XML
Semistructured Data
Extensible Markup Language
Document Type Definitions
XPATH
XQUERY
Framework

1. *Information Integration* : Making databases from various places work as one.


The Information-Integration Problem

- Related data exists in many places and could, in principle, work together.
- But different databases differ in:
  1. Model (relational, object-oriented?).
  2. Schema (normalized/unnormalized?).
  3. Terminology: are consultants employees? Retirees? Subcontractors?
  4. Conventions (meters versus feet?).
Example

- Every bar has a database.
  - One may use a relational DBMS; another keeps the menu in an MS-Word document.
  - One stores the phones of distributors, another does not.
  - One distinguishes ales from other beers, another doesn’t.
  - One counts beer inventory by bottles, another by cases.
Two Approaches to Integration

1. **Warehousing**: Make copies of the data sources at a central site and transform it to a common schema.
   - Reconstruct data daily/weekly, but do not try to keep it more up-to-date than that.

2. **Mediation**: Create a view of all sources, as if they were integrated.
   - Answer a view query by translating it to terminology of the sources and querying them.
Warehouse Diagram

Warehouse

Wrapper

Source 1

Wrapper

Source 2
A Mediator

User query

Result

Query

Result

Query

Result

Wrapper

Wrapper

Source 1

Source 2
Semistructured Data

- Purpose: represent data from independent sources more flexibly than either relational or object-oriented models.
- Think of objects, but with the type of each object its own business, not that of its “class.”
- Labels to indicate meaning of substructures.
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

Notice a new kind of data.

The bar object for Joe’s Bar

The beer object for Bud

The beer object for Bud

Notice a new kind of data.
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), which limits the labels and gives a grammar for their use.
Well-Formed XML

- Start the document with a declaration, surrounded by `<? ... ?>`.
- Normal declaration is:
  ```xml
  <? XML VERSION = "1.0"
  STANDALONE = "yes" ?>
  ``
  “Standalone” = “no DTD provided.”
- Balance of document is a root tag surrounding nested tags.
Tags

- Tags, as in HTML, are normally matched pairs, as `<FOO>` ... `</FOO>`.
- Tags may be nested arbitrarily.
Example: Well-Formed XML

```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>
```
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.
Example

◆ The `<BARS>` XML document is:

```
<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>
  ...  // More bars...
</BARS>
```
Document Type Definitions

- Essentially a context-free grammar for describing XML tags and their nesting.

- Each domain of interest (e.g., electronic components, bars-beers-drinkers) creates one DTD that describes all the documents this group will share.
DTD Structure

<!DOCTYPE <root tag> [
  <!ELEMENT <name> ( <components> )
  <more elements>]
] >
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` in place of nested tags.
Example: DTD

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

A BARS object has zero or more BAR’s nested within.

A BAR has one NAME and one or more BEER subobjects.

NAME and PRICE are text.

A BEER has a NAME and a PRICE.
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.
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:

```xml
<!ELEMENT NAME (TITLE?, FIRST, LAST) | IPADDR>
```
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.
Example (a)

```xml
<?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> ... </BAR>
</BARS>
```

The DTD

The document
Example (b)

Assume the BARS DTD is in file bar.dtd.

```xml
<?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
Attributes

- Opening tags in XML can have attributes, like `<A HREF = "...">` in HTML.
- In a DTD,

  `<!ATTLIST <element name>... >`

gives a list of attributes and their datatypes for this element.
Example: Attributes

Bars can have an attribute \textit{kind}, which is either sushi, sports, or “other.”

\begin{verbatim}
<!ELEMENT BAR (NAME BEER*)>
<!ATTLIST BAR kind = "sushi" | "sports" | "other">
\end{verbatim}
Example: Attribute Use

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

```xml
<BAR kind = "sushi">  
 <NAME>Akasaka</NAME>  
 <BEER><NAME>Sapporo</NAME>  
   <PRICE>5.00</PRICE></BEER>  
 ...  
</BAR>
```
ID’s and IDREF’s

- These are pointers from one object to another, in analogy 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.
Creating ID’s

- Give an element, say $E$, an attribute, say $A$, of type ID.
- When using tag $<E>$ in an XML document, give its attribute $A$ a unique value.
- Example:

  $<E A = "xyz">$
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.

