XML

May 2nd, 2002

Agenda

- XML as a data model
- Querying XML
- Manipulating XML
- A lot of discussion, politics and stories.

More Facts About XML

- Every database vendor has an XML page:
  - www.oracle.com/xml
  - www.microsoft.com/xml
  - www.ibm.com/xml
- Many applications are just fancier Websites
- But, most importantly, XML enables data sharing on the Web – hence our interest

What is XML?

From HTML to XML

HTML describes the presentation: easy for humans

XML describes the content: easy for applications

HTML is hard for applications

XML:

```xml
<bibliography>
  <book>
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
  </book>
  <book>
    <title>Data on the Web</title>
    <author>Abiteboul</author>
    <author>Buneman</author>
    <author>Suciu</author>
    <publisher>Morgan Kaufmann</publisher>
    <year>1999</year>
  </book>
</bibliography>
```

HTML:

```html
<h1>Bibliography</h1>
<p><i>Foundations of Databases</i>
Abiteboul, Hull, Vianu
Addison Wesley, 1995</p>
<p>Data on the Web
Abiteboul, Buneman, Suciu
Morgan Kaufmann, 1999</p>
```
XML

• eXtensible Markup Language
• Roots: comes from SGML
  – A very nasty language
• After the roots: a format for sharing data
• Emerging format for data exchange on the Web and between applications

Web Services

• A new paradigm for creating distributed applications?
• Systems communicate via messages, contracts.
• Example: order processing system.
• MS .NET, J2EE – some of the platforms
• XML – a part of the story; the data format.

XML Applications

• Sharing data between different components of an application.
• Archive data in text files.
• EDI: electronic data exchange:
  – Transactions between banks
  – Producers and suppliers sharing product data (auctions)
  – Extranets: building relationships between companies
• Scientists sharing data about experiments.

XML Syntax

• Very simple:
  
  `<db>
  <book>
  <title>Complete Guide to DB2</title>
  <author>Chamberlin</author>
  </book>
  <book>
  <title>Transaction Processing</title>
  <author>Bernstein</author>
  <author>Newcomer</author>
  </book>
  <publisher>
  <name>Morgan Kaufman</name>
  <state>CA</state>
  </publisher>
  </db>`

XML Terminology

• tags: book, title, author, ...
• start tag: `<book>`, end tag: `</book>`
• start tags must correspond to end tags, and conversely

XML Terminology

• an element: everything between tags
  – example element
    `<title>Complete Guide to DB2</title>`
  – example element
    `<author>Chamberlin</author>`

• elements may be nested
• empty element: `<red>` abbreviated `<red>`
• an XML document has a unique root element

well formed XML document: if it has matching tags
The XML Tree

Tags on nodes
Data values on leaves

More XML Syntax: Attributes

```
<book price = "55" currency = "USD">
<title> Complete Guide to DB2 </title>
<author> Chamberlin </author>
<year> 1998 </year>
</book>
```

price, currency are called attributes

Replacing Attributes with Elements

```
<book>
<title> Complete Guide to DB2 </title>
<author> Chamberlin </author>
<year> 1998 </year>
<price> 55 </price>
<currency> USD </currency>
</book>
```

attributes are alternative ways to represent data

“Types” (or “Schemas”) for XML

- Document Type Definition – DTD
- Define a grammar for the XML document, but we use it as substitute for types/schemas
- Will be replaced by XML-Schema (will extend DTDs)

An Example DTD

```
<!DOCTYPE db [ 
<!ELEMENT db ((book|publisher)*)>
<!ELEMENT book (title,author*,year?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
]
```

- PCDATA means Parsed Character Data (a mouthful for string)

More on DTDs: Attributes

```
<book price = "55" language = "English">
<title> Complete Guide to DB2 </title>
<author> Chamberlin </author>
</book>
```

The type:
CDATA = string
ID = a key
IDREF = a foreign key otherwise used

Default declaration:
#REQUIRED = required
#IMPLIED = optional
#FIXED = fixed (rarely used)
**DTDs as Grammars**

Same thing as:

```
<db ::= (book|publisher)*
  book ::= (title,author*,year?)
  title ::= string
  author ::= string
  year ::= string
  publisher ::= string
```

- A DTD is an EBNF (Extended BNF) grammar
- An XML tree is precisely a derivation tree

XML Documents that have a DTD and conform to it are called **valid**

**More on DTDs as Grammars**

```xml
<!DOCTYPE paper [
  <!ELEMENT paper (section*)>
  <!ELEMENT section ((title,section*) | text)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT text (#PCDATA)>
]>
```

```
<paper>
  <section>
    <title>…</title>
  </section>
  <section>
    …
  </section>
</paper>
```

XML documents can be nested arbitrarily deep

**XML for Representing Data**

```
<?xml version="1.0" encoding="UTF-8"?>
<persons>
  <row>
    <name>John</name>
    <phone>3634</phone>
  </row>
  <row>
    <name>Sue</name>
    <phone>6343</phone>
  </row>
  <row>
    <name>Dick</name>
    <phone>6363</phone>
  </row>
</persons>
```

**XML vs Data Models**

- **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
- **XML = semistructured data**

**Semi-structured Data Explained**

- Missing attributes:

  ```
  <person>
    <name>John</name>
    <phone>1234</phone>
  </person>
  <person>
    <name>Joe</name>
  </person>
  <person>
    <name>Mary</name>
    <phone>2345</phone>
    <phone>3456</phone>
  </person>
  ```

  - no phone !
  - two phones !

