XML

Semistructured Data
Extensible Markup Language
Document Type Definitions

Adapted from Lecture notes by Jeff Ullman @ Stanford

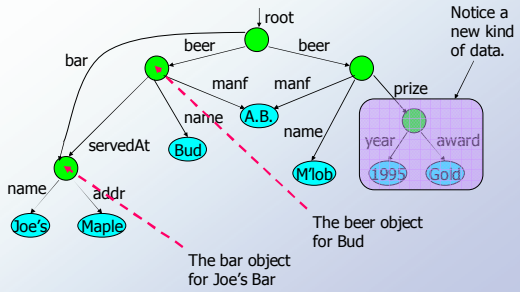
Semistructured Data

- Another data model, based on trees.
- Motivation: flexible representation of data.
  - Often, data comes from multiple sources with differences in notation, meaning, etc.
- Motivation: sharing of *documents* among systems and databases.

Graphs of Semistructured Data

- Nodes = objects.
- Labels on arcs (attributes, relationships).
- Atomic values at leaf nodes (nodes with no arcs out).
- Flexibility: no restriction on:
  - Labels out of a node.
  - Number of successors with a given label.

Example: Data Graph



XML

- XML = *Extensible Markup Language*.
- While HTML uses tags for formatting (e.g., "italic"), XML uses tags for semantics (e.g., "this is an address").
- Key idea: create tag sets for a domain (e.g., genomics), and translate all data into properly tagged XML documents.

Well-Formed and Valid XML

- *Well-Formed XML* allows you to invent your own tags.
  - Similar to labels in semistructured data.
- *Valid XML* involves a DTD (*Document Type Definition*), a grammar for tags.

Well-Formed XML

- Start the document with a *declaration*, surrounded by <?xml … ?> .
- Normal declaration is:

```
<?xml version = "1.0"
  standalone = "yes" ?>
```

  - "Standalone" = "no DTD provided."
- Balance of document is a *root tag* surrounding nested tags.

7

Tags

- Tags, as in HTML, are normally matched pairs, as <FOO> … </FOO> .
- Tags may be nested arbitrarily.
- XML tags are case sensitive.

8

Example: Well-Formed XML

<?xml version = "1.0" standalone = "yes" ?>
<BARS>
  <BAR><NAME>Joe's Bar</NAME>
    <BEER><NAME>Bud</NAME>
      <PRICE>2.50</PRICE></BEER>
    <BEER><NAME>Miller</NAME>
      <PRICE>3.00</PRICE></BEER>
  </BAR>
  <BAR> …
</BARS>

A NAME subobject

A BEER subobject

9

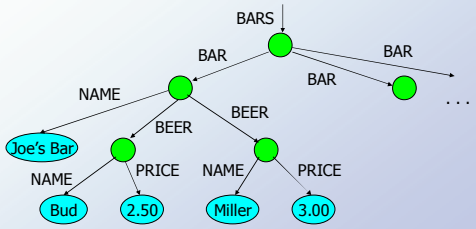
XML and Semistructured Data

- Well-Formed XML with nested tags is exactly the same idea as trees of semistructured data.
- We shall see that XML also enables nontree structures, as does the semistructured data model.

10

Example

- The <BARS> XML document is:



11

DTD Structure

```
<!DOCTYPE <root tag> [
  <!ELEMENT  <name>(<components>)>
  . . . more elements . . .
]>
```

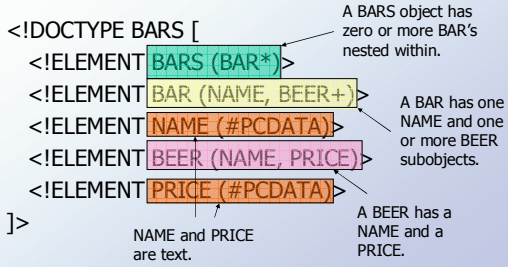
12

DTD Elements

- The description of an element consists of its name (tag), and a parenthesized description of any nested tags.
  - Includes order of subtags and their multiplicity.
- Leaves (text elements) have #PCDATA (*Parsed Character DATA*) in place of nested tags.

