Lecture 12: XML, XPath, XQuery
Friday, October 25, 2002

Outline

- XML: syntax, semantics, data, DTDs
- XPath
- XQuery

Strongly recommended readings:
- XPath:
  http://java.sun.com/webservices/docs/esa2tutorial/doc/JAXPXMLT2.html
- XQuery:
  http://www.w3.org/TR/xmqlquery-use-cases/
  http://www.xmlportfolio.com/xquery.html

XML Syntax

- tags: book, title, author, ...
- elements: <book>,<book>,<author>...
- elements are nested
- empty element: <red> </red> abbrev. <red/>
- an XML document: single root element

well formed XML document: if it has matching tags

XML Syntax

<book price = "55" currency = "USD">
  <title> Foundations of Databases </title>
  <author> Abiteboul </author>
  ...
  <year> 1995 </year>
</book>

attributes are alternative ways to represent data

XML Syntax

<person id = "o555"> <name> Jane </name> </person>
<person id = "o456"> <name> Mary </name> <children idref="o123 o555"/>
</person>
<person id = "o123" mother = "o456"> <name> John </name> </person>

oids and references in XML are just syntax

XML Semantics: a Tree!

Order matters !!!
XML Data

- XML is self-describing
- Schema elements become part of the data
  - Rational schema: persons(name, phone)
  - In XML <persons>, <name>, <phone> are part of the data, and are repeated many times
- Consequence: XML is much more flexible
- XML = semistructured data

XML is Semi-structured Data

- Missing attributes:
  ```
  <person>
    <name>John</name>
    <phone>1234</phone>
  </person>
  `<no phone`!
  ```

- Could represent in a table with nulls
  ```
  name     phone
  John     1234
  Joe      -
  ```

XML is Semi-structured Data

- Attributes with different types in different objects
  ```
  <person>
    <name>John</name>
    <first>Smith</first>
    <phone>1234</phone>
  </person>
  `<structured name`!

- Nested collections (no 1NF)
- Heterogeneous collections:
  - `<db>` contains both `<book>`s and `<publisher>`s

Relational Data as XML

<table>
<thead>
<tr>
<th>name</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>3634</td>
</tr>
<tr>
<td>Sue</td>
<td>6343</td>
</tr>
<tr>
<td>Dick</td>
<td>6363</td>
</tr>
</tbody>
</table>

Document Type Definitions

- DTD
  - part of the original XML specification
  - an XML document may have a DTD
  - XML document:
    - well-formed = if tags are correctly closed
    - Valid = if it has a DTD and conforms to it
  - validation is useful in data exchange
Very Simple DTD

```xml
<!DOCTYPE company [ 
  <!ELEMENT company ( (person|product)* )> 
  <!ELEMENT person ( ssn, name, office, phone?) > 
  <!ELEMENT ssn (#PCDATA)> 
  <!ELEMENT name (#PCDATA)> 
  <!ELEMENT office (#PCDATA)> 
  <!ELEMENT phone (#PCDATA)> 
  <!ELEMENT product ( pid, name, description? ) > 
  <!ELEMENT pid (#PCDATA)> 
  <!ELEMENT description (#PCDATA)> ]>
```

DTD: The Content Model

- Content model:
  - Complex = a regular expression over other elements
  - Text-only = #PCDATA
  - Empty = EMPTY
  - Any = ANY
  - Mixed content = (#PCDATA | A | B | C)*

Very Simple DTD

Example of valid XML document:

```xml
<company>
  <person> 123456789 </person>
  <name> John </name>
  <office> B432 </office>
  <phone> 1234 </phone>
</company>
```

XML

DTD: Regular Expressions

- sequence
- optional
- Kleene star
- alternation

Sample Data for Queries

```xml
<book>
  <publisher> Addison-Wesley </publisher>
  <author> Serge Abiteboul </author>
  <title> Foundations of Databases </title>
  <year> 1995 </year>
</book>
```
**Data Model for XPath**

```
    The root
       |         |
    bib     book
       |         |
  publish  author     ...
    Addison-Wesley  Serge Abiteboul
```

**XPath: Simple Expressions**

```
/bib/book/year
```

Result:
```
<year> 1995 </year>
<year> 1998 </year>
```

```
/bib/paper/year
```

Result: empty (there were no papers)

**XPath: Restricted Kleene Closure**

```
//author
```

Result:
```
<author> Serge Abiteboul </author>
<author> <first-name> Rick </first-name> <last-name> Hull </last-name> </author>
<author> Victor Viana </author>
<author> Jeffrey D. Ullman </author>
```

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

Result:
```
<first-name> Rick </first-name>
```

**XPath: Text Nodes**

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

Result:
```
Serge Abiteboul
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: Wildcard**

```
//author/*
```

Result:
```
<first-name> Rick </first-name>
<last-name> Hull </last-name>
```

* Matches any element

**XPath: Attribute Nodes**

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

Result: “55”

