderByClause?} \\
  \end{aligned}
  \]

  \[
  \begin{aligned}
  & \text{\"return\" Expr} \\
  \end{aligned}
  \]

  \[
  \begin{aligned}
  & = \text{XPath 2.0 has a simpler for-return expression} \\
  \end{aligned}
  \]

  \[
  \begin{aligned}
  & \text{an ordered sequence of items}
  \end{aligned}
  \]

for clauses

- for $i \in (1,2)$, $j \in (1 \text{ to } i)$
  - return \{ \text{tuple} \}

  \[
  \begin{aligned}
  & <\text{tuple}> <\text{tuple}> <\text{tuple}> <\text{tuple}>
  \end{aligned}
  \]

  Result:

  \[
  \begin{aligned}
  & <\text{tuple}> <\text{tuple}> <\text{tuple}> <\text{tuple}>
  \end{aligned}
  \]
**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 doc("bib.xml")//book
    -> many bindings (to single books)
  - let $b1 := doc("bib.xml")//book
    -> a single binding (to sequence of books)

**Example: let clauses**

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

Result:

```
<out>
<s>1</s> / <s>2</s> / <s>3</s>
</out>
```

**for/let clauses**

- A FLWOR expr may contain several fors and lets
  - 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 sequences returned by the for expressions

**where clause**

- binding tuples generated by for and 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 as Booleans (similarly to node-sets in XPath 1.0): empty ~ false, non-empty ~ true

**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 sub-expressions

**Example: for + return**

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

Result:

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

**Positional variables: 'at'**

- For items, can also get their position in the seq:
  ```xml
  for $char$ at $i$ in ("a", "b", "c")
  return concat($i, ".", $char, ";")
  ```
  -> 1.a;2.b;3.c;

- Could pair items by their position:
  ```xml
  let $boys$ := doc("kids.xml")//boy,
  $girls$ := doc("kids.xml")//girl
  for $b$ at $i$ in $boys$
  where $i$ le count($girls$)
  return <pair>{$b}, $girls[$i] </pair>
  ```
Examples (modified from "XML Query Use Cases")

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

  ```xml
  <books>
    <title>Advanced Programming in the Unix environment</title>
    <book year="1999">...<author_list>{ (: group books by authors :) }...</book>
    <book year="2000">...<author_list>{ (: group books by authors :) }...</book>
  </books>
  ```

  Result could be...

  ```xml
  <!--<recent-MK-books>...<recent-MK-books>-->"
  `<title>TCP/IP Illustrated</title>`
  `<book year="1999">...<author_list>{ (: group books by authors :) }...</book>`
  `<book year="2000">...<author_list>{ (: group books by authors :) }...</book>`
  `/recent-MK-books`
  ```

Invert the book-list structure

```xml
<author_list>{ (: group books by authors :) }
  for $a in distinct-values(
    doc("bib.xml")//author )
  return $a
</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
  - procedure elements containing
    \textit{incision} sub-elements
- Return a "critical sequence" of contents between the first and the second incisions of the first procedure

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

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

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

Computing a "critical sequence"

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

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:

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

```xml
declare function local:precedes($a as node(), $b as node()) as xs:boolean
{
  if $a is no ancestor of $b then
    empty($a//node() intersect $b
    )
  else
    $a << $b
}
```

User-defined functions: Example

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

Recursive Transformations

Example: "Table-of-contents" for nested sections
- NB if-then-else (in ordinary XPath 2.0 expressions, too)
```xml
declare namespace my=http://my.own-ns.org;
declare function my:toc($n as element()) as element()*
{
  if (name($n)="sect") then 
    <sect>
      for $c in $n/* return my:toc($c) </sect>
  else if (name($n)="title") then $n
  else 
    for $c in $n/* return my:toc($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, descrip): part numbers and descriptions
  - S (sno, sname): supplier numbers and names
  - SP (sno, pno, price): who supplies which parts and for what price?

Possible XML representation of relations

<table>
<thead>
<tr>
<th>Relational data</th>
<th>XML representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Sno, Sname</td>
</tr>
<tr>
<td>P</td>
<td>Pno, Descrip</td>
</tr>
<tr>
<td>SP</td>
<td>Sno, Pno, Price</td>
</tr>
</tbody>
</table>

Selecting in SQL vs. XQuery

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

Grouping

- Many queries involve grouping data and applying aggregation function like count or avg to each group
- in SQL: GROUP BY and HAVING clauses
- Example: Find the part number and average price for parts with at least 3 suppliers
```sql
SELECT pno, avg(price) AS avgprice FROM sp 
GROUP BY pno 
HAVING count(*) >= 3 
ORDER BY pno;
```
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>

Joins

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

for $sp in doc("sp.xml")//sp_tuple,
$pn in doc("p.xml")//p_tuple[pno = $sp/pno],
$s in doc("s.xml")//s_tuple[eno = $sp/eno]
order by $sp/descrip, $s/sname
return <sp_pair>{
  $s/sname ,
  $sp/pno, $s/descrip
}</sp_pair>

XQuery vs. XSLT 1.0

- Could we express XQuery queries with XSLT?
  - In principle yes, always. (but could be tedious)
- 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>

  - 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 select="x" />}

XQuery vs. XSLT 1.0: Example

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

XSLT for XQuery FLWOR Expressions

- NB: The sketched simulation is not complete:
  - Only two things, roughly, can be done with XSLT 1.0 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 $y in <a><key>{$y/code}</key></a>, ...
      order by $y/key

XQuery: Summary

- A recent W3C XML query language, also capable of general XML processing
- Vendor support??
  - http://www.w3.org/XML/Query
  - mentions ~ 50 prototypes or products (2004: ~ 30, 2005: ~ 40;
    free, commercial, ... Oracle, IBM DB2, MS SQL, Sercer)
- Future??
  - Interesting confluence of document and database research
  - highly potential for XML-based data integration