13

Example: DTD



14

Element Descriptions

- Subtags must appear in order shown.
- A tag may be followed by a symbol to indicate its multiplicity.
  - * = zero or more.
  - + = one or more.
  - ? = zero or one.
- Symbol | can connect alternative sequences of tags.

15

Example: Element Description

- A name is an optional title (e.g., "Prof."), a first name, and a last name, in that order, or it is an IP address:

```
<!ELEMENT NAME (
  (TITLE?, FIRST, LAST) | IPADDR
)>
```

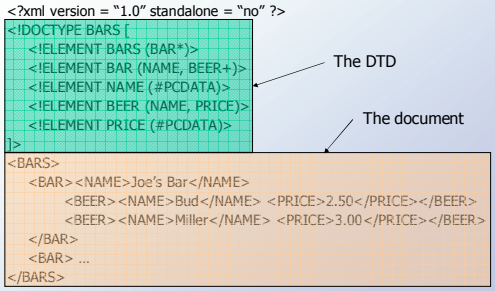
16

Use of DTD's

1. Set standalone = "no".
2. Either:
   a) Include the DTD as a preamble of the XML document, or
   b) Follow DOCTYPE and the <root tag> by SYSTEM and a path to the file where the DTD can be found.

17

Example (a)



18

Example (b)

- Assume the BARS DTD is in file bar.dtd.

```
<?xml version = "1.0" standalone = "no" ?>
<!DOCTYPE BARS SYSTEM "bar.dtd">
<BARS>
   <BAR><NAME>Joe's Bar</NAME>
        <BEER><NAME>Bud</NAME>
              <PRICE>2.50</PRICE></BEER>
        <BEER><NAME>Miller</NAME>
              <PRICE>3.00</PRICE></BEER>
   </BAR>
   <BAR> …
</BARS>
```

Get the DTD from the file bar.dtd

19

XML and Relational DB

- We now know the basics of XML, but how does it interact with relational DB, in particular
  - How to input XML to relational DB?
  - How to retrieve XML from relational DB?

20

Map XML to Relational DB

- It is easy if the DTD of the XML has been designed according to an existing relational DB schema
  - Simply parse the date according to the design
- Otherwise, requires an mapping algorithm to design a database schema that is compatible with the XML document as specified in the DTD

21

Map XML to Relational DB

- The mapping algorithm can be complicated since XML is more flexible that relational model
  - Needs to take care of lots of details
  - Still an on-going research topic
  - A good starting point: www.rpbourret.com/xml/DTDToDatabase.htm

22

Extract XML from Relational DB

- Much easier, because relational mode is more structured than XML
- To generate a DTD from a relation model:
  1. Define a scope, select a subset S of relations to be included in the scope, and remove unnecessary attributes
  2. Select a relation R in S as the root, and call subroutine LIST(R)

23

Subroutine LIST

LIST(R):

- Remove R from S, and create an element-list L = <!ELEMENT R (…)> including all attributes of R that are not marked
- If there is another relation R' in S, such that and R' has a foreign key k referencing R, then add R' * to L, mark the attributes of R' in k and call LIST(R')
- If there is another relation R' in S, such that R has a foreign key k referencing R', then replace the foreign key of R in L by R' and call LIST(R')

Because they are implied by R

Because they are implied by R'

24

Generate DTD (continues)

3. Finally, for each undefined element in the lists, generate an element type with PCDATA-only content

25

The Example

- Map the following relational model to a DTD:
  - Bar (bar_name, addr)
  - Sell (bar, beer, price)
  - Beer (beer_name, manf)

26

The DTD

```
<!DOCTYPE Bars [
<!ELEMENT Bars (Bar*)>
<!ELEMENT Bar (bar_name, addr, Sell*)>
<!ELEMENT bar_name (#PCDATA)>
<!ELEMENT addr (#PCDATA)>
<!ELEMENT Sell (Beer, price)>
<!ELEMENT price (#PCDATA)>
<!ELEMENT Beer (beer_name, manf)>
<!ELEMENT beer_name (#PCDATA)>
<!ELEMENT manf (#PCDATA)>
]>
```

Generated following the steps

27

A Sample XML Using the DTD

- Bars.xml

28

Group exercise

- Using the same relational model, write a DTD with the relation Beer as the root
  - Bar (bar_name, addr)
  - Sell (bar, beer, price)
  - Beer (beer_name, manf)

29

ID and IDREF

- What is the problem with the above XML document?
  - Redundancies
- How to eliminate the redundancies?
  - Use attributes of elements: ID and IDREF

30

Attributes

- Opening tags in XML can have *attributes*.
- In a DTD,

```
<!ATTLIST E...>
```

declares an attribute for element *E*, along with its datatype.

