XQuery

CPS 196.3
Introduction to Database Systems

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

An equivalent formulation

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

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

Another formulation

```xml
<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
    - For example, two result book elements will be created for a book with two 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>
      <!-- Is this query correct? -->
      { $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)

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

```
<result>
  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)
Ordering and sorting

- A path expression always returns a node-set in document order
- for loop will respect the ordering of nodes in a node-set
- Use sort by (sort-by-expression-list) to output results in a user-specified order
- List all books with price higher than $100, in order by first author; for books with the same first author, order by title

```xml
<Result>
  <book>
    <title>List all books with price higher than $100, in order by first author; for books with the same first author, order by title</title>
    <price>$100</price>
    <title>Book Title</title>
    <author>Author Name</author>
  </book>
</Result>
```

A tricky sorting example

- List titles of all books, sorted by their prices

```xml
<Result>
  <book>
    <title>List titles of all books, sorted by their prices</title>
    <price>$100</price>
    <title>Book Title</title>
    <author>Author Name</author>
  </book>
</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