XPath & XQuery
(continued)

CS 645
Apr 24, 2008
Today’s lecture

• Review of XPath
• continuation of XQuery
Querying XML Data

• XPath = simple navigation through the tree

• XQuery = the SQL of XML
Sample Data for Queries

<bib>
  <book>
    <publisher> Addison-Wesley </publisher>
    <author> Serge Abiteboul </author>
    <author> Rick </author>
    <author> Hull </author>
    <author> Victor Vianu </author>
    <title> Foundations of Databases </title>
    <year> 1995 </year>
  </book>
  <book price="55">
    <publisher> Freeman </publisher>
    <author> Jeffrey D. Ullman </author>
    <title> Principles of Database and Knowledge Base Systems </title>
    <year> 1998 </year>
  </book>
</bib>
Xpath: Summary

bib matches a bib element
  * matches any element
  / matches the root element
/bib matches a bib element under root
bib/paper matches a paper in bib
bib///paper matches a paper in bib, at any depth
//paper matches a paper at any depth
paper I book matches a paper or a book
@price matches a price attribute
bib/book/@price matches price attribute in book, in bib
bib/book/[@price<“55”]/author/lastname matches…
Context Nodes and Relative Paths

XPath has a notion of a context node: it’s analogous to a current directory
- “.” represents this context node
- “..” represents the parent node
- We can express relative paths:
  subpath/sub-subpath/../../ gets us back to the context node

- By default, the document root is the context node
Predicates – Selection Operations

A predicate allows us to filter the node set based on selection-like conditions over sub-XPaths:

```
/dblp/article[title = “Paper1”]
```

which is equivalent to:

```
/dblp/article[./title/text() = “Paper1”]
```
dot in XPath qualifiers

- //author
- //author[first-name]
- //author[./first-name]  equivalent
- //author[/first-name]  qualifier starts at root
- //author[//first-name]
- //author[.//first-name]
Xpath: More Predicates

/bib/book/author[firstname][address[.//zip][city]]/lastname

Result: <lastname> … </lastname>

    <lastname> … </lastname>
Axes: More Complex Traversals

Thus far, we’ve seen XPath expressions that go down the tree

- But we might want to go up, left, right, etc.
- These are expressed with so-called axes:
  - `self::path-step`
  - `child::path-step`
  - `parent::path-step`
  - `descendant::path-step`
  - `ancestor::path-step`
  - `descendant-or-self::path-step`
  - `ancestor-or-self::path-step`
  - `preceding-sibling::path-step`
  - `following-sibling::path-step`
  - `preceding::path-step`
  - `following::path-step`

- The previous XPaths we saw were in “abbreviated form”
XQuery
Query Language and Data Model

• A query language is “closed” w.r.t. its data model if input and output of a query conform to the model

• SQL
  – Set of tuples in, set of tuples out

• XPath 1.0
  – A tree of nodes (well-formed XML) in, a node set out.

• XQuery 1.0
  – Sequence of items in, sequence of items out

• Compositionality of a language
  – Output of Query 1 can be used as input to Query 2
XQuery

• XQuery extends XPath to a query language that has power similar to SQL.
• XQuery is an expression language.
  - Like relational algebra --- any XQuery expression can be an argument of any other XQuery expression.
  - Unlike RA, with the relation as the sole datatype, XQuery has a subtle type system.
XQuery Values

• Item = node or atomic value.
• Value = ordered sequence of zero or more items.
• Examples:
  • () = empty sequence.
  • ("Hello", "World")
  • ("Hello", <PRICE>2.50</PRICE>, 10)
Sample Data for Queries

<bib>
  <book>
    <publisher> Addison-Wesley </publisher>
    <author> Serge Abiteboul </author>
    <author> Rick </author>
    <last-name> Hull </last-name>
    <author> Victor Vianu </author>
    <title> Foundations of Databases </title>
    <year> 1995 </year>
  </book>
  <book price="55">
    <publisher> Freeman </publisher>
    <author> Jeffrey D. Ullman </author>
    <title> Principles of Database and Knowledge Base Systems </title>
    <year> 1998 </year>
  </book>
</bib>
Document Nodes

- **Form:**
  - `doc("<file name>")`.