@price means that price is has to be an attribute
Xpath: Predicates

/bib/book/author[firstname]

Result: <author> <firstname> Rick </firstname>
        <lastname> Hull </lastname>
    </author>

Xpath: More Predicates

/bib/book[@price < “60”]

/bib/book/author[@age < “25”]

/bib/book/author/text()
**XQuery**

Find all book titles published after 1995:

```xquery
FOR $x$ IN document("bib.xml")/bib/book
WHERE $x$/year > 1995
RETURN { $x$/title }
```

Result:

```
<title> abc </title>
<title> def </title>
<title> ghi </title>
```

Find book titles by the coauthors of “Database Theory”:

```xquery
FOR $x$ IN bib/book[@title/text() = "Database Theory"]/author
$y$ IN bib/book[author/text() = $x$/text()]@title
RETURN <answer> { $y$/text() } </answer>
```

Result:

```
<answer> abc </answer>
<answer> def </answer>
<answer> ghi </answer>
```

**XQuery**

Same as before, but eliminate duplicates:

```xquery
FOR $x$ IN bib/book[@title/text() = "Database Theory"]/author
$y$ IN distinct(bib/book[author/text() = $x$/text()]@title)
RETURN <answer> { $y$/text() } </answer>
```

Result:

```
<answer> abc </answer>
<answer> def </answer>
<answer> ghi </answer>
```

**XQuery: Nesting**

For each author of a book by Morgan Kaufmann, list all books she published:

```xquery
FOR $x$ IN distinct(document("bib.xml"))/bib/book[@publisher = "Morgan Kaufmann"]/author
$y$ IN bib/book[author/text() = $x$/text()]
RETURN <result>
{ $y$/title }
</result>
```

Result:

```
<result>
  <author> Jones </author>
  <title> abc </title>
  <title> def </title>
</result>
<result>
  <author> Smith </author>
  <title> ghi </title>
</result>
```

**XQuery**

- FOR $x$ in expr -- binds $x$ to each value in the list expr
- LET $x$ = expr -- binds $x$ to the entire list expr
  – Useful for common subexpressions and for aggregations
XQuery

**XQuery**

Find books whose price is larger than average:

```xquery
LET $a = \text{avg}(\text{/bib/book/price})
FOR $b$ in \text{/bib/book}
WHERE $b/price > \$a$
RETURN { $b$ }
```

Summary:
- FOR-LET-WHERE-RETURN = FLWR
  - FORLET Clauses
    - List of tuples
  - WHERE Clause
    - List of tuples
  - RETURN Clause
    - Instance of XQuery data model

FOR v.s. LET

- FOR
  - Binds node variables → iteration
- LET
  - Binds collection variables → one value

Collections in XQuery

- Ordered and unordered collections
  - /bib/book/author = an ordered collection
  - Distinct/bib/book/author = an unordered collection
- LET $sa = \text{/bib/book}$ → $sa$ is a collection
- $sb/author$ → a collection (several authors...)

```
LET $sa$ = \text{/bib/book}
RETURN <result> { $sa$ } </result>

LET $sb$ IN \text{/bib/book/author}
RETURN <result> { $sb/author$ } </result>
```

=count a (aggregate) function that returns the number of els
Collections in XQuery

What about collections in expressions?

- $b/price → list of n prices
- $b/price * 0.7 → list of n numbers
- $b/price * $b/quantity → list of n x m numbers ??
- $b/price * ($b/quantity1 + $b/quantity2) ≠ $b/price * $b/quantity1 + $b/price * $b/quantity2 !!

Sorting in XQuery

```xml
<publisher_list>
  FOR $p IN distinct(document("bib.xml")//publisher)
  RETURN <publisher> <name> { $p/text() } <name>,
  FOR $b IN document("bib.xml")//book[publisher = $p]
  RETURN <book> <title>, <price>
  </book> SORTBY(price DESCENDING)
</publisher_list>
```

If-Then-Else

```xml
<holding>
  FOR $h IN //holding
  RETURN <holding> 
    { $h/text(),
      IF $h/@type = "Journal" THEN $h/editor
      ELSE $h/author
    }
  </holding> SORTBY(title)
</holding>
```

Existential Quantifiers

```xml
<book>
  FOR $b IN //book
  WHERE SOME $p IN $b//para SATISFIES contains($p, "sailing") AND contains($p, "windsurfing")
  RETURN { $b/text() }
</book>
```

Universal Quantifiers

```xml
<book>
  FOR $b IN //book
  WHERE EVERY $p IN $b//para SATISFIES contains($p, "sailing")
  RETURN { $b/text() }
</book>
```

Other Stuff in XQuery

- **BEFORE and AFTER**
  - for dealing with order in the input
- **FILTER**
  - deletes some edges in the result tree
- **Recursive functions**
  - Currently: arbitrary recursion
  - Perhaps more restrictions in the future?