# Semi-Structured Data and XML

Edited By M. Naci Akkøk spring 2003, 2004 and 2005.
Based upon slides by Pål Halvorsen (26/2-2002).
Contains slides made by Arthur M. Keller and Vera Goebel.

# Information Integration - I

Problem: related data exists in many places. They talk about the same things, but differ in model, schema, conventions (*e.g.*, terminology).
*How should one retrieve data from different places?*

Examples:

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.

# Information Integration - II

- Warehousing:
  Make copies of information at each data source centrally, combine into a global schema. Query data stored at the warehouse. Reconstruct (recopy) data daily/weekly/monthly, but do not try to keep it up-to-date.

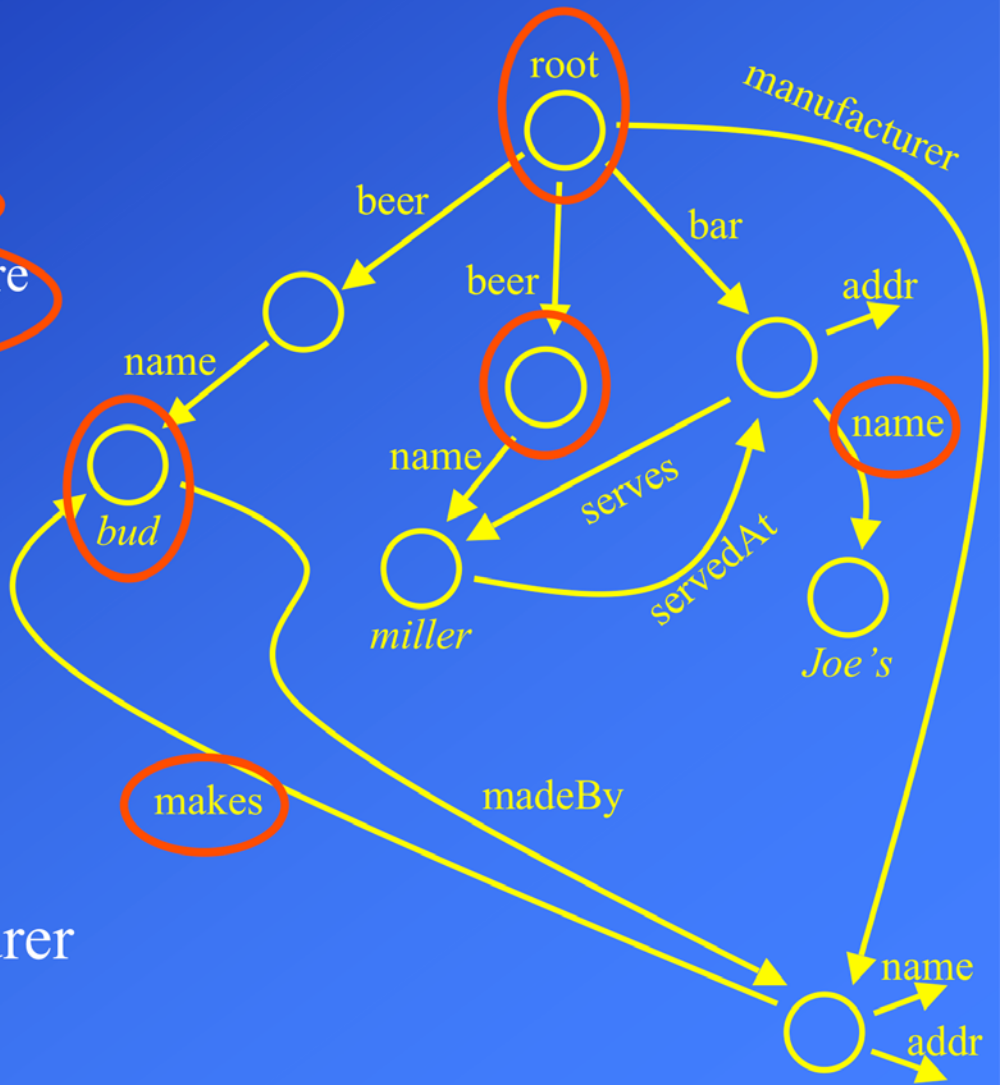
- Mediation:
  Create a view of all information, but do not make copies. Answer queries by sending appropriate queries to sources (no local data).