- **Establishes a document to which a query applies.**

- **Example:**
  - `doc("/courses/445/bib.xml")`
FOR-WHERE-RETURN

Find all book titles published after 1995:

\[
\text{for } \$x \text{ in doc("bib.xml")/bib/book}
\text{where } \$x/\text{year/text()} > 1995
\text{return } \$x/\text{title}
\]

Result:

\[
<\text{title}> \text{abc} </\text{title}>
<\text{title}> \text{def} </\text{title}>
<\text{title}> \text{ghi} </\text{title}>
\]
FOR-WHERE-RETURN

Equivalently (perhaps more geekish)

```
return $x
```

And even shorter:

```
```
FOR-WHERE-RETURN

• Find all book titles and the year when they were published:

```xml
for $x$ in doc("bib.xml")/bib/book
return <answer>
  <what>{$x/title/text()}</what>
  <when>{$x/year/text()}</when>
</answer>
```

We can construct whatever XML results we want!
Answer

<answer>
  <what> How to cook a Turkey </what>
  <when> 2005 </when>
</answer>

<answer>
  <what> Cooking While Watching TV </what>
  <when> 2006 </when>
</answer>

<answer>
  <what> Turkeys on TV </what>
  <when> 2007 </when>
</answer>

......
FOR-WHERE-RETURN

• Notice the use of “{" and “}”
• What is the result without them?

```xml
for $x$ in doc("bib.xml")/bib/book
return <answer>
    <title>  $x/title/text() </title>
    <year> $x/year/text() </year>
</answer>
```
More Examples of WHERE

• Selections

```xml
for $b in doc("bib.xml")/bib/book
where $b/publisher = "Addison Wesley" and $b/@year = "1998"
return $b/title

for $b in doc("bib.xml")/bib/book
where empty($b/author)
return $b/title

for $b in doc("bib.xml")/bib/book
where count($b/author) = 1
return $b/title
```

Aggregates over a sequence: count, avg, sum, min, max
Aggregates

Find all books with more than 3 authors:

```xml
for $x$ in doc("bib.xml")/bib/book
where count($x/author)>3
return $x
```

count = a function that counts
avg = computes the average
sum = computes the sum
distinct-values = eliminates duplicates
Aggregates

Same thing:

```xml
for $x$ in doc("bib.xml")/bib/book[count(author)>3]
RETURN $x
```
FLWOR expressions

• FLWOR is a high-level construct that
  – supports iteration and binding of variables to intermediate results
  – is useful for joins and restructuring data
• Syntax: For-Let-Where-Order by-Return

  for $x$ in expression1  
  [let $y := expression2 ]  
  [where expression3 ]  
  [order by expression4 (ascending|descending)? ]  

  return expression4
Example FLOWR Expression

for $x$ in doc("bib.xml")/bib/book  // iterate, bind each item to $x$
let $y := $x/author               // no iteration, bind a sequence to $y$
where $x/title="XML""              // filter each tuple ($x, $y)
order by $x/@year descending       // order tuples
return count($y)                  // one result per surviving tuple

• The for clause iterates over all books in an input document, binding $x$ to each book in turn.
• For each binding of $x$, the let clause binds $y$ to all authors of this book.
• The result of for and let clauses is a tuple stream in which each tuple contains a pair of bindings for $x$ and $y$, i.e. ($x$, $y$).
• The where clause filters each tuple ($x$, $y$) by checking predicates.
• The order by clause orders surviving tuples.
• The return clause returns the count of $y$ for each surviving tuple.
FOR v.s. LET

FOR
• Binds *node variables* → iteration

LET
• Binds *collection variables* → one value
FOR v.s. LET

\[
\text{for } $x \text{ in } /\text{bib/book} \\
\text{return } <\text{result}> \{ \ $x \ \} \ </\text{result}>
\]

Returns:
\[
<\text{result}> <\text{book}>...</\text{book}></\text{result}>
<\text{result}> <\text{book}>...</\text{book}></\text{result}>
<\text{result}> <\text{book}>...</\text{book}></\text{result}>
...
\]

\[
\text{let } $x := /\text{bib/book} \\
\text{return } <\text{result}> \{ \ $x \ \} \ </\text{result}>
\]

Returns:
\[
<\text{result}> <\text{book}>...</\text{book}>
<\text{book}>...</\text{book}>
<\text{book}>...</\text{book}>
...

