XML & XPath

CS 445
Fall 2010
Structure in data representation

• Relational data is highly structured
  – structure is defined by the schema
  – good for system design
  – good for precise query semantics / answers

• Structure can be limiting
  – authoring is constrained: schema-first
  – changes to structure not easy
  – querying constrained: must know schema
  – data exchange hard: integration of diff schema

Some reasons why more data is not in databases
Structured data - Databases

Unstructured Text - Documents

Semistructured Data

WWW
XML data

<data>
  <person id="o555">
    <name>Mary</name>
    <address>
      <street>Maple</street>
      <no>345</no>
      <city>Seattle</city>
    </address>
  </person>
  <person>
    <name>John</name>
    <address>Thailand</address>
    <phone>23456</phone>
  </person>
</data>
Need for loose structure

- Evolving, unknown, or irregular structure
- Integration of structured, but heterogeneous data sources
- Textual data with tags and links
- Combination of data models
XML: Syntax
XML Syntax

- **tags:** `book`, `title`, `author`, …
- **start tag:** `<book>`, **end tag:** `</book>`
- **elements:** `<book>…</book>`, `<author>…</author>`
- **elements are nested**
- **empty element:** `<red></red>` abbrv. `<red/>`
- **an XML document:** single *root element*

An XML document is **well formed** if it has matching tags
XML Syntax

```xml
<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

```xml
<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
  – Relational 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

Some real data:
http://www.cs.washington.edu/research/xmldatasets/
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>

XML:

```xml
<person>
  <row>
    <name>John</name>
    <phone>3634</phone>
  </row>
  <row>
    <name>Sue</name>
    <phone>6343</phone>
  </row>
  <row>
    <name>Dick</name>
    <phone>6363</phone>
  </row>
</person>
```
XML is Semi-structured Data

• Missing attributes:

  `<person>  <name> John </name>
       <phone> 1234 </phone>
  </person>`

  `<person>  <name> Joe </name>
           </person>`

← no phone!

• Could represent in a table with nulls

<table>
<thead>
<tr>
<th>name</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>1234</td>
</tr>
<tr>
<td>Joe</td>
<td>-</td>
</tr>
</tbody>
</table>
XML is Semi-structured Data

• Repeated attributes

  
  <person> <name> Mary</name> 
  <phone>2345</phone> 
  <phone>3456</phone> 
  </person>

  ← two phones!

• Impossible in tables:

<table>
<thead>
<tr>
<th>name</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>2345</td>
</tr>
</tbody>
</table>

??

?
XML is Semi-structured Data

• Attributes with different types in different objects

<person> <name> <first> John </first> 
        <last> Smith </last> 
   </name> 
   <phone>1234</phone> 
</person>

← structured name!

• Nested collections (non 1NF)
• Heterogeneous collections:
  – <db> contains both <book>s and <publisher>s
Represent as tree

```xml
<A>
  <A> 1  </A>
  <B>
    <B> 2  </B>
    <B> 3  </B>
  </B>
</A>

<A>
  <B> 4  </B>
  <A>
    <A> 5  </A>
    <A> 6  </A>
  </A>
</A>
</A>
```
Querying XML Data

• Querying XML has two components
  – Selecting data
    • pattern matching on structural & path properties
    • typical selection conditions
  – Construct output, or transform data
    • construct new elements
    • restructure
    • order
Querying XML Data

• XPath = simple navigation through the tree

• XQuery = the SQL of XML

• XSLT = recursive traversal
Querying XML

How do you query a directed graph? a tree?