- Repeated attributes

  ```
  <person>
    <name>Bill</name>
    <phone>1234</phone>
    <phone>2345</phone>
  </person>
  ```

  - \(\) repeated attribute

**Semistructured Data Explained**

- Attributes with different types in different objects

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

  - structured name !

- Nested collections (no 1NF)
- Heterogeneous collections:

  ```
  <db>
    <book>
      …
    </book>
    <publisher>
      …
    </publisher>
  </db>
  ```
XML Data v.s. E/R, ODL, Relational

- Q: is XML better or worse?
- A: serves different purposes
  - E/R, ODL, Relational models:
    - For centralized processing, when we control the data
    - XML:
      - Data sharing between different systems
      - We do not have control over the entire data
      - E.g. on the Web
    - DO NOT use XML to model your data! Use E/R, ODL, or relational instead.

Exporting Relational Data to XML

product —> makes —> company

- Product(pid, name, weight)
- Company(cid, name, address)
- Makes(pid, cid, price)

Export data grouped by companies

Export Data by Products

The DTD

Which One Do We Choose?

- The structure of the XML data is determined by agreement, with our partners, or dictated by committees
- Many XML dialects (called applications)
- XML Data is often nested, irregular, etc
- No normal forms for XML ☺
XML Query Languages

- Xpath
- XML-QL
- Xquery

XPath

- Syntax for XML document navigation and node selection
- A recommendation of the W3C (i.e. a standard)
- Building block for other W3C standards:
  - XSL Transformations (XSLT)
  - XQuery
  - XML Link (XLink)
  - XML Pointer (XPointer)
- Was originally part of XSL – “XSL pattern language”

An Example of XML Data

```xml
<bib>
  <book>
    <publisher>Addison-Wesley</publisher>
    <author>Serge Abiteboul</author>
    <author><first-name>Rick</first-name><last-name>Hull</last-name></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>
```

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

```
/bib/book/author/text()
```
Result: Serge Abiteboul
Jeffrey D. Ullman

Rick Hull doesn’t appear because he has firstname, lastname
Xpath: Wildcard

//author/*

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

* Matches any element

Xpath: Attribute Nodes

/bib/book/@price

Result: “55”

@price means that price is to be an attribute

Xpath: Qualifiers

/bib/book/author[firstname]

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

Xpath: More Qualifiers

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

Result: …<last-name>…</last-name>

Xpath: More Qualifiers

/bib/book[@price < "60"]
/bib/book[author/@age < "25"]
/bib/book[author/text()]

Xpath: Summary

bib matches a bib element
* matches any element
/ matches the root element
/bib matches a bib element under root
bib/paper matches a paper in bib
bib/paper matches a paper in bib, at any depth
//paper matches a paper at any depth
paper/book matches a paper or a book
@price matches a price attribute
bib/book/@price matches price attribute in book, in bib
bib/book[@price="55"]/author/lastname matches…
Xpath: More Details

• An Xpath expression, p, establishes a relation between:
  – A context node, and
  – A node in the answer set
• In other words, p denotes a function:
  – \( S[p] : \text{Nodes} \rightarrow \{ \text{Nodes} \} \)
• Examples:
  – author/firstname
  – . = self
  – . = parent
  – part/*/*/subpart/../name = what does it mean?

The Root and the Root

• `<bib>` `<paper>` 1 `<paper>` `<paper>` 2 `<paper>` `<bib>`
• `<bib>` is the “document element”
• The “root” is above `<bib>`
• `/` = returns the document element
• `/` = returns the root
• Why ? Because we may have comments before and after `<bib`; they become siblings of `<bib>`
• This is advanced xmlogy

Xpath: More Details

• We can navigate along 13 axes:
  ancestor
  ancestor-or-self
  attribute
  child
  descendant
  descendant-or-self
  following
  following-sibling
  namespace
  parent
  preceding
  preceding-sibling
  self

• Examples:
  – child::author/child:lastname = author/lastname
  – child::author/descendant::zip = author//zip
  – child::author/parent::* = author/..
  – child::author/attribute::age = author/@age

XQuery

• Based on Quilt
  (which is based on XML-QL)
• Check out the W3C web site for the latest.
• XML Query data model
  – Ordered !