<\text{result}>
\]

28
FOR-WHERE-RETURN

• “Flatten” the authors, i.e. return a list of (author, title) pairs

```xml
for $b$ in doc("bib.xml")/bib/book,
    $x$ in $b/title/text()$,
    $y$ in $b/author$
return <answer>
    <title> { $x$ } </title>
    { $y$ }  
</answer>
```

Answer:
```xml
<answer>
    <title> abc </title>
    <author> efg </author>
</answer>
<answer>
    <title> abc </title>
    <author> hkj </author>
</answer>
```
XQuery: Nesting

For each author of a book by Morgan Kaufmann, list all books he/she published:

```
for $b in doc("bib.xml")/bib,
   $a in $b/book[publisher/text()="Morgan Kaufmann"]/author
return <result>
   { $a,
     for $t in $b/book[author/text()=$a/text()]/title
     return $t
   }
</result>
```

In the RETURN clause comma concatenates XML fragments.
XQuery

Result:

<result>
   <author>Jones</author>
   <title>abc</title>
   <title>def</title>
</result>
<result>
   <author>Smith</author>
   <title>ghi</title>
</result>
Getting Distinct Values from FOR

- Distinct values: the \texttt{fn:distinct-values} function eliminates duplicates in a sequence \textit{by value}
  - The \texttt{for} expression evaluates to a sequence of nodes
- \texttt{fn:distinct-values} converts it to a sequence of atomic values and removes duplicates

\begin{verbatim}
for $a$ in distinct-values(doc("bib.xml")/book/author)
return <author-name> {$a} </author-name>
\end{verbatim}

versus

\begin{verbatim}
for $a$ in doc("bib.xml")/book/author
return $a$
\end{verbatim}
Value Comparison

• Value comparison “eq”: compares single values
• “eq” applies atomization \((\text{fn:}\text{data}(\ ))\) to each operand
  – Given a sequence of nodes, \(\text{fn:}\text{data}(\ )\) returns an atomic value for each node which consists of:
    • a string value, i.e., the concatenation of the string values of all its Text Node descendants in document order
    • a type, e.g., \(\text{xdt:}\text{untypedAtomic}\)
  – For each operand, “eq” uses the \(\text{fn:}\text{data}()\) result if it evaluates to a singleton sequence, o.w. runtime error.

✓ for \$a\ in\ \text{doc(“bib.xml”)}/\text{bib/book/author}\nwhere \$a\ \text{eq} \text{“PeterBuneman”}\nreturn \$a/..

✗ for \$b\ in\ \text{doc(“bib.xml”)}/\text{bib/book}\nwhere \$b/\text{author}\ \text{eq} \text{“PeterBuneman”}\nreturn \$b/\text{author}
General Comparison

• General comparison operators (\(=, \neq, <, >, \leq, \geq\)): 
  existentially quantified comparisons, applied to operand sequences of any length

• Atomization (fn:data()) is applied to each operand to get a sequence of atomic values

• Comparison is true if one value from a sequence satisfies the comparison

```xml
for $b$ in doc("bib.xml")/bib/book
where $b$/author = "PeterBuneman"
return $b$/author
```
String Operations

- Functions for string matching
  \[ \text{fn:contains}(\text{x}s\text{:string}, \text{x}s\text{:string}) \]
  \[ \text{fn:starts}(\text{ends})-\text{with}(\text{x}s\text{:string}, \text{x}s\text{:string}) \]
  \[ \text{fn:substring}-\text{before}(\text{after})(\text{x}s\text{:string}, \text{x}s\text{:string}) \]
  ...

- Again, atomization (fn:data()) is applied to each function parameter to get an atomic value.

```xml
for $a$ in doc("bib.xml")//author
where contains($a, "Ullman")
return $a
```

```xml
<author>
  <first>Jeffery</first>
  <last>Ullman</last>
</author>
```
**Joins in FOR, LET, WHERE**

- **Joins**

```xml
for $b$ in doc("bib.xml")/book,
    $p$ in doc("publishers.xml")/publisher
where $b/publisher = p/name
return ($b/title, $p/name, $p/address)
```

A tuple here is ($b, $p), a unique combination of bindings of $b, $p.

```xml
for $d$ in doc("depts.xml")/depts/deptno
let $e := doc("emps.xml")/emps/emp[deptno = $d]
where count($e) >= 10
order by avg($e/salary) descending
return <big-dept>
    { $d,
        <headcount>{count($e)}</headcount>,
        <avgsal>{avg($e/salary)}</avgsal>  }
</big-dept>
```

A tuple here is ($d, $e)
Element Construction

```xml
<bib>
{ for $b in doc("bib.xml")/bib/book
  where $b/publisher = "Addison-Wesley"
  and $b/@year > 1991
  return <book year="{ $b/@year }">
    { $b/title }
  </book>
}
</bib>
```
Nested FLWOR

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

The nested FLOWR effectively implements “group books by author”. No Group By in XQuery!