7 Querying XML

- How to access XML data sources?
- **XQuery**: XML Query Lang, W3C Rec, 01/07
  - joint work by XML Query and XSL WGs
  - operate on XPath 2.0 and XSLT 2.0
- influenced by many research groups and query languages
  - Quilt, XPath, XQL, XML-QL, SQL, OQL, Lorel, ...
- A query language for any XML-represented data: both documents and databases

**Examples**

```xml
<xquery version="1.0">
  <cheatBooks>
    <Title>Cheap Books</Title>
    <book price="26.50">
      <title>Designing Internet applications</title>
      <author>Michael Levenstein</author>
      <publisher>Prentice Hall</publisher>
      <year>1999</year>
    </book>
    <book price="40.00">
      <title>Designing Internet applications</title>
      <author>Michael Levenstein</author>
      <publisher>Prentice Hall</publisher>
      <year>1999</year>
    </book>
  </cheatBooks>
</xquery>
```

A possible result

```xml
<?xml version="1.0" encoding="UTF-8"?>
<cheatBooks>
  <Title>Cheap Books</Title>
  <book price="26.50">
    <title>Designing Internet applications</title>
    <author>Michael Levenstein</author>
    <publisher>Prentice Hall</publisher>
    <year>1999</year>
  </book>
  <book price="40.00">
    <title>Designing Internet applications</title>
    <author>Michael Levenstein</author>
    <publisher>Prentice Hall</publisher>
    <year>1999</year>
  </book>
</cheatBooks>
```

XQuery: Basics

- A query is an expression (lauseke)
- Expressions operate on, and return sequences of
  - atomic values (of XML Schema simple types) and
  - nodes (of 6 types = XPath node types = NS nodes)
  - an item = a singleton sequence
  - sequences are flat: no sequences as items
  - (1, 2, 3), (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)

XQuery: Accessing Documents

- XQuery operates on nodes accessible by input functions
  - `fn:doc("URI")`
  - `fn:element-of-node("URI")`
  - `fn:attribute-of-node("URI")`
  - `fn:collection("URI")`
  - `fn:sequence-of-nodes-from-uri("URI")`
  - predeclared prefix for the default function namespace:
    `fn=http://www.w3.org/2005/04 xpath-functions`

SDPL 2007 7: Querying XML with XQuery 1
Circle XQuery Expressions

- Path expressions
- Sequence expressions
- Comparison operators
- Quantified expressions
  (some/every $x... in ... satisfies ...)
- Element constructors (= XSLT templates)
- FLWOR expressions
  ("flower", for-let-where-order-by-return)
- and others, in examples ...

Path Expressions

- Similar to XPath 1.0: [][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 (→ Exercises)
  - produce a sequence of nodes in document order, without duplicates

Sequence Expressions

- Constant sequences constructed by listing values
  - comma (,) is a catenation operator
  - $(1, (2, 3), (1), (1), 1) = (1, 2, 3, 1)$
  - Also dynamically generated:
    - $(<start />, $emp/child, <end/>)$
  - Shorthands for numeric sequences:
    - 1 to 4  →  $(1, 2, 3, 4)$
    - 4 to 1  →  $(1)$
    - fn:reverse(1 to 4) →  $(4, 3, 2, 1)$

Set Operations on Node (!) Sequences

- Assume variable bindings:
  - $s1 = \ldots \ldots \ldots \ldots$ $s2$
  - Then:
    - $s1 \text{ union } s2 = \ldots \ldots \ldots \ldots$
    - $s1 \text{ intersect } s2 = \ldots \ldots \ldots \ldots$
    - $s1 \text{ except } s2 = \ldots \ldots \ldots \ldots$

Node Comparisons

- To compare single nodes,
  - for identity:
    - $\langle \text{book}[\text{id}=\text{ch1}] \rangle$ is $\langle \text{book}[\text{id}=\text{ch1}] \rangle$
    - true iff the chapter with id=ch1 is indeed the first
  - for document order: $<<$ and $>>$
    - $\langle \text{book}[\text{id}=\text{ch2}] \rangle$ $>>$
    - $\langle \text{book}[\text{title}=\text{"Intro"}] \rangle$
    - true iff the chapter with id=ch2 appears after
  - $\langle \text{title}=\text{Intro} \rangle$

Comparing values of sequences and items

- General comparisons btw sequences:
  - $=,$ $\neq,$ $<,$ $>_,$ $\geq,$ $\leq,$ $>,$ $\geq$
  - existential semantics: true iff some pair of values from operand sequences satisfy the condition
  - $(1,2) = (2,3); (2,3) = (3,4); (1,2) \neq (3,4)$
  - Same as in XPath 1.0:
    - $\langle /\text{book[author=\"Aho\"]} \rangle$
    - $\langle \text{books where some author is Aho} \rangle$
- Value comparisons btw single values:
  - $\text{eq, ne, lt, le, gt, ge}$
  - $1 \text{ eq } 3 \neq 2; 10 \text{ lt } 20; \langle \text{books[price le 100]}\rangle$

