

Database – Slide 18

Plan

1. Information integration: important new application that motivates what follows.
2. Semistructured data: a new data model designed to cope with problems of information integration.
3. XML: a new Web standard that is essentially semistructured data.
4. XQUERY: an emerging standard query language for XML data.

Information Integration

Problem: related data exists in many places. They talk about the same things, but differ in model, schema, conventions (*e.g.*, terminology).

Example

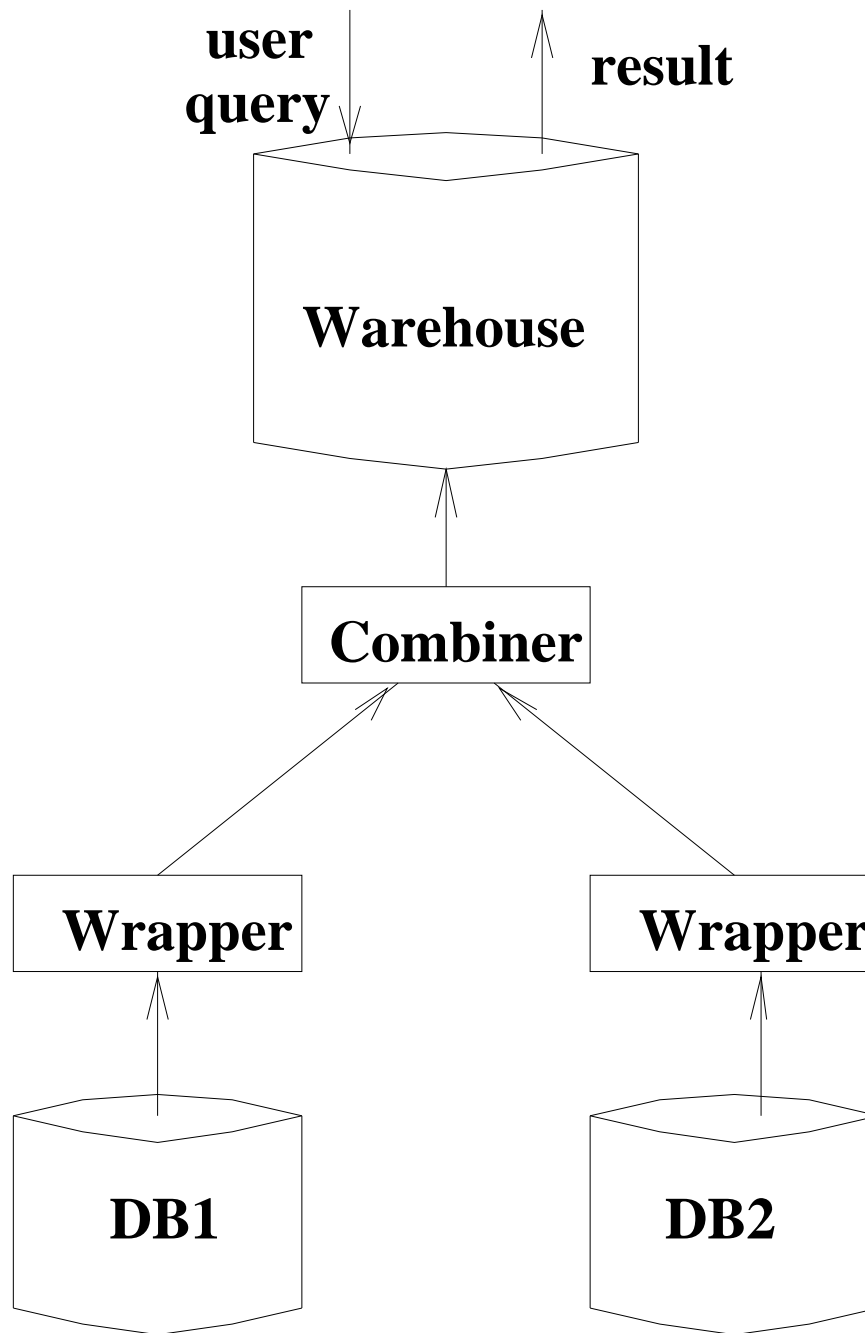
In the real world, every bar has its own database.

- Some may have relations like beer-price; others have an Microsoft Word file from which the menu is printed.
- Some keep phones of manufacturers but not addresses.
- Some distinguish beers and ales; others do not.

