XQuery

CPS 216
Advanced Database Systems

Announcements (March 2)
- Sample midterm and sample solution for Homework #2 available outside my office
  - Remember to check the bulletin board
- Midterm this Thursday in class
  - Everything before XML
  - Open book, open notes
- Project milestone 1 due this Friday
  - See project description for what and how to submit

XQuery

- XPath + full-fledged SQL-like query language
- XQuery expressions can be
  - XPath expressions
  - FLWR (�性) expressions
  - Quantified expressions
  - Aggregation, sorting, and more...
- An XQuery expression returns a result XML document
  - Compare with an XPath expression, which returns a node-set or an atomic value (boolean, number, string)

A simple XQuery based on XPath
Find all books with price lower than $50

```xml
<result>
{ document("bib.xml")/bibliography/book[@price<50] }
</result>
```

- Things outside `{}`’s are copied to output verbatim
- Things inside `{}`’s are evaluated and replaced by the results
  - document("bib.xml") specifies the document to query
  - The XPath expression returns a set of book elements
  - These elements (including all their descendents) are copied to output

FLWR expressions

- Retrieve the titles of books published before 2000, together with their publisher

```xml
<result>
  for $b in document("bib.xml")/bibliography/book
  let $p := $b/publisher
  where $b/year < 2000
  return
  <book>
    { $b/title }
    { $p }
  </book>
</result>
```

- for: loop
  - $b ranges over the result node-set, getting one node at a time
- let: assignment
  - $p gets the entire result of $b/publisher (possibly many nodes)
- where: filter condition
- return: result structuring
  - Invoked in the “innermost loop,” i.e., once for each successful binding of all query variables

An equivalent formulation

- Retrieve the titles of books published before 2000, together with their publisher

```xml
<result>
  return
  <book>
    { $b/title }
    { $b/publisher }
  </book>
</result>
```
Another formulation

- Retrieve the titles of books published before 2000, together with their publisher

```
<result>
  for $b in document('bib.xml')/bibliography/book,
  $p in $b/publisher
  where $b/year < 2000
  return
  <book>
    { $b/title }
    { $p }
  </book>
</result>
```

- Is this query equivalent to the previous two?
  - Yes, if there is one publisher per book
  - No, in general

- No, in general
  - Two result book elements will be created for a book with two publishers
  - No result book element will be created for a book with no publishers

Yet another formulation

- Retrieve the titles of books published before 2000, together with their publisher

```
<result>
  let $b := document('bib.xml')/bibliography/book
  where $b/year < 2000
  return
  <book>
    { $b/title }
    { $b/publisher }
  </book>
</result>
```

- Is this query correct?
  - No!
  - It will produce only one output book element, with all titles clumped together and all publishers clumped together
  - All books will be processed (as long as one is published before 2000)

Subqueries in return

- Extract book titles and their authors; make title an attribute and rename author to writer

```
<bibliography>
  for $b in document('bib.xml')/bibliography/book
  return
  <book title="{$b/title}">
    for $a in $b/author
    return <writer>{string($a)}</writer>
  </book>
</bibliography>
```

An explicit join

- Find pairs of books that have common author(s)

```
<result>
  for $b1 in document('bib.xml')//book
  for $b2 in document('bib.xml')//book
  where $b1/author = $b2/author
  return
  <pair>
    {$b1/title}
    {$b2/title}
  </pair>
</result>
```

Existentially quantified expressions

(some $var in node-set satisfies condition)

- Can be used in where as a condition
- Find titles of books in which XML is mentioned in some section

```
<result>
  for $b in document('bib.xml')//book
  where (some $section in $b//section satisfies contains(string($section), "XML"))
  return $b/title
</result>
```

Universally quantified expressions

(every $var in node-set satisfies condition)

- Can be used in where as a condition
- Find titles of books in which XML is mentioned in every section

```
<result>
  for $b in document('bib.xml')//book
  where (every $section in $b//section satisfies contains(string($section), "XML"))
  return $b/title
</result>
```
Aggregation

- List each publisher and the average prices of all its books

```xml
<result>
  for $pub in distinct-values(document("bib.xml")//publisher)
  let $price :=
    avg(document("bib.xml")//book[publisher=$pub]/@price)
  return
    <publisherpricing>
      <$pub>
      <avgprice>{$price}</avgprice>
    </publisherpricing>
</result>
```

- `distinct-values(node-set)` removes duplicates
  - Two elements are considered duplicates if their names, attributes, and "normalized contents" are equal (still under active discussion)
- `avg(node-set)` computes the average of `node-set` (assuming each node in `node-set` can be converted to a numeric value)

Tricky semantics

- List titles of all books, sorted by their prices

```xml
<result>
  (document("bib.xml")//book sort by (@price))/title
</result>
```

- What is wrong?
  - A path expression always returns results in document order!
- Correct versions

```xml
<result>
  for $b in document("bib.xml")//book sort by (@price)
  return $b/title
</result>
```

```xml
<result>
  document("bib.xml")//book/title sort by (../@price)
</result>
```

Current version of sorting

As of November 2003

- `sort by` has been ditched
- Add a new `order by` clause in FLWR (which now becomes FLWOR)

```xml
<result>
  for $b in document("bib.xml")//book[@price>100]
  stable order by author[1], title empty least
  return $b
</result>
```

Summary

- Many, many more features not covered in class
- XPath is fairly mature and stable
  - Already a W3C recommendation
  - Implemented in many systems
  - Used in many other standards
- XQuery is still evolving
  - Still a W3C working draft
  - Some vendors are coming out with implementations
  - To become the SQL for XML?
- XQuery versus SQL
  - Where did the join go?
  - Weak typing
  - Strong ordering constraints