XML

May 2nd, 2002

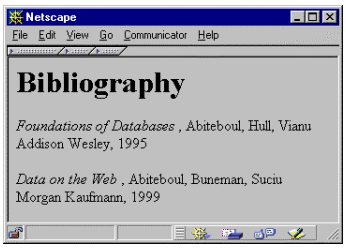
Agenda

- 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

HTML

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

HTML is hard for applications

XML

```
<bibliography>
    <book>   <title> Foundations… </title>
             <author> Abiteboul </author>
             <author> Hull </author>
             <author> Vianu </author>
             <publisher> Addison Wesley </publisher>
             <year> 1995 </year>
    </book>
    …
</bibliography>
```

XML describes the content: easy for applications

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

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.

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 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

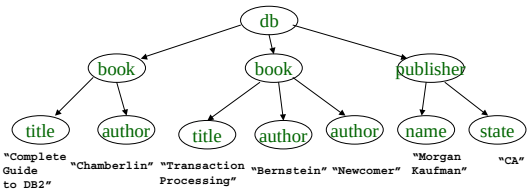
- an element: everything between tags
  - example element:
    <title>Complete Guide to DB2</title>
  - example element:

```
<book> <title> Complete Guide to DB2 </title>
       <author>Chamberlin</author>
</book>
```

- elements may be nested
- empty element: <red></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

```
<!DOCTYPE db [
  <!ELEMENT db   ((book|publisher)*)>
  <!ELEMENT book (title,author*,year?)>
  . . .
  <!ATTLIS book   price      CDATA  #REQUIRED
                  language CDATA #IMPLIED>
  <!ATTLIS author phone     CDATA #IMPLIED>
]>
```

Default declaration:
#REQUIRED=required
#IMPLIED=optional
#FIXED=fixed (rarely used)

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

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

DTDs as Grammars

Same thing as:

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

- A DTD is a 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

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

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

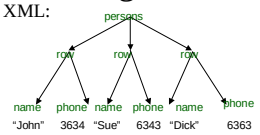
XML documents can be nested arbitrarily deep

XML for Representing Data

persons

name	phone
John	3634
Sue	6343
Dick	6363

XML:




```
<persons>
    <row> <name>John</name>
          <phone> 3634</phone></row>
    <row> <name>Sue</name>
          <phone> 6343</phone>
    <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>
```

← no phone !

- Repeated attributes

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

← two phones !

Semistructured Data Explained

- Attributes with different types in different objects

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

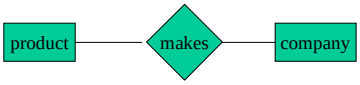
← structured name !

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

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

Export data grouped by companies

```
<db><company>   <name> GizmoWorks </name>
                <address> Tacoma </address>
                <product> <name> gizmo </name>
                          <price> 19.99 </price>
                </product>
                <product> …</product>
      …
    </company>
    <company>   <name> Bang </name>
                <address> Kirkland </address>
                <product> <name> gizmo </name>
                 <price> 22.99 </price>
                </product>
                 …
    </company>
    …
</db>
```

Redundant representation of products

The DTD

```
<!ELEMENT db (company*)>
<!ELEMENT company (name, address, product*)>
<!ELEMENT product (name,price)>
<!ELEMENT name    (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT price     (#PCDATA)>
```

Export Data by Products

```
<db> <product> <name> Gizmo </name>
               <manufacturer>
                     <name> GizmoWorks </name>
                     <price> 19.99 </price>
                     <address> Tacoma </address>
               </manufacturer>
               <manufacturer>
                      <name> Bang </name>
                      <price> 22.99 </price>
                      <address> Kirkland </address>
               </manufacturer>
                …
     </product>
     <product> <name> OneClick </name> …
</db>
```

Redundant Representation of companies

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

An Example of XML Data

```
<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>
```

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"

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>
<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: Qualifiers

/bib/book/author[firstname]

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

Xpath: More Qualifiers

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

Result: <lastname> … </lastname>
<lastname> … </lastname>

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] : Nodes -> {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