# Semi-Structured Data

- Semi-structured data model allows information from several sources, with related but different properties, to be fit together in one whole. Thus, suitable for
  - integration of databases
  - sharing information on the Web
- Semi-structured data is data that may be irregular or incomplete and have a structure that may change rapidly or unpredictably.
  - It generally has some structure, but does not conform to a fixed schema
  - "Schemaless" and self-describing, i.e., data carries information about its own schema (e.g., in terms of XML element tags)
- Characteristics
  - Heterogeneous
  - Irregular structure
  - Large evolving schema
- Major application: XML documents

# Semi-Structured Data:
# Graph Representation

- Collection of nodes
  - Atomic values on *leaf* nodes
  - *Interior* nodes have one or more arcs
- Nodes connected in a general rooted graph structure
- Labels on arcs
  - name of attribute/type
  - relationship
- Example: Beer-Bar-Manufacturer



# Extensible Markup Language (XML)

# Data Models & Database System Architectures
## - Chronological Overview -

- Network Data Models (1964)
- Hierarchical Data Models (1968)
- Relational Data Models (1970)
- Object-oriented Data Models (~ 1985)
- Object-relational Data Models (~ 1990)
- ***Semistructured Data Models (XML 1.0)*** ***(~1998)***

# Extensible Markup Language (XML)

- Standard of the World Wide Web Consortium (W3C) in 1998
- An XML document is only a file of characters
- Similar to HTML, but
  - HTML uses tags for *formatting* (e.g., "italic").
  - XML uses tags for *semantics* (e.g., "this is an address").
- Two modes:
  - *Well-formed XML* allows you to invent your own tags, much like labels in semi-structured data.
  - *Valid XML* involves a Document Type Definition (DTD) that tells the labels and gives a grammar for how they may be nested.

# XML:
# Tags

- Tags are text surrounded by brackets, i.e., <. . .>
- Tags come in matching pairs, e.g.,
  <FOO> is balanced by </FOO>
- Nesting allowed (start and end in same range), e.g.,
  <BAR> <NAME></NAME> </BAR>
- Unbalanced tags not allowed, e.g.,
  <P>, <BR>, and <HR> in HTML

# XML:
# Well-Formed XML

- Minimal requirement:
  XML declaration and root tags surrounding entire body

```
<? XML VERSION = "1.0" STANDALONE = "yes" ?>
<XXX>
        .....
</XXX>
```

***NOTE 1:***
XML version

***NOTE 2:***
there is no DTD specified

# XML:
# Well-Formed XML: 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>
                 ...
        </BAR>
</BARS>
```

***NOTE 1:***
only balanced tags

***NOTE 2:***
value between two surrounding tags

***NOTE 3:***
nesting within the same range

# XML:
# 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.
- Structure of a DTD:

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

- The root-tag is used to surround the document which uses these rules

# XML:
# 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.
- 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:
  1. `*` = zero or more of.
  2. `+` = one or more of.
  3. `?` = zero or one of.
- In addition: | = "or."

# XML:
# DTD: Example

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

***NOTE 1:***
BARS is root-tag

***NOTE 2:***
multiplicity of tags

***NOTE 3:***
name (and price) has a text value

***NOTE 4:***
Inside <BARS>-tag we'll find zero or more <BAR>-tags

***NOTE 5:***
a BAR has a name and serves one or more beers (which again has components)

# XML: Using a DTD

- To use a DTD, set `STANDALONE = "no"`:

```
<?XML VERSION = "1.0" STANDALONE = "no"?>
```

- Either
  - Include the DTD as a preamble, or
  - Follow the `XML` tag by a `DOCTYPE` declaration with the root tag, the keyword `SYSTEM`, and a file where the DTD can be found.

# XML:
# Using a DTD: Example

```
<?XML VERSION = "1.0" STANDALONE = "no"?>
<!DOCTYPE Bars [SYSTEM "bar.dtd">
   <!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>
```

***NOTE 1:***
DTD may be in a separate file

***NOTE 2:***
DTD may be included as a preamble

***NOTE 3:***
`BARS` is root-tag and surround the document which uses these rules

***NOTE 4:***
`BEER` has a name and a price

***NOTE 5:***
`BAR` has a name and serves one or more beers.

# XML:
# Attribute Lists

- Opening tags can have "arguments" that appear within the tag, in analogy to constructs like `<A HREF = . . .>` in HTML.
- Keyword `!ATTLIST` introduces a list of attributes and their types for a given element in the DTD.
- Example of declaration:

```
<!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 = "sports">
        . . .
</BAR>
```

# XML: ID's and IDREF's

- `ID` is used to give a unique name for an element/object
- `IDREF` is used to provide pointers to elements/object (by the `ID`-name), and multiple object references within one tag is allowed. `IDREFS` is used if there might be a set of references
- 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.

# XML:
# ID's and IDREF's: 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 MANUFACTURER (ADDR,...)>
        <!ATTLIST MANUFACTURER (name ID)>
    <!ELEMENT ADDR (#PCDATA)>
    <!ELEMENT BEER (NAME, PRICE)>
        <!ATTLIST BEER (manf IDREF)>
    <!ELEMENT PRICE (#PCDATA)>
]>
...
<MANUFACTURER name= ="X">...</MANUFACTURER>
...
<BEER manf="X"><NAME>Bud</NAME><PRICE>2.50</PRICE></BEER>
```

***NOTE 1:***
`MANUFACTURER` has a name-`ID`

***NOTE 2:***
`BEER` has a poiner to a manufacturer

***NOTE 3:***
The `IDREF` value in `BEER` equals the `ID` value in the corresponding `manufacturer`

# Summary

- Semi-structured data
- Extensible Markup Language (XML)