31

Example: Attributes

- Bars can have an attribute `kind`, a character string describing the bar.

```
<!ELEMENT BAR (NAME, BEER*)>
<!ATTLIST BAR kind CDATA
    #IMPLIED>
```

Character string type; no tags

Attribute is optional opposite: #REQUIRED

32

Example: Attribute Use

- In a document that allows BAR tags, we might see:

```
<BAR kind = "sushi">
  <NAME>Akasaka</NAME>
  <BEER><NAME>Sapporo</NAME>
        <PRICE>5.00</PRICE></BEER>
  ...
</BAR>
```

Note attribute values are quoted

33

ID's and IDREF's

- Attributes can be pointers from one object to another.
  - Compare to HTML's NAME = "foo" and HREF = "#foo".
- Allows the structure of an XML document to be a general graph, rather than just a tree.

34

Creating ID's

- Give an element *E* an attribute *a* of type ID.

```
<!ATTLIST E a ID …>
```

- When using tag <*E*> in an XML document, give its attribute *a* a unique value (do not include space and quote).

```
<E  a = "xyz">
```

35

Creating IDREF's

- To allow objects of type *F* to refer to another object with an ID attribute, give *F* an attribute of type IDREF.

```
<!ATTLIST F b IDREF …>
```

- Or, let the attribute have type IDREFS, so the *F*–object can refer to any number of other objects.

36

The Example

```
<!DOCTYPE Bars [
<!ELEMENT Bars (Bar*)>

<!ELEMENT Bar (bar_name, addr, Sell*)>
<!ELEMENT bar_name (#PCDATA)>
<!ELEMENT addr (#PCDATA)>

<!ELEMENT Sell (Beer, price)>
<!ELEMENT price (#PCDATA)>

<!ELEMENT Beer (beer_name, manf)>
<!ELEMENT beer_name (#PCDATA)>
<!ELEMENT manf (#PCDATA)>
]>
```

<!ELEMENT Sell (price)>
<!ATTLIST Sell beer IDREF #REQUIRED>

<!ELEMENT Beer (manf)>
<!ATTLIST BEER beer_name ID #REQUIRED>

A Sample XML Using the DTD

- Bars2.xml

Avoid Redundancy in DTD

1. Use the procedure discussed last time to generate DTD (with redundancies)
2. For an element type R has another element type R' in its ELEMENT list because of a foreign key constraint, then
   - Remove R' from the ELEMENT list of R, add an IDREF to the ATTLIST of R;
   - Move the key of R' from the ELEMENT list of R' to the ATTLIST of R and make it type ID

Group Exercise

- Suppose we want to retrieve from the following relational DB a XML that contains the name and phone of all drinkers, and for each drinker, the beers he/she likes, and their price at each bar. Write a DTD for this purpose. Avoid unnecessary redundancies.

  Beers(name, manf)
  Bars(name, addr, phone)
  Drinkers(name, addr, phone)
  Likes(drinker, beer)
  Sells(bar, beer, price)
  Frequents(drinker, bar)

Limitation of ID and IDREF

- You can regard ID attributes as primary keys and IDREF attributes as foreign keys
- But they are quite limited
  - ID and IDREF cannot represents composite primary and foreign keys
  - They are not scoped; IDREF can refer to any ID in the same XML doc

Integrity Constraints

- If ID and IDREF are limited, why don't we simply specify keys and foreign keys in DTD just like what we did with relations?
  - The problem of checking whether a given specification is consistent, i.e., make sure that there is at least some XML document satisfies a given specification, is undecidable (Fan, Libkin, 2002)

Alternatives

- Alternatives to DTD:
  - XML schema
  - XML Data
- They generally support more expressive specifications for key and foreign keys, but the problem of consistency checking on them are still open

43