- Or, let the attribute have type IDREFS, so the $F$–object can refer to any number of other objects.
Example: ID’s and IDREF’s

- Let’s redesign our BARS DTD to include both BAR and BEER subelements.
- Both bars and beers will have ID attributes called name.
- Bars have PRICE subobjects, consisting of a number (the price of one beer) and an IDREF theBeer leading to that beer.
- Beers have attribute soldBy, which is an IDREFS leading to all the bars that sell it.
The DTD

```xml
<!DOCTYPE Bars [ 
<!ELEMENT BARS (BAR*, BEER*)>
<!ELEMENT BAR (PRICE+)>
<!ELEMENT PRICE (#PCDATA)>
<!ATTLIST BAR name = ID>
<!ATTLIST PRICE theBeer = IDREF>
<!ELEMENT BEER ()>
<!ATTLIST BEER name = ID, soldBy = IDREFS>
]> 
```

Bar objects have name as an ID attribute and have one or more PRICE subobjects.

PRICE objects have a number (the price) and one reference to a beer.

Beer objects have an ID attribute called name, and a soldBy attribute that is a set of Bar names.
Example Document

<BARS>
  <BAR name = "JoesBar">
    <PRICE theBeer = "Bud">2.50</PRICE>
    <PRICE theBeer = "Miller">3.00</PRICE>
  </BAR>
  ...
  <BEER name = "Bud",
    soldBy = "JoesBar, SuesBar,...">
  </BEER>
  ...
</BARS>
XPATH and XQUERY

- XPATH is a language for describing paths in XML documents.
  - Really think of the semistructured data graph and its paths.

- XQUERY is a full query language for XML documents with power similar to OQL.
Example DTD

<!DOCTYPE Bars [
  <!ELEMENT BARS (BAR*, BEER*)>
  <!ELEMENT BAR (PRICE+)>
    <!ATTLIST BAR name = ID>
  <!ELEMENT PRICE (#PCDATA)>
    <!ATTLIST PRICE theBeer = IDREF>
  <!ELEMENT BEER ()>
    <!ATTLIST BEER name = ID, soldBy = IDREFS>
]>


Example Document

<BARS>
  <BAR name = “JoesBar”>
    <PRICE theBeer = “Bud”>2.50</PRICE>
    <PRICE theBeer = “Miller”>3.00</PRICE>
  </BAR> ...
  <BEER name = “Bud”, soldBy = “JoesBar, SuesBar,...”>
    ...
  </BEER> ...
</BARS>
Path Descriptors

- Simple path descriptors are sequences of tags separated by slashes (/).
- If the descriptor begins with /, then the path starts at the root and has those tags, in order.
- If the descriptor begins with ///, then the path can start anywhere.
Example: /BARS/BAR/PRICE

<BARS>
  <BAR name = “JoesBar”>
    <PRICE theBeer = “Bud”>2.50</PRICE>
    <PRICE theBeer = “Miller”>3.00</PRICE>
  </BAR> ...
  <BEER name = “Bud”, soldBy = “JoesBar, SuesBar,...”>
  </BEER> ...
</BARS>

/BARS/BAR/PRICE describes the set with these two PRICE objects as well as the PRICE objects for any other bars.
Example: //PRICE

<BARS>
  <BAR name = “JoesBar”>
    <PRICE theBeer = “Bud”>2.50</PRICE>
    <PRICE theBeer = “Miller”>3.00</PRICE>
  </BAR> ...

  <BEER name = “Bud”, soldBy = “JoesBar, SuesBar,...”>
    ...
  </BEER> ...
</BARS>

//PRICE describes the same PRICE objects, but only because the DTD forces every PRICE to appear within a BARS and a BAR.
Wild-Card *

- A star (*) in place of a tag represents any one tag.

- Example: /*/*/PRICE represents all price objects at the third level of nesting.
Example: /*BARS*/

```xml
<BARS>
  <BAR name = "JoesBar">
    <PRICE theBeer = "Bud">2.50</PRICE>
    <PRICE theBeer = "Miller">3.00</PRICE>
  </BAR> ...
  <BEER name = "Bud", soldBy = "JoesBar, SuesBar,...">
  </BEER> ...
</BARS>
```

/*BARS*/ captures all BAR and BEER objects, such as these.
Attributes

- In XPATH, we refer to attributes by prepending @ to their name.

- Attributes of a tag may appear in paths as if they were nested within that tag.
Example: /BARS/*/@name

```xml
<BARS>
  <BAR name="JoesBar">
    <PRICE theBeer="Bud">2.50</PRICE>
    <PRICE theBeer="Miller">3.00</PRICE>
  </BAR>
  ...
  <BEER name="Bud", soldBy="JoesBar,
  SuesBar,...">
  </BEER>
  ...
</BARS>
```

/BARS/*/@name selects all name attributes of immediate subobjects of the BARS object.
Selection Conditions

- A condition inside [...] may follow a tag.

