Lecture 11
XML

Wednesday, Oct. 24, 2001

Outline

- XML:
  - Syntax, DTDs (*Data on the Web*, 3.1)
  - Semistructured data in XML (3.2)
  - Publishing XML (8.3.1), Storing XML (8.2.1)

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

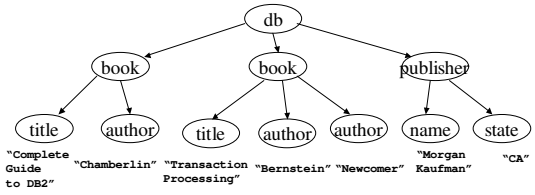
- an element: everything between tags
  - example element:

    <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
- Being replaced by XML-Schema (extends 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

Example XML

```
<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>
```

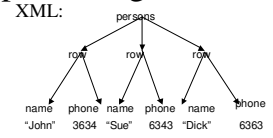
XML documents can be nested arbitrarily deep

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
  - Reational 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 !

- Could represent in a table with nulls

name	phone
John	1234
Joe	-

Semi-structured Data Explained

- Repeated attributes

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

← two phones !

- Impossible in tables:

name	phone	
Mary	2345	3456

???

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 !

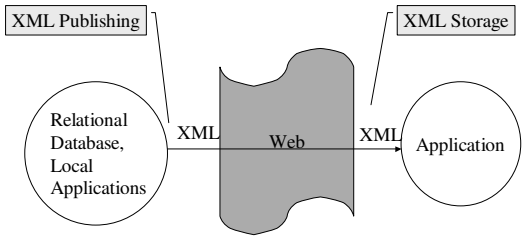
- Nested collections (no 1NF)
- Heterogeneous collections:
  - <db> contains both <book>s and <publisher>s

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.

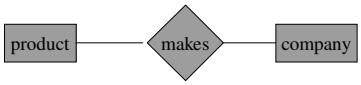
19

Two XML Applications



20

XML Publishing from Relational Databases



- Relational schema:

  Product(pid, name, weight)

  Company(cid, name, address)

  Makes(pid, cid, price)

21

XML Publishing from Relational Databases

```
<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>
```

Group by companies

Redundant representation of products

22

XML Publishing from Relational Databases

The DTD:

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

23

XML Publishing from Relational Databases

```
<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>
```

Group by products

Redundant Representation of companies

24

XML Publishing from Relational Databases

How do we choose the output structure ?

- Determined by agreement, with our partners, or dictated by committees
  - XML dialects (called *applications*) = DTDs
- XML Data is often nested, irregular, etc
- No normal forms for XML

25

XML Storage

- Often the XML data is small and is parsed directly into the application (DOM API)
- Sometimes it is big, and we need to store it in a database
- The XML storage problem:
  - How do we choose the schema of the database ?
- Much harder than XML publishing (why ?)

26

XML Storage

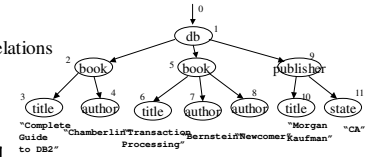
Two solutions:

- Edge relation
- Schema derived from DTD

27

XML Storage

1. Edge (and value) relations



Edge

Source	Tag	Dest
0	db	1
1	book	2
2	title	3
2	author	4
1	book	5
5	title	6
5	author	7
. . .	. . .	. . .

Value

Source	Val
3	Complete guide . . .
4	Chamberlin
6	. . .
. . .	. . .

28

XML Storage

Edge relation summary:

- Same relational schema for every XML document:
  - Edge(Source, Tag, Dest)
  - Value(Source, Val)
- Inefficient:
  - Repeat tags multiple times
  - Need many joins to reconstruct data

29

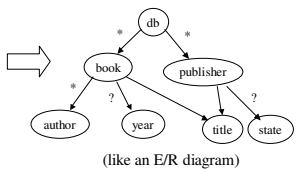
XML Storage

2. Use the DTD to derive relational schema

DTD

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

DTD Graph



(like an E/R diagram)

Relational schema:

book(<u>bid</u>, year, title)
publisher(<u>pid</u>, title, state)
author(<u>aid</u>, bid, value)

30

XML Storage

DTD to relations (summary):

- Much like converting an E/R diagram to a relational schema
- More efficient than the edge table
- But need DTD
- Problems when the DTD changes

31

Other XML Topics

- XML API:
  - DOM = "Document Object Model"
- XML languages:
  - Xpath – will discuss in class
  - Xquery – will discuss in class
  - XSLT
- XML Schema
- Xlink
- SOAP

Available from www.w3.org
(but don't spend rest of your life reading those standards !)

32

Research on XML Data Management at UW

- Processing:
  - Query languages (XML-QL, a precursor of Xquery)
  - Tukwila
  - XML updates
- XML publishing/storage
  - SilkRoute
  - STORED
- XML tools
  - Compressor: Xmill
  - Toolkit
- Theory:
  - Typechecking
  - Xpath containment

33