Filter Expressions

- Location steps can be filtered by predicates:
  - $\langle \text{book[\text{chap}[\text{id}=\text{ch2}]]} \rangle$ $<<$
  - the title of the last chapter of appendix, whichever is last
- Other sequences, too:
  - $(1 \text{ to } 20); (1 \text{ mod } 5 \text{ eq } 0) \rightarrow (5, 10, 15, 20)$
  - $(1)$ generalize from XPath 1.0 shorthand for self::node() into the context item

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
Example

An `emp` element with an `empid` attribute and child elements `name` and `job`, from values in variables `sid`, `sn`, and `sj`:

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

Also computed constructors:

- `element ("emp") {` attribute ("empid") {$id}.
- `name ("{$n}")` `<name>`.
- `<job> {"{$j}" } </job>`

---

Flow of data in a FLWOR expression

```
FOR/LET Clauses:

\[\text{Ordered list of tuples of bound variables}\]

WHERE Clause:

\[\text{Pruned list of tuples of bound variables}\]

RETURN Clause:

\[\text{Instance of XML Query data model}\]
```

---

Example: `for` clause

```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>`

---

Example: let clauses

```xml
let $a := \langle{\text{one}/}, <\text{two}/>, <\text{three}/}\rangle
return <out> {$a} </out>
```

Result:

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

---

FLWOR ("flower") Expressions

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

---

for clauses

- `for $v_i$ in Exp_i,` $\vdots$
  - 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 though of as loops iterating over the items
  returned by respective expressions

---

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 $i$ in doc("bib.xml")//book`
    - many bindings to single books
  - `let $ib := doc("bib.xml")//book`
    - a single binding to sequence of books

---

for/let clauses

- A FLWOR expr may contain several `for`s
  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

---

**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} </book>
  </book>
</recent-MK-books>
```

---

**Publishers with avg price of their books:**

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

Queries on Document Order

- Operators << and >>:
  - $x << y$ is true if 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

```xml
for $p$ in distinct-values(doc("report.xml")//procedure)[]
order by $p$
return $<publisher>
  <name>{$p//name}</name>,
  <procedure>{$p//procedure}</procedure>
  <incision>{$p//incision}</incision>: supplier numbers and names
</publisher>
```

User-defined functions: Example

```xml
declare function local:precedes($a as node(),
$b as node()) as xs:boolean
(a << $b and (: $a is no ancestor of $b :) empty($a//node()) intersect $b ));
```

```xml
<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>
```

Recursive Transformations

- Example: "Table-of-contents" for nested sections
  - NB if-then-else in (ordinary XPath 2.0 expressions, too)
  - 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?

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}</name>,
  <book>{$p//book}
    <title>{$p/title},
    <price>{$p/price} </book>
</publisher>
```

User-defined functions: Example

```xml
Now, "critical sequence" without ancestors of incision:
```

```xml
for $p$ :=
(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>
```

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?
In principle:

Many queries involve grouping data and applying aggregation function like `count` or `avg` to each group.

In SQL, GROUP BY and HAVING clauses are used:

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

For XQuery, using the `group` operator:

```
for $p in doc("p.xml")//p_tuple
where starts-with($p/descriv, "Gear")
order by $p/pno
return $p/pno
```

XQuery: for $p in distinct-values(doc("p.xml")//p_tuple) where count($p) >= 3
order by $p
return <well_supplied_item>
   <pno>{$p/pno}</pno>,
   <avgprice>{avg($p/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
where $sp/sno = $sp/pno,
order by $sp/descrip, $sp/sname
return <sp_pair>
   $sp/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)
- 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: let $y := Expr ...
   corresponds directly to:
   <xsl:if test="Condition">
      ... translation of the rest
   </xsl:if>
XQuery vs. XSLT 1.0

XQuery: return ElemConstructor

can be simulated with a corresponding XSLT template:
  • static fragments as such
  • enclosed expressions in element content, e.g.
    <xsl:copy-of select="$s/sname"/>

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 $y in (<byte>{$y/code}</key></a>, ...)
    order by $y/key

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)">($b/title)</book>

⇒

<xsl:template match="/">
  <xsl:if test="($b/publ = 'MK' and $b/year > 1998)">
    <book year="($b/year)">
      <xsl:copy-of select="($b/title)" />
    </book>
  </xsl:if>
</xsl:template>

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)
– Future?? Interesting confluence of document and database research, and highly potential for
  XML-based data integration