Introduction to Database Systems
CSE 444

Lecture 11
Xpath/XQuery

October 19, 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

Data Model for XPath

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

Result:

```xml
<author> Serge Abiteboul </author>
<author> <first-name> Rick </first-name> <last-name> Hull </last-name>
<author> Victor Vianu </author>
<author> Jeffrey D. Ullman </author>
```

```
/bib/first-name
```

Result: 

```xml
<first-name> Rick </first-name>
```

Xpath: Attribute Nodes

```
/bib/book/@price
```

Result: “55”

@price means that price has to be an attribute

Xpath: Wildcard

```
//author
```

Result: 

```xml
<first-name> Rick </first-name>
<last-name> Hull </last-name>
```

* Matches any element

@* Matches any attribute

Xpath: Text Nodes

```
/bib/book/author/text() 
```

Result: 

```xml
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: 

```xml
<author> <first-name> Rick </first-name> <last-name> Hull </last-name>
```

Xpath: More Predicates

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

Result: 

```xml
<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[2]
The 2nd book
/bib/book[last()]
The last book
/bib/book[@year = 1998][2]
The 2nd of all books in 1998
/bib/book[2][@year = 1998]
2nd book IF it is in 1998

Xpath: More Axes

./means current node
/bib/book[./review]
Same as
/bib/book[review]
/bib/book[./review]
Same as
/bib/book[review]
/bib/author/../firstname
Same as
/bib/author/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)

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

And even shorter:


COERCION

The query:

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

Is rewritten by the system into:


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> ghi </title> <year> 1980 </year> </answer>

FOR-WHERE-RETURN

• Notice the use of "{ " and "}"

• What is the result without them?

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

<answer> <title> abc </title> <year> 1995 </year> </answer>
<answer> <title> def </title> <year> 2002 </year> </answer>
<answer> <title> ghi </title> <year> 1980 </year> </answer>
Nesting
For each author of a book by Morgan Kaufmann, list all books she published:

```xml
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.

Aggregates
Find all books with more than 3 authors:

```xml
FOR $x IN document("bib.xml")/bib/book
WHERE count($x/author)>3
RETURN $x
```

Same thing:

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

Eliminating Duplicates
Print all authors:

```xml
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:

```xml
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

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

Output:
<answer>
<title> Databases </title>
<author> Widom </author>
</answer>
<answer>
<title> Databases </title>
<author> Ullman </author>
</answer>

Re-grouping

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

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

Output:
<answer>
<author> efg </author>
<title> abc </title>
<title> klm </title>
</answer>

Re-grouping

• Same, but eliminate duplicate authors:

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

Output:
<answer>
<author> def </author>
<title> abc </title>
<title> klm </title>
</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 $x/name, $x/price

Xquery’s 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>
```

SQL and XQuery Side-by-side

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

```
SELE
```

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

```
SELECT y.name, count(*)
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[revenue/text()<1000000]
RETURN <theCompany>
  <companyName> { $y/name/text() } </companyName>
  <avgPrice> avg($p/price/text()) </avgPrice>
</theCompany>
```

A collection

An element

A collection

SQL and XQuery Side-by-side