FLWR (“Flower”) Expressions

FOR ... LET... FOR... LET...
WHERE...
RETURN...
XQuery

Find all book titles published after 1995:

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

Result:
```
<title>a b c</title>
<title>def</title>
<title>ghi</title>
```

XQuery

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

```
FOR $a IN distinct(document("bib.xml")/bib/book/publisher="Morgan Kaufmann")/author
RETURN <result>$a,
FOR $t IN /bib/book[author=$a]/title
RETURN $t</result>
```

Result:
```
<result><author>Jones</author><title>a b c</title><title>d e f</title></result>
<result><author>Smith</author><title>g h i</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

```
count = a (aggregate) function that returns the number of elms
```

XQuery

Find books whose price is larger than average:

```
LET $a = avg(document("bib.xml")/bib/book/price)
FOR $b in document("bib.xml")/bib/book
WHERE $b/price > $a
RETURN $b
```

distinct = a function that eliminates duplicates
XQuery

Summary:
• FOR-LET-WHERE-RETURN = FLWR

FOR/LET 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 $a = /bib/book/author$ is a collection
• $b/author$ → a collection (several authors...)

Sorting in XQuery

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

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 ??
• $b/price * (b/quant1 + b/quant2) \neq b/price * b/quant1 + b/price * b/quant2$  !!
Sorting in XQuery

- Sorting arguments: refer to the name space of the RETURN clause, not the FOR clause

If-Then-Else

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

Existential Quantifiers

```xquery
FOR $b$ IN //book
WHERE SOME $p$ IN $b//para SATISFIES contains($p, "sailing")
AND contains($p, "windsurfing")
RETURN $b/title
```

Universal Quantifiers

```xquery
FOR $b$ IN //book
WHERE EVERY $p$ IN $b//para SATISFIES contains($p, "sailing")
RETURN $b/title
```

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?

Processing XML Data

- Do we really need to process XML data?
  What are we processing XML for?
- How are we going to do it? Use existing technology?
- Are there other processing paradigms that we need to consider?
Query Processing For XML

- Approach 1: store XML in a relational database. Translate an XML-QL/Quilt query into a set of SQL queries.
  - Leverage 20 years of research & development.
- Approach 2: store XML in an object-oriented database system.
  - OO model is closest to XML, but systems do not perform well and are not well accepted.
- Approach 3: build a native XML query processing engine.
  - Still in the research phase; see Zack next week.

Relational Approach

- Step 1: given a DTD, create a relational schema.
- Step 2: map the XML document into tuples in the relational database.
- Step 3: given a query Q in Xquery, translate it to a set of queries P over the relational database.
- Step 4: translate the tuples returned from the relational database into XML elements.

Which Relational Schema?

- The key question! Affects performance.
- No magic solution.
- Some options:
  - The EDGE table: put everything in one table
  - The Attribute tables: create a table for every tag name.
  - The inlining method: inline as much data into the tables.

An Example DTD

```xml
<!DOCTYPE db [
<!ELEMENT db ((book|publisher)*)>
<!ELEMENT book (title,author*,year?)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT publisher (name, state)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ATTLIST book pub IDREF #IMPLIED>
]>
```

Recall: The XML Tree

- Tags on nodes
- Data values on leaves

The Edge Approach

<table>
<thead>
<tr>
<th>sourceID</th>
<th>tag</th>
<th>destID</th>
<th>destValue</th>
</tr>
</thead>
</table>

- Don’t need a DTD.
- Very simple to implement.
The Attribute Approach

Book
- rootID
- bookId

Title
- bookID
- title

Author
- bookID
- author

Publisher
- rootID
- pubID

PubName
- bookID
- pubID
- pubName

PubState
- pubID
- state

The In-lining Approach

Book
- bookID
- title
- pubName
- pubState

BookAuthor
- bookID
- author

Publisher
- sourceID
- tag
- destID
- destValue

Let the Querying Begin!

- Matching data using elements patterns.

```
FOR $t IN document(bib.xml)/book/[author="bernstein"]/author/title
RETURN <bernsteinBook> $t </bernsteinBook>
```

The Edge Approach

```
SELECT e3.destValue
FROM Ea se 1 ,Ea se 2 ,Ea se 3
WHERE
e1.tag = "book" and
e1.destID=e2.sourceID and
e2.tag="title" and
e1.destID=e3.sourceID and
e3.tag="author" and
e2.author="Bernstein"
```

The Attribute Approach

```
SELECT Title.title
FROM Book, Title, Author
WHERE
  Book.bookID = Author.bookID and
  Book.bookID = Title.bookID and
  Author.author = "Bernstein"
```

The In-lining Approach

```
SELECT Book.title
FROM Book, BookAuthor
WHERE
  Book.bookID =BookAuthor.bookID and
  BookAuthor.author = "Bernstein"
```
A Challenge: Reconstructing Elements

- Matching data using elements patterns.
  SELECT $b$ IN document(bib.xml)/book/[author="bernstein"]
  RETURN
  <bernsteinBook> $b </bernsteinBook>

Reconstructing XML Elements

- Matching data using elements patterns.
  WHERE <book>
    <author> Bernstein </author>
    <title> $t$ </title>
  </book> ELEMENT-AS $e$
  IN “www.a.b.c/bib.xml”
  CONSTRUCT
  $e$

Some Open Questions

- Native query processing for XML
- To order or not to order?
- Combining IR-style keyword queries with DB-style structured queries
- Updates
- Automatic selection of a relational schema
- How should we extend relational engines to better support XML storage and querying?