Lecture 11: Xpath/XQuery

Wednesday, April 17, 2007
Outline

• XPath
• XQuery

• See recommend readings in previous lecture
Querying XML Data

• XPath = simple navigation through the tree
• XQuery = the SQL of XML
• XSLT = recursive traversal
  – will not discuss in class
Sample Data for Queries

<bib>
  <book>
    <publisher> Addison-Wesley </publisher>
    <author> Serge Abiteboul </author>
    <author> <first-name> Rick </first-name> <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>
Data Model for XPath

The root element

The root

Addison-Wesley

Serge Abiteboul
XPath: Simple Expressions

/bib/book/year

Result:  <year> 1995 </year>
         <year> 1998 </year>

/bib/paper/year

Result:  empty  (there were no papers)

What’s the difference?
XPath: Restricted Kleene Closure

//author

Result: <author> Serge Abiteboul </author>
       <author> <first-name> Rick </first-name> <last-name> Hull </last-name> </author>

<author> Victor Vianu </author>
<author> Jeffrey D. Ullman </author>

/bib//first-name

Rick </first-name>
Xpath: Attribute Nodes

/bib/book/@price

Result: “55”

@price means that price is has to be an attribute
Xpath: Wildcard

```
//author/*
```

Result: `<first-name> Rick </first-name>
<last-name> Hull </last-name>`

* Matches any element
@* Matches any attribute
Xpath: Text Nodes

/bib/book/author/text()  

Result:  
  Serge Abiteboul  
  Victor Vianu  
  Jeffrey D. Ullman

Rick Hull doesn’t appear because he has firstname, lastname

Functions in XPath:

  – text() = matches the text value
  – node() = matches any node (= * or @* or text())
  – name() = returns the name of the current tag
Xpath: Predicates

/bib/book/author[firstname]

Result: <author> <first-name> Rick </first-name>
         <last-name> Hull </last-name>
      </author>
Xpath: More Predicates

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

Result: <lastname> … </lastname>
   <lastname> … </lastname>

How do we read this?
First remove all qualifiers (predicates):
/bib/book/author /lastname

Then add them one by one:
/bib/book/author[firstname][address]/lastname

etc
Xpath: More Predicates

/bib/book[@price < 60]

/bib/book[author/@age < 25]

/bib/book[author/text()]
Xpath: Position Predicates

/bib/book[last()] - The last book
Xpath: More Axes

. means current node

\[ \text{Same as} \quad /\text{bib}/\text{book}[\text{./review}] \]

\[ /\text{bib}/\text{book}[\text{./review}] \quad \text{Same as} \quad /\text{bib}/\text{book}[\text{review}] \]

\[ /\text{bib}/\text{author}/. /\text{firstname} \quad \text{Same as} \quad /\text{bib}/\text{author}/\text{firstname} \]
Xpath: More Axes

.. means parent node

/bib/author/.. /author/zip  

Same as  

/bib/author/zip

/bib/book[.//review/../.comments]

Same as

/bib/book[.//*[/comments][review]]

Hint: don’t use ..
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|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…
XQuery

- Based on Quilt, which is based on XML-QL
- Uses XPath to express more complex queries
FLWR ("Flower") Expressions

```
FOR ...
LET...
WHERE...
RETURN...
```
FOR-WHERE-RETURN

Find all book titles published after 1995:

FOR $x$ IN document("bib.xml")/bib/book
WHERE $x$/year/text() > 1995
RETURN $x$/title

Result:
<title> abc </title>
<title> def </title>
<title> ghi </title>
FOR-WHERE-RETURN

Equivalently (perhaps more geekish)

RETURN $x$

And even shorter:

COERCION

The query:

```
RETURN $x
```

Is rewritten by the system into:

```
RETURN $x
```
FOR-WHERE-RETURN

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

FOR $x$ IN document("bib.xml")/ bib/book
RETURN <answer>
   <title>{ $x/title/text() } </title>
   <year>{ $x/year/text() } </year>
</answer>

Result:
<answer> <title> abc </title> <year> 1995 </year> </answer>
<answer> <title> def </title> <year> 2002 </year> </answer>
<answer> <title> ghk </title> <year> 1980 </year> </answer>
FOR-WHERE-RETURN

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

```xml
FOR $x IN document("bib.xml")/ bib/book
RETURN <answer>
    <title> $x/title/text() </title>
    <year> $x/year/text() </year>
</answer>
```

```
<answer> <title> $x/title/text() </title> <year> $x/year/text() </year> </answer>
<answer> <title> $x/title/text() </title> <year> $x/year/text() </year> </answer>
<answer> <title> $x/title/text() </title> <year> $x/year/text() </year> </answer>
```
Nesting

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

```
FOR $b IN document("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
<result>
    <author>Jones</author>
    <title>abc</title>
    <title>def</title>
</result>

<result>
    <author>Smith</author>
    <title>ghi</title>
</result>
Aggregates

Find all books with more than 3 authors:

```
FOR $x$ IN document("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:

FOR $x$ IN document("bib.xml")/bib/book[count(author)>3]
RETURN $x$
Eliminating Duplicates

Print all authors:

```text
FOR $a IN distinct-values($b/book/author/text())
RETURN <author> { $a } </author>
```