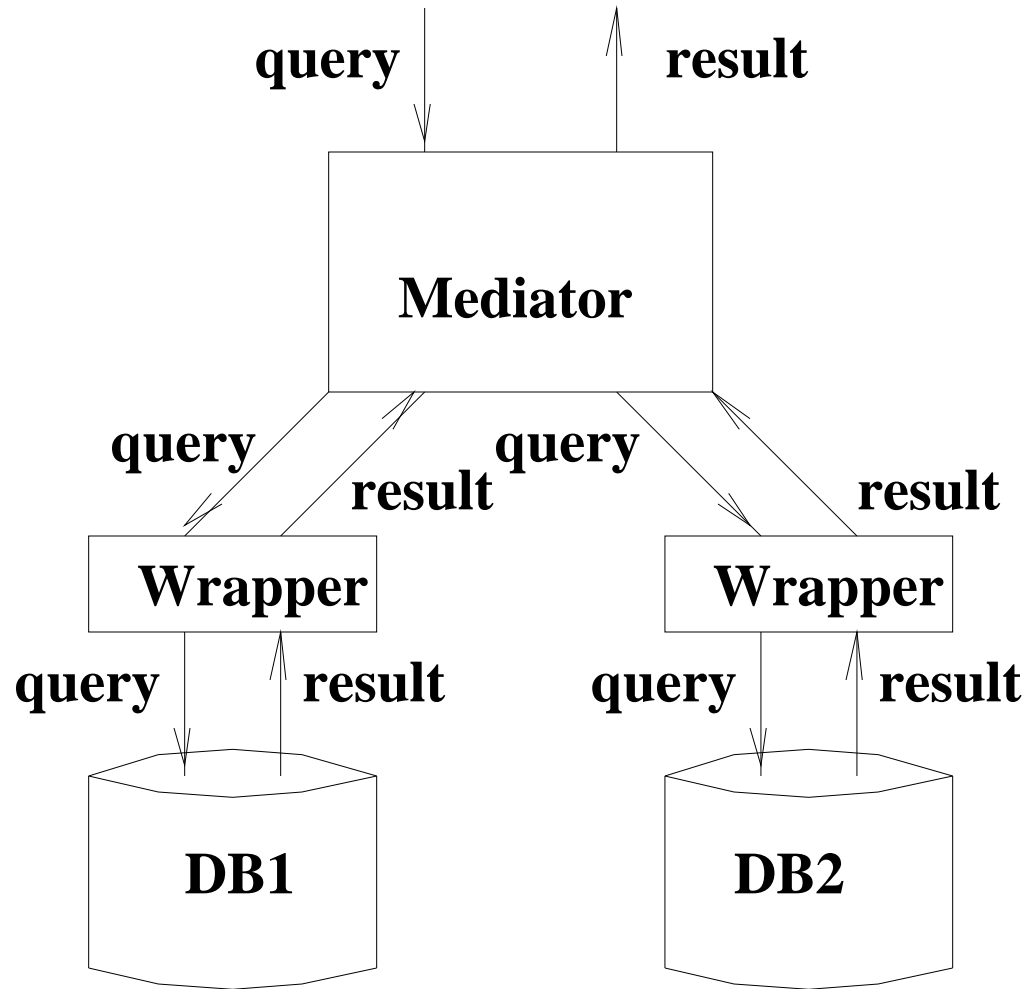
Two approaches

1. *Warehousing*: Make copies of information at each data source centrally.
 - ◆ Reconstruct data daily/weekly/monthly, but do not try to keep it up-to-date.
2. *Mediation*: Create a view of all information, but do not make copies.
 - ◆ Answer queries by sending appropriate queries to sources.

