XQuery

CPS 116

Introduction to Database Systems

Announcements (October 19)

- ❖ Homework #3 assigned today
  - Due on October 31
- ❖ Project milestone #1 feedbacks available by the end of today, or tomorrow at the latest

XQuery

- ❖ XPath + full-fledged SQL-like query language
- ❖ XQuery expressions can be
  - XPath expressions
  - FLWOR (❀) expressions
  - Quantified expressions
  - Aggregation, sorting, and more…
- ❖ An XQuery expression in general can return a new result XML document
  - Compare with an XPath expression, which always returns a sequence of nodes from the input document or atomic values (boolean, number, string, etc.)

A simple XQuery based on XPath

Find all books with price lower than $50

```
<result>
{
   doc("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
  - `doc("bib.xml")` specifies the document to query
    - Can be omitted if there is a default context document
  - The XPath expression returns a sequence 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

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

- ❖ `for`: loop
  - `$b` ranges over the result sequence, getting one item at a time
- ❖ `let`: assignment
  - `$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 that satisfies `where`

An equivalent formulation

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

```
<result>{
  for $b in doc("bib.xml")/bibliography/book[year<2000]
  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 doc("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
  - 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 := doc("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 $b in doc("bib.xml")/bibliography/book
  return
    <book title="{normalize-space($b/title)}">{
      for $a in $b/author
      return <writer>{string($a)}</writer>
    }</book>
}</bibliography>
```

What happens if we replace it with `$a`?

- `normalize-space(`*string*`)` removes leading and trailing spaces from string, and replaces all internal sequences of white spaces with one white space

An explicit join

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

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

← These are string comparisons, not identity comparisons!

Existentially quantified expressions

`(some $`*var*` in `*collection*` 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 doc("bib.xml")//book
  where (some $section in $b//section satisfies
         contains(string($section), "XML"))
  return $b/title
}</result>
```

Universally quantified expressions

`(every $`*var*` in `*collection*` 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 doc("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

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

  - `distinct-values(`*collection*`)` removes duplicates by value
    - If the collection consists of elements (with no explicitly declared types), they are first converted to strings representing their "normalized contents"
  - `avg(`*collection*`)` computes the average of *collection* (assuming each item in *collection* can be converted to a numeric value)

Sorting (a brief history)

- ❖ XPath always returns a sequence of nodes in original document order
- ❖ `for` loop will respect the ordering in the sequence
- ❖ August 2002 (http://www.w3.org/TR/2002/WD-xquery-20020816/)
  - Introduce an operator `sort by (`*sort-by-expression-list*`)` to output results in a user-specified order
  - Example: list all books with price higher than $100, in order by first author; for books with the same first author, order by title

```
<result>{
  doc("bib.xml")//book[@price>100]
  sort by (author[1], title)
}</result>
```

Tricky semantics

- ❖ List titles of all books, sorted by their prices

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

  - What is wrong?
    - A path expression always returns a sequence of nodes in document order!
  - Correct versions

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

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

Current version of sorting

As of June 2006

- ❖ `sort by` has been ditched
- ❖ Add a new `order by` clause in FLWR (which now becomes FLWOR)
- ❖ Example: list all books with price higher than $100, in order by first author; for books with the same first author, order by title

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

Summary

- ❖ Many, many more features not covered in class
- ❖ XPath is fairly mature and stable
  - 1.0 is already a W3C recommendation
    - Implemented in many systems
    - Used in many other standards
  - 2.0 is being developed jointly with XQuery
- ❖ XQuery is still evolving
  - Still a W3C "candidate" recommendation
  - Many vendors are coming out with implementations
  - Poised to become the SQL for XML

XQuery vs. SQL

- ❖ Where did the join go?
- ❖ Is navigational query going to destroy physical data independence?
- ❖ Strong ordering constraint
  - Can be overridden by `unordered { for… }`
  - Why does that matter?