Note: distinct-values applies ONLY to values, NOT elements
The LET Clause

Find books whose price is larger than average:

```
FOR $b in document("bib.xml")/bib
LET $a:=avg($b/book/price/text())
FOR $x in $b/book
WHERE $x/price/text() > $a
RETURN $x
```
Flattening

• Compute a list of (author, title) pairs

```
FOR $b IN document("bib.xml")/bib/book, $x IN $b/title/text(), $y IN $b/author/text() RETURN <answer>
  <title> { $x } </title>
  <author> { $y } </author>
</answer>
```

Input:
```
<book>
  <title> Databases </title>
  <author> Widom </author>
  <author> Ullman </author>
</answer>
```

Output:
```
<answer>
  <title> Databases </title>
  <author> Widom </author>
</answer>
```
Re-grouping

• For each author, return all titles of her/his books

```
FOR $b IN document("bib.xml")/bib,
  $x IN $b/book/author/text()
RETURN
  <answer>
    <author> { $x } </author>
    { FOR $y IN $b/book[author/text()=$x]/title
      RETURN $y }
  </answer>
```

Result:
```
<answer>
  <author> efg </author>
  <title> abc </title>
  <title> klm </title>
  . . .
</answer>
```

What about duplicate authors?
Re-grouping

• Same, but eliminate duplicate authors:

```xml
FOR $b IN document("bib.xml")/bib
LET $a := distinct-values($b/book/author/text())
FOR $x IN $a
RETURN
   <answer>
   <author> $x </author>
   { FOR $y IN $b/book[author/text()=$x]/title
     RETURN $y }
   </answer>
```
Re-grouping

- Same thing:

```plaintext
FOR $b$ IN document("bib.xml")/bib,
  $x$ IN distinct-values($b/book/author/text())
RETURN
  <answer>
    <author> $x$ </author>
    { FOR $y$ IN $b/book[author/text()=$x]/title
      RETURN $y$ } }
  </answer>
```
SQL and XQuery Side-by-side

Product(pid, name, maker, price)  
Find all product names, prices, sort by price

SQL

SELECT x.name, x.price
FROM Product x
ORDER BY x.price

XQuery

FOR $x in document("db.xml")/db/Product/row
ORDER BY $x/price/text()
RETURN <answer>
    { $x/name, $x/price }
</answer>
Xquery’s Answer

<answer>
    <name> abc </name>
    <price> 7 </price>
</answer>

<answer>
    <name> def </name>
    <price> 23 </price>
</answer>

Notice: this is NOT a well-formed document!
(WHY ???)
Producing a Well-Formed Answer

<myQuery>
{ FOR $x in document("db.xml")/db/Product/row
  ORDER BY $x/price/text()
  RETURN <answer>
    { $x/name, $x/price }
  </answer>
}
</myQuery>
Xquery’s Answer

<myQuery>
  <answer>
    <name> abc </name>
    <price> 7 </price>
  </answer>
  <answer>
    <name> def </name>
    <price> 23 </price>
  </answer>
  ....
</myQuery>

Now it is well-formed!
SQL and XQuery Side-by-side

Product(pid, name, maker, price)  
Company(cid, name, city, revenues)  

Find all products made in Seattle

SQL

```
SELECT x.name
FROM Product x, Company y
WHERE x.maker=y.cid
    and y.city="Seattle"
```

XQuery

```
FOR $r in document("db.xml")/db,
    $x in $r/Product/row,
    $y in $r/Company/row
WHERE
    $x/maker/text()=$y/cid/text()
    and $y/city/text() = “Seattle”
RETURN { $x/name }
```

```
FOR $y in /db/Company/row[city/text()=“Seattle”],
    $x in /db/Product/row[maker/text()=$y/cid/text()]
RETURN { $x/name }
```
<product>
    <row> <pid> 123 </pid> 
        <name> abc </name> 
        <maker> efg </maker> 
    </row>
    <row> ..... </row>
    ... 
</product>
<product>
    ... 
</product>
....
SQL and XQuery Side-by-side

For each company with revenues < 1M count the products over $100

```sql
SELECT y.name, count(*)
FROM Product x, Company y
WHERE x.price > 100 and x.maker=y.cid and y.revenue < 1000000
GROUP BY y.cid, y.name
```

```xquery
FOR $r in document("db.xml")/db,
    $y in $r/Company/row[revenue/text()<1000000]
RETURN
  <proudCompany>
    <companyName> { $y/name/text() } </companyName>
    <numberOfExpensiveProducts>
      { count($r/Product/row[maker/text()=$y/cid/text()][price/text()>100]) } 
    </numberOfExpensiveProducts>
  </proudCompany>
```
SQL and XQuery Side-by-side

Find companies with at least 30 products, and their average price

SELECT y.name, avg(x.price) FROM Product x, Company y WHERE x.maker=y.cid GROUP BY y.cid, y.name HAVING count(*) > 30

FOR $r in document("db.xml")/db, $y in $r/Company/row LET $p := $r/Product/row[maker/text()=$y/cid/text()] WHERE count($p) > 30 RETURN
</theCompany>

A collection

An element