Warehousing



Mediation



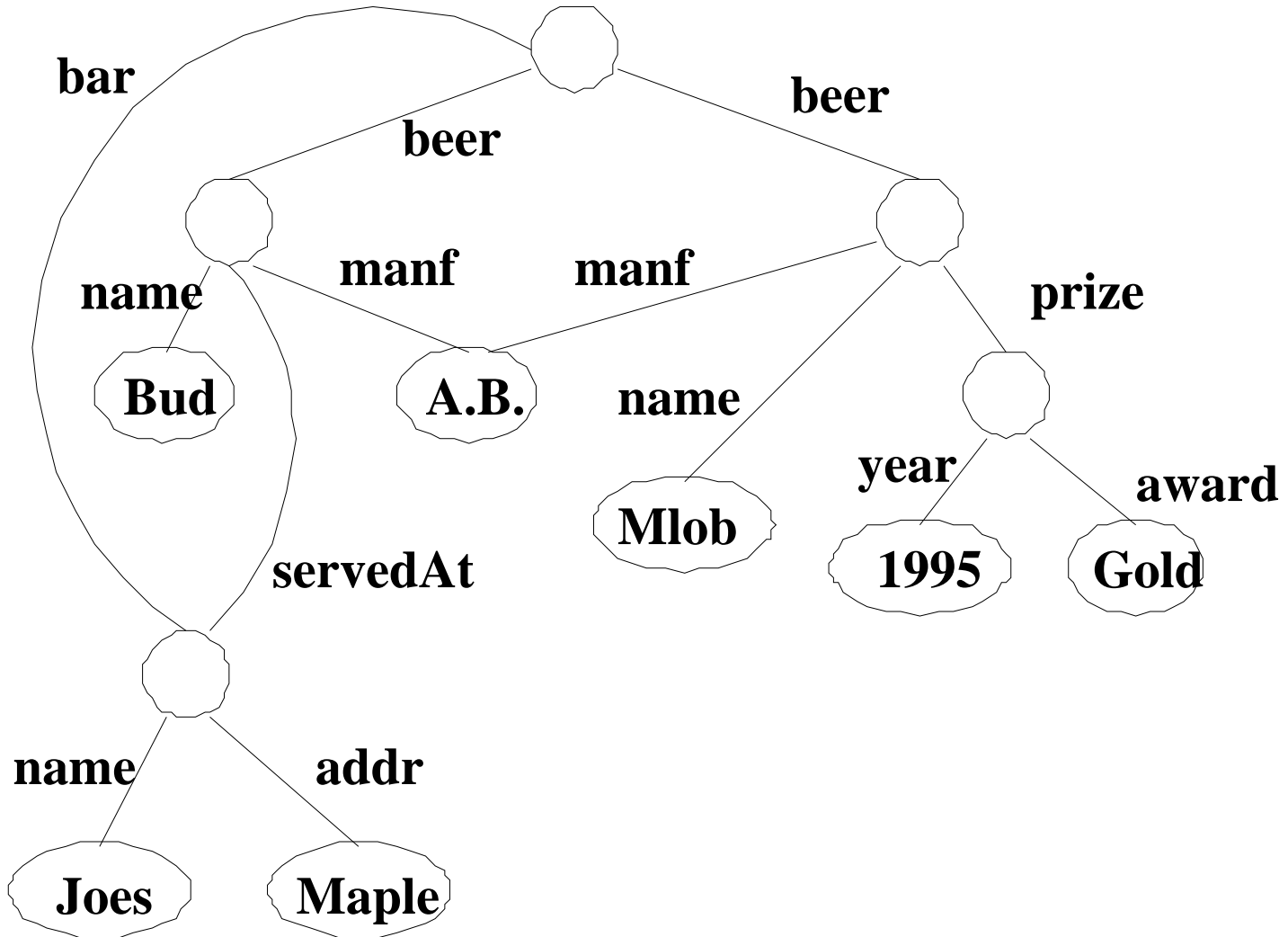
Semistructured Data

- A different kind of data model, more suited to information-integration applications than either relational or OO.
 - ◆ Think of “objects,” but with the type of an object its own business rather than the business of the class to which it belongs.
 - ◆ Allows information from several sources, with related but different properties, to be fit together in one whole.
- Major application: XML documents.

Graph Representation of Semistructured Data

- Nodes = objects.
- Nodes connected in a general rooted graph structure.
- Labels on arcs.
- Atomic values on leaf nodes.
- Big deal: no restriction on labels (roughly = attributes).
 - ◆ Zero, one, or many children of a given label type are all OK.

Example



XML (Extensible Markup Language)

HTML uses tags for formatting (*e.g.*, “*italic*”).

XML uses tags for semantics (*e.g.*, “this is an address”).

- Two modes:
 1. *Well-formed XML* allows you to invent your own tags, much like labels in semistructured data.
 2. *Valid XML* involves a DTD (Document Type Definition) that tells the labels and gives a grammar for how they may be nested.

Well-Formed XML

1. Declaration = `<? . . . ?>` .

◆ Normal declaration is

```
<? XML VERSION = "1.0" STANDALONE = "yes" ?>
```

◆ “Standalone” means that there is no DTD specified.

2. *Root* tag surrounds the entire balance of the document.

◆ `<FOO>` is balanced by `</FOO>`, as in HTML.

3. Any balanced structure of tags OK.

◆ Option of tags that don't *require* balance, like `<P>` in HTML.

Example

```
<?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>
```

Document Type Definitions (DTD)

Essentially a grammar describing the legal nesting of tags.

- Intention is that DTD's will be standards for a domain, used by everyone preparing or using data in that domain.
 - ◆ Example: a DTD for describing protein structure; a DTD for describing bar menus, etc.