- If so, then only paths that have that tag and also satisfy the condition are included in the result of a path expression.
Example: Selection Condition

◆ /BARS/BAR/PRICE[PRICE < 2.75]

<BARS>
  <BAR name = “JoesBar”>
    <PRICE theBeer = “Bud”>2.50</PRICE>
    <PRICE theBeer = “Miller”>3.00</PRICE>
  </BAR> ...

The condition that the PRICE be < $2.75 makes this price but not the Miller price satisfy the path descriptor.
Example: Attribute in Selection

◆ /BARS/BAR/PRICE[@theBeer = “Miller”]

<BARS>
  <BAR name = “JoesBar”>
    <PRICE theBeer = “Bud”>2.50</PRICE>
    <PRICE theBeer = “Miller”>3.00</PRICE>
  </BAR> ...
</BARS>

Now, this PRICE object is selected, along with any other prices for Miller.
Axes

◆ In general, path expressions allow us to start at the root and execute a sequence of steps to find a set of nodes at each step.
◆ At each step, we may follow any one of several axes.
◆ The default axis is `child::` --- go to any child of the current set of nodes.
Example: Axes

◆ /BARS/BEER is really shorthand for /BARS/child::BEER.
◆ @ is really shorthand for the attribute::axis.
  • Thus, /BARS/BEER[@name = "Bud"] is shorthand for /BARS/BEER[attribute::name = "Bud"]
More Axes

- Some other useful axes are:
  1. parent:: = parent(s) of the current node(s).
  2. descendant-or-self:: = the current node(s) and all descendants.
     - Note: // is really a shorthand for this axis.
  3. ancestor::, ancestor-or-self, etc.
XQUERY

- XQUERY allows us to query XML documents, using path expressions from XPATH to describe important sets.
- Corresponding to SQL’s select-from-where is the XQUERY FLWR expression, standing for “for-let-where-return.”
FLWR Expressions

1. One or more FOR and/or LET clauses.
2. Then an optional WHERE clause.
3. A RETURN clause.
FOR Clauses

FOR <variable> IN <path expression>,...

- Variables begin with $.
- A FOR variable takes on each object in the set denoted by the path expression, in turn.
- Whatever follows this FOR is executed once for each value of the variable.
Example: FOR

FOR $beer$ IN /BARS/BEER/@name
RETURN

$beer$ ranges over the name attributes of all beers in our example document.

Result is a list of tagged names, like

$<BEERNAME>Bud</BEERNAME>$
$<BEERNAME>Miller</BEERNAME>$...
LET Clauses

LET <variable> := <path expression>,...

◆ Value of the variable becomes the set of objects defined by the path expression.

◆ Note LET does not cause iteration; FOR does.
Example: LET

LET $beers := /BARS/BEER/@name
RETURN
  <BEER NAMES>$beers</BEER NAMES>

\* Returns one object with all the names of the beers, like:

<BEER NAMES>Bud, Miller,...</BEER NAMES>
Following IDREF’s

◆ XQUERY (but not XPATH) allows us to use paths that follow attributes that are IDREF’s.

◆ If $x$ denotes a set of IDREF’s, then $x => y$ denotes all the objects with tag $y$ whose ID’s are one of these IDREF’s.
Example

- Find all the beer objects where the beer is sold by Joe’s Bar for less than 3.00.

- Strategy:
  1. $beer will for-loop over all beer objects.
  2. For each $beer, let $joe be either the Joe’s-Bar object, if Joe sells the beer, or the empty set of bar objects.
  3. Test whether $joe sells the beer for < 3.00.
Example: The Query

FOR $beer IN /BARS/BEER
LET $joe := $beer/@soldBy=>BAR[@name="JoesBar"]
LET $joePrice := $joe/PRICE[@theBeer=$beer/@name]
WHERE $joePrice < 3.00
RETURN <CHEAPBEER>$beer</CHEAPBEER>

Attribute soldBy is of type IDREFS. Follow each ref to a BAR and check if its name is Joe’s Bar.

Find that PRICE subobject of the Joe’s Bar object that represents whatever beer is currently $beer.

Only pass the values of $beer, $joe, $joePrice to the RETURN clause if the string inside the PRICE object $joePrice is < 3.00