- /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

Xpath: More Details

- 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> abc </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>
```

distinct = a function that eliminates duplicates

XQuery

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

```
<big_publishers>
    FOR $p IN distinct(document("bib.xml")//publisher)
    LET $b := document("bib.xml")/book[publisher = $p]
    WHERE count($b) > 100
    RETURN $p
</big_publishers>
```

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

XQuery

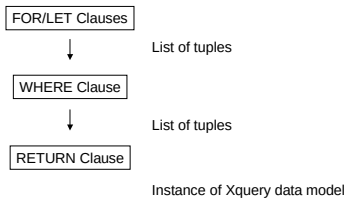
Find books whose price is larger than average:

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

XQuery

Summary:

- FOR-LET-WHERE-RETURN = FLWR



FOR v.s. LET

FOR

- Binds *node variables* → iteration

LET

- Binds *collection variables* → one value

FOR v.s. LET

```
FOR $x IN document("bib.xml")/bib/book
RETURN <result> $x </result>
```

Returns:

```
<result> <book>...</book></result>
<result> <book>...</book></result>
<result> <book>...</book></result>
...
```

```
LET $x IN document("bib.xml")/bib/book
RETURN <result> $x </result>
```

Returns:

```
<result> <book>...</book>
         <book>...</book>
         <book>...</book>
         ...
</result>
```

Collections in XQuery

- Ordered and unordered collections
  - /bib/book/author = an ordered collection
  - Distinct(/bib/book/author) = an unordered collection
- LET $a = /bib/book → $a is a collection
- $b/author → a collection (several authors...)

```
RETURN <result> $b/author </result>
```

Returns:

```
<result> <author>...</author>
         <author>...</author>
         <author>...</author>
         ...
</result>
```

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) ≠ $b/price * $b/quant1 + $b/price * $b/quant2 !!

Sorting in XQuery

```
<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>
                    $b/title ,
                    $b/price
                  </book> SORTBY(price DESCENDING)
         </publisher> SORTBY(name)
</publisher_list>
```

Sorting in XQuery

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

If-Then-Else

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

Existential Quantifiers

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

Universal Quantifiers

```
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.

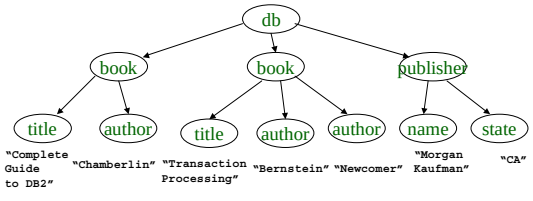
Which Relational Schema?

- 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

```
<!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



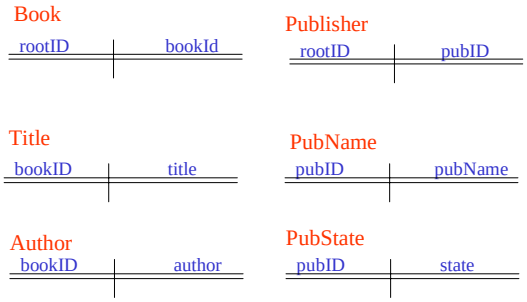
Tags on nodes
Data values on leaves

The Edge Approach

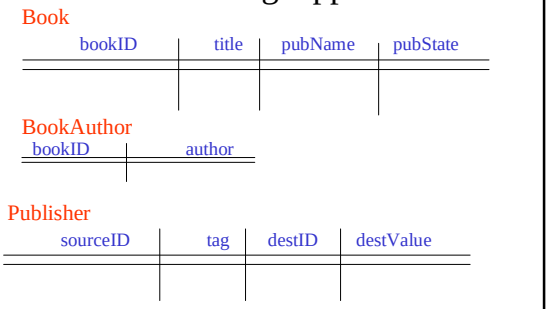
sourceID	tag	destID	destValue

- Don't need a DTD.
- Very simple to implement.

The Attribute Approach



The In-lining Approach



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     E as e1, E as e2, E as e3
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.**

```
FOR   $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?