XPath and XQuery

CPS 116
Introduction to Database Systems

Announcements (Thu. Oct. 2)
- Deadline for Homework #2 non-Gradiance part extended to next Tuesday
  - Gradiance part is still due today!
- Midterm next Thursday in class
  - Sample midterm (from last year) available
  - Sample solution will be available next Tuesday
- Project milestone #1 due in 2 weeks!

Query languages for XML
- XPath
  - Path expressions with conditions
  - Building block of other standards (XQuery, XSLT, XLink, XPointer, etc.)
- XQuery
  - XPath + full-fledged SQL-like query language
- XSLT
  - XPath + transformation templates

Example DTD and XML
```xml
<?xml version="1.0"?>
<!DOCTYPE bibliography [ 
<!ELEMENT bibliography (book+)>
<!ELEMENT book (title, author*, publisher?, year?, section*)>
<!ATTLIST book ISBN CDATA #REQUIRED>
<!ATTLIST book price CDATA #IMPLIED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT section (title, content?, section*)>
]>
<bibliography>
  <book ISBN="ISBN-10" price="80.00">
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
    <section>…</section>…
  </book>
…
</bibliography>
```

XPath
- XPath specifies path expressions that match XML data by navigating down (and occasionally up and across) the tree
- Example
  - Query: /bibliography/book/author
    - Like a UNIX path
  - Result: all author elements reachable from root via the path /bibliography/book/author

Basic XPath constructs
- `/*` matches any child element with this tag name
- `@*` matches any attribute

Sample XPath expressions:
- `//section` matches any descendant element or the current element itself
  - matches the current element
  - matches the parent element
Simple XPath examples

- All book titles
  /bibliography/book/title
- All book ISBN numbers
  /bibliography/book/@ISBN
- All title elements, anywhere in the document
  //title
- All section titles, anywhere in the document
  //section/title
- Authors of bibliographical entries (suppose there are
  articles, reports, etc. in addition to books)
  /bibliography/*/author

Predicates in path expressions

- [condition] matches the “current” element if condition evaluates to true on the current element
  - Books with price lower than $50
    /bibliography/book[@price<50]
  - Books with author “Abiteboul”
    /bibliography/book[author='Abiteboul']
  - Books with a publisher child element
    /bibliography/book/publisher
  - Prices of books authored by “Abiteboul”
    /bibliography/book[author='Abiteboul']/@price

More complex predicates

Predicates can have and’s and or’s

- Books with price between $40 and $50
  /bibliography/book[40<=@price and @price<=50]
- Books authored by “Abiteboul” or those with price lower than $50
  /bibliography/book[author="Abiteboul" or @price<50]

Predicates involving node-sets

- There may be multiple authors, so author in general returns a node-set (in XPath terminology)
- The predicate evaluates to true as long as it evaluates true for at least one node in the node-set,
  i.e., at least one author is “Abiteboul”
- Tricky query
  /bibliography/book[author='Abiteboul' and author!='Abiteboul']
  - Will it return any books?

XPath operators and functions

Frequently used in conditions:
  x + y, x - y, x * y, x div y, x mod y
  contains(x, y) true if string x contains string y
  count(node-set) counts the number nodes in node-set
  position() returns the “context position” (roughly, the
  position of the current node in the node-set containing it)
  last() returns the “context size” (roughly, the size of
  the node-set containing the current node)
  name() returns the tag name of the current element

More XPath examples

- All elements whose tag names contain “section” (e.g., “subsection”)
  ///*[contains(name(), 'section')]
- Title of the first section in each book
  /bibliography/book/section[position()=1]/title
  - A shorthand: /bibliography/book/section[1]/title
- Title of the last section in each book
  /bibliography/book/section[position()=last()]/title
- Books with fewer than 10 sections
  /bibliography/book[count(section)<10]
- All elements whose parent’s tag name is not “book”
  ///*[name()!='book']/*
A tricky example

- Suppose that `price` is a child element of `book`, and there may be multiple prices per book
- Books with some price in range [20, 50]
  - How about:
    `/bibliography/book [price >= 20 and price <= 50]`
  - Correct answer:
    `/bibliography/book [price[. >= 20 and . <= 50]]`

De-referencing IDREF's

- `id(identifier)` returns the element with `identifier`
- Suppose that books can reference other books
  - Or simply:

General XPath location steps

- Technically, each XPath query consists of a series of location steps separated by `/`
- Each location step consists of:
  - An axis: one of `self`, `attribute`, `parent`, `child`, `ancestor`, `ancestor-or-self`, `following`, `following-sibling`, `preceding`, `preceding-sibling`, `namespace`
  - A node-test: either a name test (e.g., `book`, `section`, `*`), or a type test (e.g., `text()`, `node()`, `comment()`), separated from the axis by `::`
  - Zero of more predicates (or conditions) enclosed in square brackets
- These reverse axes produce result node-sets in reverse document order; others (forward axes) produce node-sets in document order

Example of verbose syntax

- Verbose (axis, node test, predicate):
- Abbreviated:

Some technical details on evaluation

Given a context node, evaluate a location path as follows:

1. Start with node-set `N = {context node}`
2. For each location step, from left to right:
   - `U ← ∅`
   - For each node `n` in `N`:
     - Using `n` as the context node, compute a node-set `N' from the axis and the node-test
     - Each predicate in turn filters `N'`
       - For each node `n'` in `N'`, evaluate predicate with the following context:
         - Context size: `P` of nodes that passed the first condition
         - Context position: position of the context node within the list of nodes
       - `U ← U ∪ N'`
     - `N ← U`
3. Return `N`
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

```xml
<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

FLWR expressions

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

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

An equivalent formulation

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

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

Another formulation

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

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

Yet another formulation

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

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

- A path expression in XPath returns a sequence of nodes in original document order
- for loop will respect the ordering in the sequence
  - 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
    
    ```xml
    <result>
    doc("bib.xml")//book[@price>100]
    sort by (author[1], title)
    </result>
    ```
Tricky semantics

- List titles of all books, sorted by their prices
  
  ```xml
  <result>
  (doc("bib.xml")//book sort by (@price))/title
  </result>
  ```

  - What is wrong?
    - The last step in the path expression will return nodes in document order!

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

    ```xml
    <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
  ```xml
  <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 very mature and stable
  - Implemented in many systems
  - Used in many other standards
  - Current version is 2.0 (developed jointly with XQuery)
  - Already a W3C recommendation since 1.0
- XQuery has recently been standardized
  - W3C recommendation since January 2007
  - 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?