Gross Structure of a DTD

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

Elements of a DTD

An *element* is a name (its tag) and a parenthesized description of tags within an element.

- Special case: (#PCDATA) after an element name means it is text.

Example

```
<!DOCTYPE Bars [  
    <!ELEMENT BARS (BAR*)>  
    <!ELEMENT BAR (NAME, BEER+)>  
    <!ELEMENT NAME (#PCDATA)>  
    <!ELEMENT BEER (NAME, PRICE)>  
    <!ELEMENT PRICE (#PCDATA)>  
>
```

Components

- Each element name is a tag.
- Its components are the tags that appear nested within, in the order specified.
- Multiplicity of a tag is controlled by:
 - a) * = zero or more of.
 - b) + = one or more of.
 - c) ? = zero or one of.
- In addition, | = “or.”

Using a DTD

1. Set `STANDALONE = "no"`.
2. Either
 - a) Include the DTD as a preamble, or
 - b) Follow the XML tag by a `DOCTYPE` declaration with the root tag, the keyword `SYSTEM`, and a file where the DTD can be found.

Example of (a)

```
<?XML VERSION = "1.0" STANDALONE = "no"?>
<!DOCTYPE Bars [
  <!ELEMENT BARS (BAR*)>
  <!ELEMENT BAR (NAME, BEER+)>
  <!ELEMENT NAME (#PCDATA)>
  <!ELEMENT BEER (NAME, PRICE)>
  <!ELEMENT PRICE (#PCDATA)>
]>
<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>
```

Example of (b)

Suppose our 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>
```

Attribute Lists

Opening tags can have “arguments” that appear within the tag, in analogy to constructs like `` in HTML.

- Keyword `!ATTLIST` introduces a list of attributes and their types for a given element.

Example

```
<!ELEMENT BAR (NAME BEER*) >
<!ATTLIST BAR
  type = "sushi" | "sports" | "other"
>
```

- Bar objects can have a type, and the value of that type is limited to the three strings shown.
- Example of use:

```
<BAR type = "sushi" >
. . .
</BAR >
```

ID's and IDREF's

These are pointers from one object to another, analogous to `NAME = foo` and `HREF = #foo` in HTML.

- Allows the structure of an XML document to be a general graph, rather than just a tree.
- An attribute of type `ID` can be used to give the object (string between opening and closing tags) a unique string identifier.
- An attribute of type `IDREF` refers to some object by its identifier.
 - ◆ Also `IDREFS` to allow multiple object references within one tag.

Example

Let us include in our `Bars` document type elements that are the manufacturers of beers, and have each beer object link, with an `IDREF`, to the proper manufacturer object.

```
<!DOCTYPE Bars [  
  <!ELEMENT BARS (BAR* )>  
  <!ELEMENT BAR (NAME , BEER+ )>  
  <!ELEMENT NAME (#PCDATA )>  
  <!ELEMENT MANF (ADDR )>  
    <!ATTLIST MANF (name ID )>  
  <!ELEMENT ADDR (#PCDATA )>  
  <!ELEMENT BEER (NAME , PRICE )>  
    <!ATTLIST BEER (manf = IDREF )>  
  <!ELEMENT PRICE (#PCDATA )>  
>
```

XQUERY

Emerging standard for querying XML documents.

Basic form:

FOR <variables ranging over sets of elements>

WHERE <condition>

RETURN <set of elements> ;

- Sets of elements described by paths, consisting of:
 1. URL, if necessary.
 2. Element names forming a path in the semistructured data graph, e.g., //BAR/NAME =
“start at any BAR node and go to a NAME child.”
 3. Ending condition of the form
[<condition about subelements, @attributes, and values>]

Example

The file `http://www.cse.ucsc.edu/bars.xml`:

```
<?XML VERSION = "1.0" STANDALONE = "no"?>
<!DOCTYPE Bars SYSTEM "bar.dtd">
<BARS>
  <BAR type = "sports">
    <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 type = "sushi">
    <NAME>Homma's</NAME>
    <BEER><NAME>Sapporo</NAME>
      <PRICE>4.00</PRICE></BEER>
  </BAR> ...
</BARS>
```

XQUERY Query

Find the prices charged for Bud by sports bars that serve Miller.

```
FOR $ba IN document("http://www.cse.ucsc.edu/bars.html")
    //BAR[@type = "sports"],
    $be IN $ba/BEER[NAME = "Bud"]
WHERE $ba/BEER/[NAME = "Miller"]
RETURN $be/PRICE;
```