The standard approach used by many XML, semistructured-data, and object query languages:

• Define some sort of a template describing traversals from the root of the directed graph

• In XML, the basis of this template is called an XPath
XPath is widely used

- XML Schema uses simple XPaths in defining keys and uniqueness constraints
- XQuery
- XSLT
- XLink and XPointer, hyperlinks for XML
In its simplest form, an XPath is like a path in a file system:

/mypath/subpath/*/morepath

• The XPath returns a node set representing the XML nodes (and their subtrees) at the end of the path

• XPaths can have node tests at the end, returning only particular node types, e.g., text(), element(), attribute()

• XPath is fundamentally an ordered language: it can query in order-aware fashion, and it returns nodes in order
Sample Data for Queries

```xml
<bib>
  <book>
    <publisher> Addison-Wesley </publisher>
    <author> Serge Abiteboul </author>
    <author> <first-name> Rick </first-name> <last-name> Hull </last-name> </author>
    <author> Victor Vianu </author>
    <title> Foundations of Databases </title>
    <year> 1995 </year>
  </book>
  <book price="55">
    <publisher> Freeman </publisher>
    <author> Jeffrey D. Ullman </author>
    <title> Principles of Database and Knowledge Base Systems </title>
    <year> 1998 </year>
  </book>
</bib>
```
Data Model for XPath

The root

The root element

bib

book

book

publisher

author

Addison-Wesley

Serge Abiteboul
XPath

/bib/book/year
/bib/paper/year
//author
/bib//first-name
//author/*
/bib/book/@price
/bib/book/author[firstname]
/bib/book/author[firstname][address[./zip][city]]/lastname
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 Vianu </author>
    <author> Jeffrey D. Ullman </author>

/bib//first-name

Result:  <first-name> Rick </first-name>
Xpath: Text Nodes

Result:  Serge Abiteboul
          Victor Vianu
          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 has to be an attribute
Xpath: Predicates

/bib/book/author[firstname]

Result: <author> <first-name> Rick </first-name> <last-name> Hull </last-name> </author>
Xpath: More Predicates

```
/bib/book/author[firstname][address[.//zip][city]]/lastname
```

Result: `<lastname> ... </lastname>

`<lastname> ... </lastname>`

Note that two predicates next to one another are combined conjunctively. In this case `author[firstname][address[..][..]]` requires that author have both a firstname AND an address child (with additional conditions on the address child).
Xpath: More Predicates

/bib/book[@price < 60]

/bib/book[author/@age < 25]

/bib/book[author/text()]
Context Nodes and Relative Paths

XPath has a notion of a context node: it’s analogous to a current directory

- “.” represents this context node
- “..” represents the parent node
- We can express relative paths:
  
  subpath/sub-subpath/../../../ gets us back to the context node

- By default, the document root is the context node
dot in XPath qualifiers

- //author
- //author[first-name]
- //author[./first-name]
- //author[/first-name]
- //author[//first-name]
- //author[.///first-name]

<table>
<thead>
<tr>
<th>equivalent</th>
<th>qualifier starts at root</th>
</tr>
</thead>
</table>
### Xpath: Summary

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bib</code></td>
<td>matches a <code>bib</code> element</td>
</tr>
<tr>
<td><code>*</code></td>
<td>matches any element</td>
</tr>
<tr>
<td><code>/</code></td>
<td>matches the <code>root</code> element</td>
</tr>
<tr>
<td><code>/bib</code></td>
<td>matches a <code>bib</code> element under <code>root</code></td>
</tr>
<tr>
<td><code>bib/paper</code></td>
<td>matches a <code>paper</code> in <code>bib</code></td>
</tr>
<tr>
<td><code>bib//paper</code></td>
<td>matches a <code>paper</code> in <code>bib</code>, at any depth</td>
</tr>
<tr>
<td><code>//paper</code></td>
<td>matches a <code>paper</code> at any depth</td>
</tr>
<tr>
<td>`paper</td>
<td>book`</td>
</tr>
<tr>
<td><code>@price</code></td>
<td>matches a <code>price</code> attribute</td>
</tr>
<tr>
<td><code>bib/book/@price</code></td>
<td>matches <code>price</code> attribute in <code>book</code>, in <code>bib</code></td>
</tr>
<tr>
<td><code>bib/book[./@price=&quot;55&quot;]/author/lastname</code></td>
<td>matches…</td>
</tr>
</tbody>
</table>
Overview of Research issues

• Data modeling and normalization
• Query language design
• Storage & publishing of XML
  – XML → Relations
  – Relations → XML
• Theoretical work
  – expressiveness
  – containment, type checking
• Query execution & optimization