Semistructured data and XML

CS 645
2010
Outline of lecture

• Semistructured data
  – History and motivation
• XML: syntax and typing
• Querying XML data
  – XPath
  – XQuery
• Overview of research issues
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

Semistructured Data

WWW

Unstructured Text - Documents
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 is the preeminent format for semi-structured data

XML is the confluence of many factors:

- The Web needed a more declarative format for data
- Documents needed a mechanism for extended tags
- Database people needed a more flexible interchange format
- It’s parsable even if we don’t know what it means!

Original expectation:

- The whole web would go to XML instead of HTML

Today’s reality:

- Not so… But XML is used all over “under the covers”
Why DB People Like XML

Can get data from all sorts of sources

- Allows us to touch data we don’t own!
- This was actually a huge change in the DB community

Blends schema and data into one format

- Unlike relational model, where we need schema first
- … But too little schema can be a drawback, too!
XML: Syntax & Typing


XML Syntax

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

An XML document is **well formed** 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 !!!

<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>
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>
```

- Impossible in tables:

<table>
<thead>
<tr>
<th>name</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>2345</td>
</tr>
<tr>
<td></td>
<td>3456</td>
</tr>
</tbody>
</table>

← two phones!
XML is Semi-structured Data

• Attributes with different types in different objects

  \[
  \text{<person>} \text{<name>} \text{<first> John } \text{<first> } \text{<last> Smith } \text{<last>} \\
  \text{</name>} \\
  \text{<phone>1234</phone>} \\
  \text{</person>}
  \]

  ← structured name!

• Nested collections (non 1NF)

• Heterogeneous collections:
  – \text{<db>} contains both \text{<book>}s and \text{<publisher>}s
Data Typing in XML

- Data typing in the relational model: schema
- Data typing in XML
  - Much more complex
  - Typing restricts valid trees that can occur
    - theoretical foundation: tree languages
  - Practical methods:
    - DTD (Document Type Descriptor)
    - XML Schema
Document Type Definitions

DTD

- Part of the original XML specification
- To be replaced by XML Schema
  - Much more complex
- 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
<!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)>
]>
Basics of XML Schema

Need to use the XML Schema namespace (generally named xsd)

- **simpleTypes** are a way of restricting domains on scalars
  - Can define a simpleType based on integer, with values within a particular range

- **complexType**es are a way of defining element/attribute structures
  - Basically equivalent to !ELEMENT, but more powerful
  - Specify sequence, choice between child elements
  - Specify minOccurs and maxOccurs (default 1)

- Must associate an element/attribute with a simpleType, or an element with a complexType
Simple Schema Example

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="mastersthesis" type="ThesisType"/>
  <xsd:complexType name="ThesisType">
    <xsd:attribute name="mdate" type="xsd:date"/>
    <xsd:attribute name="key" type="xsd:string"/>
    <xsd:attribute name="advisor" type="xsd:string"/>
    <xsd:sequence>
      <xsd:element name="author" type="xsd:string"/>
      <xsd:element name="title" type="xsd:string"/>
      <xsd:element name="year" type="xsd:integer"/>
      <xsd:element name="school" type="xsd:string"/>
      <xsd:element name="committeemember" type="CommitteeType" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```
Designing an XML Schema/DTD

Not as formalized as relational data design

- We can still use ER diagrams to break into entity, relationship sets
- ER diagrams have extensions for “aggregation” – treating smaller diagrams as entities – and for composite attributes
- Note that often we already have our data in relations and need to design the XML schema to export them!

Generally orient the XML tree around the “central” objects

Decision: element vs. attribute

- Element if it has its own properties, or if you *might* have more than one of them
- Attribute if it is a single property – or perhaps not!
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
  - next week

- XSLT = recursive traversal
  - will not discuss in class
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

<bib>
  <book>
    <publisher> Addison-Wesley </publisher>
    <author> Serge Abiteboul </author>
    <author> Rick </author>
    <author> Hull </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 element

The root

Addison-Wesley

Serge Abiteboul

publisher

author

book

book
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>
Xpath: More Predicates

/bib/book[@price < 60]

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

/bib/book[author/text()]
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]  equivalent
- //author[./first-name]
- //author[/first-name]  qualifier starts at root
- //author[/first-name]
- //author[../first-name]
# Xpath: Summary

<table>
<thead>
<tr>
<th>Expression</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td>bib</td>
<td>matches a bib element</td>
</tr>
<tr>
<td>*</td>
<td>matches any element</td>
</tr>
<tr>
<td>/</td>
<td>matches the root element</td>
</tr>
<tr>
<td>/bib</td>
<td>matches a bib element under root</td>
</tr>
<tr>
<td>bib/paper</td>
<td>matches a paper in bib</td>
</tr>
<tr>
<td>bib///paper</td>
<td>matches a paper in bib, at any depth</td>
</tr>
<tr>
<td>//paper</td>
<td>matches a paper at any depth</td>
</tr>
<tr>
<td>paper</td>
<td>book</td>
</tr>
<tr>
<td>@price</td>
<td>matches a price attribute</td>
</tr>
<tr>
<td>bib/book/@price</td>
<td>matches price attribute in book, in bib</td>
</tr>
<tr>
<td>bib/book[.@price&lt;“55”]/author/lastname</td>
<td>matches…</td>
</tr>
</tbody>
</table>
Axes: More Complex Traversals

Thus far, we’ve seen XPath expressions that go down the tree

- But we might want to go up, left, right, etc.
- These are expressed with so-called axes:
  - `self::path-step`
  - `child::path-step`    `parent::path-step`
  - `descendant::path-step`    `ancestor::path-step`
  - `descendant-or-self::path-step`    `ancestor-or-self::path-step`
  - `preceding-sibling::path-step`    `following-sibling::path-step`
  - `preceding::path-step`    `following::path-step`
- The previous XPaths we saw were in “abbreviated form”
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
XPath containment

• XPath expressions return sets of nodes
  – $P_1(doc) = \text{node set}$
• $P_1 \subseteq P_2$ if $P_1(doc) \subseteq P_2(doc)$ for all $doc$

• Limited features $/,$ $//,$ $*,$ $[,]$
• XPath expressions = tree patterns

/a[a]/*[b]/c
Deciding containment by tree matching

Deciding containment for simple XPath expressions in coNP-complete

Implementation: http://www.ifis.uni-luebeck.de/projects/XPathContainment/containmentFrame.htm